THE DEVELOPMENT OF A NOVEL MODEM STRUCTURE FOR CONNECTION OF RURAL TO DIGINET.

Thesis prepared by : Timothy D. Courtenay

prepared for : Dr. Robin Braun
Senior Lecturer,
Department of Electrical and
Electronic Engineering,
University of Cape Town.

October 1990

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Electronic Engineering at the University of Cape Town.

To my parents, Rob and Jenny. I greatly appreciate the opportunities you have given me.

DECLARATION

I declare that this thesis is my own unaided work. This thesis has not previously been submitted, in part or in full, to any university for any degree.

Signed signature removed

Timothy D. Courtenay
6th October 1990

ACKNOWLEDGEMENTS

There are a number of people who gave me invaluable assistance throughout the duration of this thesis. I would like to thank:-

Dr. Robin Braun, for his supervision and guidance since the start of this thesis.

Steve Schrire, UCT's guru of practical electronics, for excellent advice and assistance.

Les Devonish of STC for coordinating funding for the project.

Russel Horwitz, for his continuous support and constructive criticism.

TERMS OF REFERENCE

This thesis was requested by Dr. Robin Braun of the Department of Electrical and Electronic Engineering at U.C.T. in January 1989. The aim of the thesis was to investigate a novel modem structure for linking rural locations to the Diginet network. Dr. Braun's specific instructions were:-

1. To implement partial response signalling as a modulation technique in a modem structure.

2. To design the transmitted signal for transmission over M.1020 telephone-type lines.

3. To design the modem for transmission rates of 2400, 4800 and 9600 bits/sec.

4. To limit the scope of the thesis to the investigation of the modulation/demodulation scheme only, and not include the investigation of adaptive equalization, protocol control and other aspects of modem design.

5. To complete the investigation within a two year period.

SYNOPSIS

This thesis investigates the use of partial response signalling as a modulation scheme in a modem structure. The modem structure consists of transmitter modulation and receiver demodulation sections only. The modem is designed to operate at data rates of 2400, 4800 and 9600 bps. The signalling format replaces the CCITT Recommendation V.29 format. The transmitted signal is required to conform to the bandwidth limitations of CCITT Recommendation M.1020 leased telephone circuits.

The data rate that can be achieved over a communication channel is directly dependant on the usable bandwidth of the channel. Often modems intended for use over bandlimited channels such as telephone lines are designed to transmit at as high a data rate as possible, given the available bandwidth. Such modems are said to be 'spectrally efficient'. The theoretical spectral, or bandwidth, efficiency of a digital modulation system is 2 symbols/sec/Hz.

Class 1 partial response signalling (PRS) was chosen as a modulation technique for the modem due to its excellent signal spectral properties. This technique uses correlative coding methods to allow the maximum theoretical bandwidth efficiency of a digital communication system to be reached. Conventional modulation techniques such as quadrature amplitude modulation (QAM) cannot achieve this bandwidth efficiency. PRS modulation schemes provide better performance than QAM when operated near the limit of bandwidth efficiency. The modulation techniques corresponding to data rates of 2400, 4800 and 9600 bps are 3-PRS, 9-QPRS and 49-QPRS respectively (QPRS - quadrature partial response signalling).

The modem design is based on the use of the Motorola DSP56001 signal processor, and is both software and

hardware-oriented. The transmitter and receiver sections each utilize a DSP56001 processor to perform the majority of the signal processing functions. These functions include data coding and decoding, lowpass filtering, carrier modulation and demodulation, and pilot tone addition. The design did not include adaptive equalization and phase-locked loop timing recovery, due to the time limitations on the project. An important aspect of the digital design implementation was that of digital filtering. A square-root raised-cosine filter response was chosen for both the transmitter and receiver lowpass filters. This filter response is particularly well suited to this application.

Measurements were taken of the transmitted and received signal spectra and time-domain waveforms (eye patterns). It was found that the use of a digital architecture allowed very precise and deterministic signal characteristics to be achieved. The use of various root raised-cosine filters was investigated, and it was found that the case of 10% excess bandwidth gave excellent time and frequency domain performance. The observed signals showed minimal inherent intersymbol interference, and conformed to the frequency limits of M.1020 lines.

Measurements of the bit error rate versus signal-to-noise ratio performance were taken for all three operational bit rates. It was found that the measured performance of the system was close to the theoretical predictions. For a given bit error rate, the transmitted signal required between 2 and 5dB more signal power than theoretically calculated. The degradation was more severe as the bit rate increased. The system showed no measurable error floor.

The results are very promising and suggest that further investigation and development of the modem structure could prove to be very fruitful. It is recommended that future work should be directed at the development of a receiver adaptive equalizer and synchronization recovery circuit. To

improve the error performance, optimal decoding techniques should be used at the receiver. A digital design approach should be maintained throughout in further design work.

TABLE OF CONTENTS PAGE

LIST OF FIGURES PAGE

LIST OF TABLES PAGE

CHAPTER 1. INTRODUCTION

The Diginet system was developed in South Africa to provide a dedicated communications network for subscribers [1.1]. This system operates independently of the switched telephone network, and is designed specifically for point-to-point and point-to-multipoint digital transmission. In urban centres this facility enables digital transmission of up to 64 kilobits/second (kbps), and bypasses the need for analogue modems (modulator-demodulator). However, the connection of distant rural areas to Diginet requires an analogue telephone-type link. This is because it is not economically viable to install a long digital link for low-volume data traffic. The rural location might be a private farm or a small community. Data communication over the analogue rural link therefore requires the use of modems.

The factor that limits the rate of data transfer over telephone lines is the frequency transfer characteristic of the line. The CCITT Recommendation M.1020 specifies the worst-case limits for amplitude and group delay distortion that may occur over special quality leased line circuits [1.2]. These characteristics are shown in Appendix I. M.1020 lines, which are used for the rural links in Diginet, typically have a usable bandwidth of 300 - 3000Hz. The amplitude and group delay distortion that occurs in these lines sometimes demands the use of adaptive equalization. A modem that operates over such a link must therefore cater for the channel characteristics.

In the case of data communication over telephone lines, it is desirable to design a 'spectrally efficient' modem that can achieve high data rates (above 4800 bps) in the given bandwidth. A modem is said to be spectrally efficient if it achieves a bandwidth efficiency of greater than 2 bits/sec/Hz. This is in contrast to 'power efficient' modems that are designed to allow data transmission in noisy channels, and operate at bandwidth efficiencies of less than

2 bits/sec/Hz. Conventional modem designs aim for spectral efficiency by using multilevel memoryless modulation techniques. For example, the CCITT Recommendations V.26, V.27 and V.29 that were initiated in 1968 [1.3] specify various formats of phase shift keying (PSK) as modulation techniques for data transmission rates of up to 9600 bps.

The maximum theoretical bandwidth efficiency that a digital communication system can reach is 2 symbols/sec/Hz. Conventional memoryless modulation schemes, however, cannot operate at this limit, and often operate at closer to 1 symbol/sec/Hz.

Partial response signalling (PRS) is a coding technique invented by Lender [1.4] that allows the maximum bandwidth efficiency of 2 symbols/sec/Hz to be reached. The technique introduces correlation between successive data symbols to achieve convenient spectral shaping. Consequently, when operated at close to 2 symbols/sec/Hz, PRS can provide better error performance than memoryless systems.

The aim of this thesis is to investigate partial response signalling as a modulation technique in a modem design. The modem is intended for use in rural links to Diginet over 4-wire M.1020 telephone lines. The modem is to operate at data rates of 2400, 4800, and 9600 bps in full-duplex mode. The bandwidth occupancy of the transmitted signal is to be approximately 2400Hz, i.e. the modem will operate at a maximum spectral efficiency of 9600/2400 = 4 bits/sec/Hz. The system performance over a linear channel is investigated. The use of adaptive equalization and testing over M.1020 lines is omitted, as it is beyond the scope of the project.

A modem design incorporates many aspects other than the modulation and demodulation technique. For it to be operational and marketable, a modem must include adaptive equalization, protocol control, scrambling capabilities, computer interfacing equipment etc. This scope of this thesis is to investigate the modulation and demodulation technique only.

The thesis report first develops the background theory that is relevant to the project, and then presents the experimental work done in the project. Chapter 2 gives a review of pertinent digital communications and digital signal processing concepts. Chapter 3 describes the theory of partial response signalling and emphasizes its advantages over conventional data transmission techniques. Chapter 4 gives an outline of the proposed modem structure, with particular attention given to the use of a digital architecture. Chapters 5 and 6 discuss the modem transmitter and receiver designs. Both the software and hardware components are described. Experimental results and measurements of the modem performance are shown in Chapter 7. In Chapter 8, conclusions are drawn based on these results, and recommendations for future work are made. A summary of the work done on the modem transmitter structure can be found in the published paper in Appendix K.

The research procedure of this project is outlined by the flowchart in Figure 1.1.

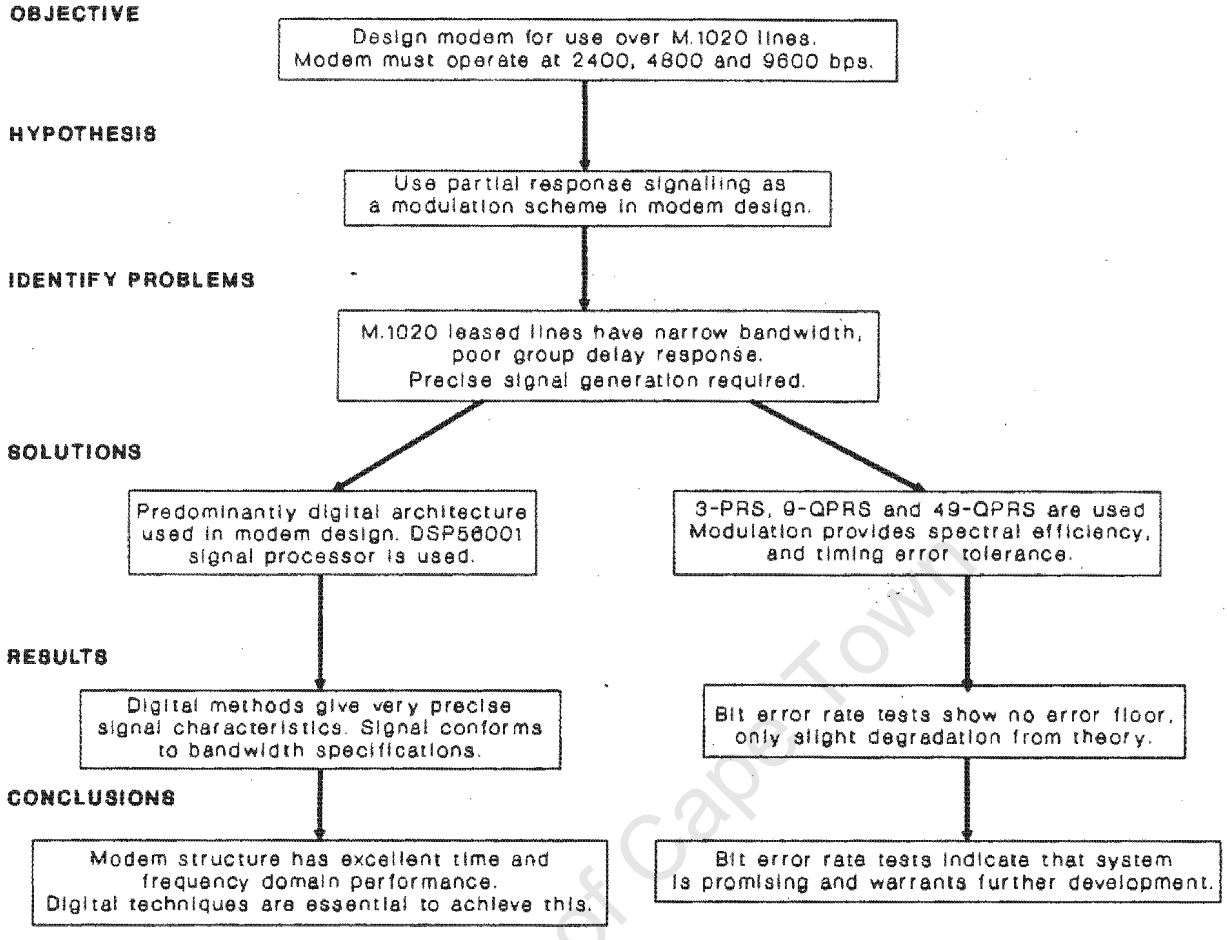


Figure 1.1 - Research phases of this project.

REFERENCES

[1.1] Diginet Service and Operating Guide, Issue 1, 1st Edition, Department of Posts and Telecommunications.

[1.2] CCITT Recommendation *Characteristics of special quality international leased circuits, with special bandwidth conditioning*, vol. IV, fascile IV.2, Rec. M.1020.

[1.3] CCITT Recommendation *9600 bits per second modem standardized for use on point-to-point 4-wire leased telephone-type circuits*, fascile VIII.1, Rec. V.29.

[1.4] A. Lender, 'The Duobinary Technique for High-Speed Data Transmission', IEEE Trans. Commun. Electron., vol. 82, pp.214-218, May 1963.

CHAPTER 2. REVIEW OF DIGITAL COMMUNICATIONS AND SIGNAL PROCESSING

2.1 INTRODUCTION

The term 'digital communications' refers to the transfer of digital information over a communications link. The information may be directly from a digital source, e.g. a computer, or it may be the quantized result of sampling an analogue signal. The information transfer is achieved by transmitting symbols from a finite-length alphabet. For example, a binary system can only transmit a sequence of 1s and 0s, since its alphabet only consists of the symbols 1 and 0.

A digital system therefore transmits a finite amount of information, whereas an analogue system transmits an infinite amount of information. The receiver of a digital system is concerned only with determining which of the symbols was transmitted, whereas in an analogue system the receiver attempts to reproduce the original waveform accurately.

Figure 2.1 shows a block diagram of the signal processing steps that might be found in a typical digital communications system. The functions performed by the various blocks are:-

Formatting - this converts the signal (if necessary) into a format compatible with digital signal processing. This may involve sampling, quantizing, etc.

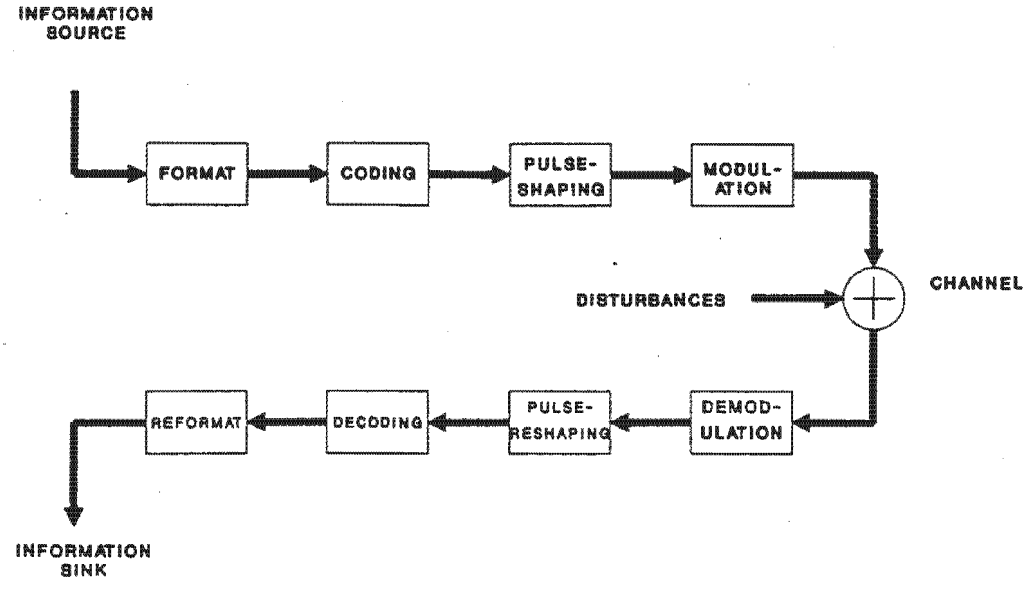


Figure 2.1 - A typical communication system (from [2.1]).

Coding - this refers to coding of the digital data in such a way as to enhance certain signal parameters, such as bit error rate, bandwidth, secrecy and so on. An example is partial response signalling.

Pulse Shaping - this is sometimes necessary to either limit the bandwidth occupancy of the signal or to counteract in advance the distortion properties of the channel. In its simplest form it is lowpass filtering.

Modulation - here the signal (at this stage either digital or analogue) is used to modulate a carrier of suitably high frequency. Amplitude, frequency or phase modulation can be used.

Demodulation - this usually involves coherent recovery of the information that was used to modulate the carrier in the previous stage.

Pulse reshaping - often the pulse shaping steps are divided equally between the transmitter and the receiver. This step may involve lowpass filtering to recover the transmitted baseband information.

Decoding - this recovers the encoded information symbols.

Reformatting - if necessary convert the signal back to analogue format.

The above operations are not necessarily always distinct, and some overlapping may occur. This chapter reviews some of the aspects of digital communications and digital signal processing that are relevant to this project.

2.2 SAMPLING AND QUANTIZATION

2.2.1 SAMPLING IN THE TIME DOMAIN

The sampling theorem states the following:

> A lowpass signal f(t) that is bandlimited in that it has no frequency components higher than some frequency f_h, is completely described by its values at equally spaced points in time separated by $T_S \leq 2/f_h$.

The minimum sampling frequency $f_S = 1/T_S$ is called the Nyquist frequency. The action of sampling a signal at a certain frequency can be thought of as follows:-

1. The original analogue signal is multiplied by a Dirac impulse train of amplitude 1, with impulses separated by T_S.

2. Multiplication in the time domain is analogous to convolution in the frequency domain. The Fourier transform of a train of Dirac impulses separated by T_s in the time domain is an infinite set of discrete frequencies spaced at intervals $1/T_s$ in the frequency domain.

3. The resulting frequency convolution is a set of replicated signal spectrums with centre frequencies separated by $1/T_s$. In theory this repetition extends to $\pm \infty$.

These steps are shown graphically in Figure 2.2.

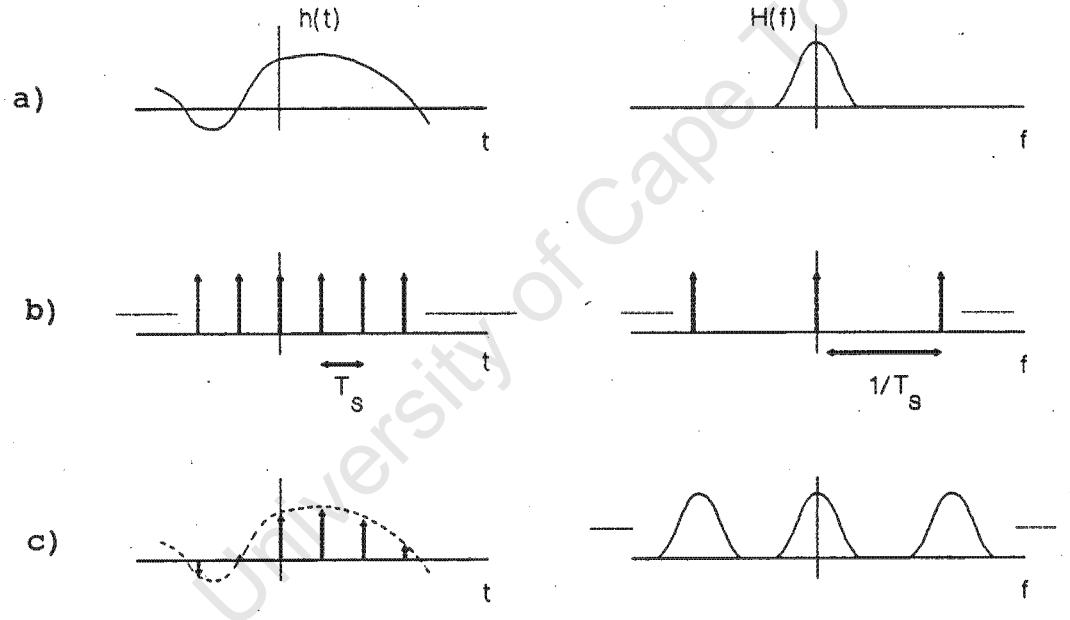


Figure 2.2 - The sampling of a lowpass signal.
a) The signal and its spectrum.
b) The Dirac impulse train and its spectrum.
c) The result of multiplying in the time domain and convolving in the frequency domain.

We now have a set of impulse values in the time domain that contain all the information of the original signal. There are two important implications of this fact. Firstly, by storing these impulse sample values in the form of numbers,

we can perform numerical signal processing tasks on the numbers. This is the essence of digital signal processing. Secondly, if the sample impulse values are generated first, the correct analogue signal can be extracted by lowpass filtering the sampled signal. Both these factors are fundamental to this project.

If a signal is sampled at less than the Nyquist frequency a phenomenon called **aliasing** occurs. This is shown in Figure 2.3. As the sampling frequency decreases, the spacing between the replicated signal spectrums decreases until, at the Nyquist sampling frequency, overlap occurs between adjacent spectra. This causes signal distortion, and the true signal can not be recovered. Aliasing is thus avoided by bandlimiting the signal and then sampling at greater than the Nyquist frequency.

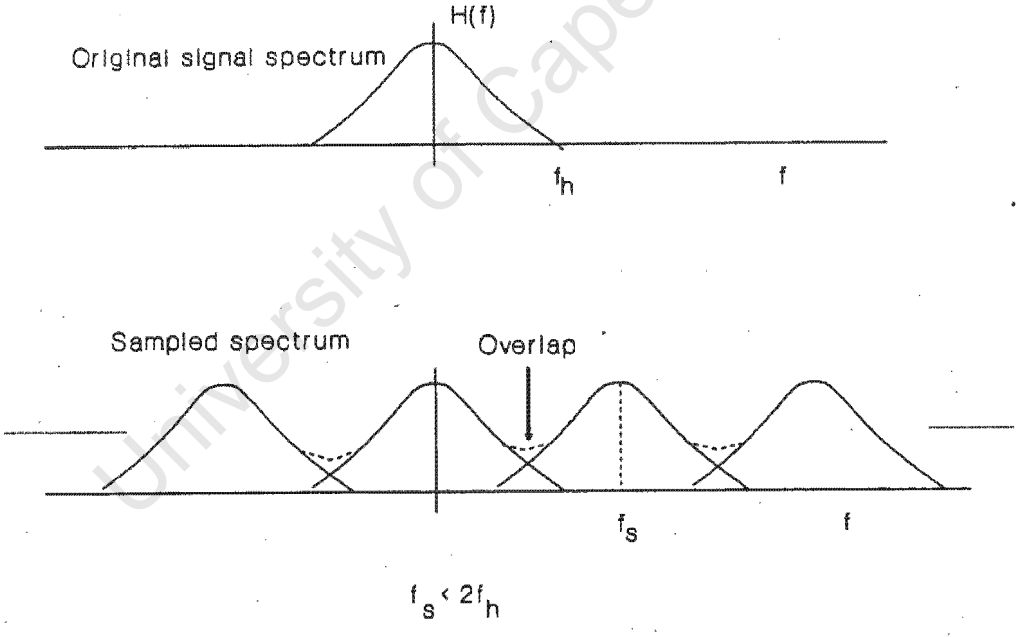


Figure 2.3 - Aliasing effects in the frequency domain.

2.2.2 QUANTIZATION

Often the first step in dealing with a sampled signal is to quantize the sampled signal values by assigning an n-bit binary number to them. The values can thus be processed in a digital system, e.g. a computer. The term quantization arises from the fact that only a discrete number of signal

levels can be described by a given number of bits. For an analogue signal, an infinite number of levels exist, so quantizing such a signal into a finite number of levels will result in some form of error. This error manifests itself as a form of 'noise' called **quantization noise**, and can be thought of as noise added to the original signal to give the quantized level. A derivation of an expression for quantization noise is given in [2.2], and the result is stated here.

For **n** possible quantized levels, with a spacing of **a** between adjacent levels and each level equiprobable, the mean-square signal-to-quantization noise ratio is

$$S/N = n^2 - 1$$

This term is dependent only on the number of quantization levels. For 8-bit quantization, **S/N** = 48.2 dB, and for each added bit of resolution the signal-to-quantization noise ratio increases by about 6 dB. The important point is that the bit-resolution of a digital system, such as this project, should be chosen so that this ratio is high enough to have minimal effect on the overall signal-to-noise ratio performance of the communication system.

2.3 CHANNEL CHARACTERISTICS

The transmission channel is the most crucial element in a communication link. It limits the information capacity of a system by bandlimiting, amplitude and phase distorting, and adding noise to the transmitted signal. Consider for example the amplitude and phase response of the M.1020 line, given in Appendix I, which applies in the case of this project. A communication channel might have a narrow bandwidth (e.g. telephone lines) , or it might be very noisy (meteor burst link) etc. The communication system must then tailor the

transmission method or modulation scheme to best suit the channel.

The channel capacity is defined as the maximum rate of information that a channel can carry. It is a definition that was first formularized by C.E. Shannon [2.2]. The well-known Hartley-Shannon theorem gives an expression for the channel capacity **C** as follows:-

$$\mathbf{C} = \mathbf{B} \log_2 (1 + \mathbf{S/N}) \text{ bits/second.}$$

where **B** is the channel bandwidth
S/N is the mean-square signal to noise ratio.

The practical limits on information transfer through a channel are more severe, however, and Shannon's limit is never reached in practice.

2.4 BASEBAND DIGITAL COMMUNICATION

2.4.1 INTRODUCTION

It is sometimes necessary to transmit data over bandlimited channels, again an example being the connection of computer equipment over telephone lines. A binary (or multilevel) data stream (also referred to as a symbol stream), however, has a signal spectrum that extends up to infinity. It must therefore be bandlimited to allow transmission over the channel. Transmitting a lowpass filtered data stream is referred to as 'baseband digital communication'. The lowpass signal from the transmitter must be tailored so that the receiver can correctly detect the transmitted symbol sequence. The following sections describe the aspects of baseband digital communication that apply to this thesis.

2.4.2 NYQUIST'S CRITERIA FOR INTERSYMBOL INTERFERENCE-FREE TRANSMISSION

Intersymbol interference (ISI) is one of the most common factors that degrades a digital communication system. It describes the interference that can arise at the symbol sampling instants in a transmitted or received waveform. Many digital communications systems rely on the use of a lowpass filter to bandlimit the digital waveform prior to transmission. Ideally this lowpass filter would have an impulse response that has evenly spaced zero-crossings. These zero-crossings are fundamental to the operation of the system, as this is where the symbol sampling occurs. If the lowpass filter response does not comply with certain conditions, however, the impulse response zero-crossings occur at uneven intervals. This is known as intersymbol interference. Nyquist [2.3] developed two well-known theorems concerning the elimination of ISI in baseband pulse amplitude modulated (PAM) systems. A good summary of these two theorems is given by Feher [2.4]:

NYQUIST'S FIRST THEOREM - If an impulse train with symbols occurring at a rate of f_s is applied to an ideal 'brick-wall' lowpass, linear phase filter with a cutoff frequency $f_s/2$ Hz, then sampling the response at rate f_s, at the correct instants, will yield the correct ISI-free impulse values.

NYQUIST'S SECOND THEOREM - ISI-free transmission can also be achieved by the use of a lowpass filter that has a real-valued transmit function $X(w)$ and is odd-symmetrical about the cutoff frequency $w_n = f_s/2$. The symmetry about this frequency is defined as

$$X(w_n - x) = X(w_n + x) \qquad 0 < x < w_n$$

Figure 2.4 shows the transfer function and impulse response of a 'brick-wall' filter. Note that the zero-crossings of the impulse response occur at spacing $T_S = 1/2f_S$, except at the impulse maximum. This allows ISI-free transmission at a rate f_S.

The 'brick-wall' filter is a theoretical model and is impossible to realize, as it would require an infinite number of filter sections to attain an infinitely sharp cutoff. A filter characteristic that satisfies Nyquist's second requirement, however, is the raised-cosine filter transfer function. This is much easier to approximate in practice than the 'brick-wall' filter.

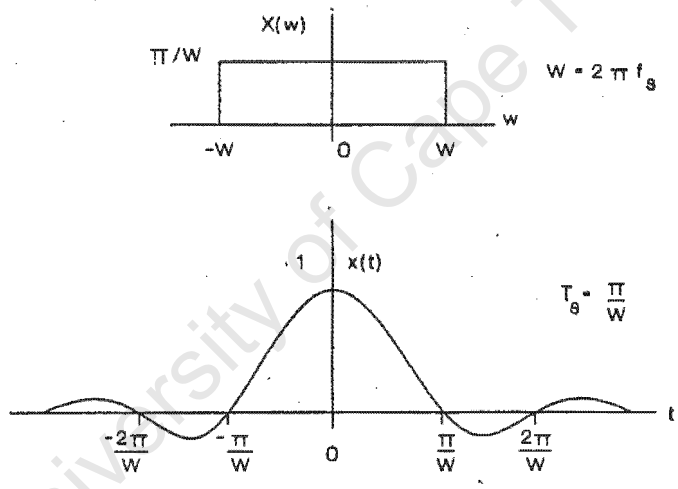


Figure 2.4 - Ideal lowpass filter frequency characteristic and impulse response.

The frequency transfer function for the raised-cosine filter is [2.5]:

$$X(w) = \begin{cases} T_S & \text{for } 0 \le |w| \le (1-\alpha)W \\ 0.5\,T_S\,(\,1 - \sin[\,(\pi/2\alpha W)(|w|-W)\,]\,) & \text{for } (1-\alpha)W \le |w| \le (1+\alpha)W \\ 0 & \text{for } |w| > (1+\alpha)W \end{cases}$$

where $W = \pi / T_S$.

The term α is the excess bandwidth used divided by the minimum Nyquist bandwidth, and is sometimes called the 'roll-off factor'. $\alpha = 0$ corresponds to the 'brick-wall' filter. The corresponding impulse response for the raised-cosine transfer function is:

$$x(t) = \left[\frac{\sin Wt}{Wt}\right] \left[\frac{\cos \alpha Wt}{1 - (2\alpha Wt/\pi)^2}\right]$$

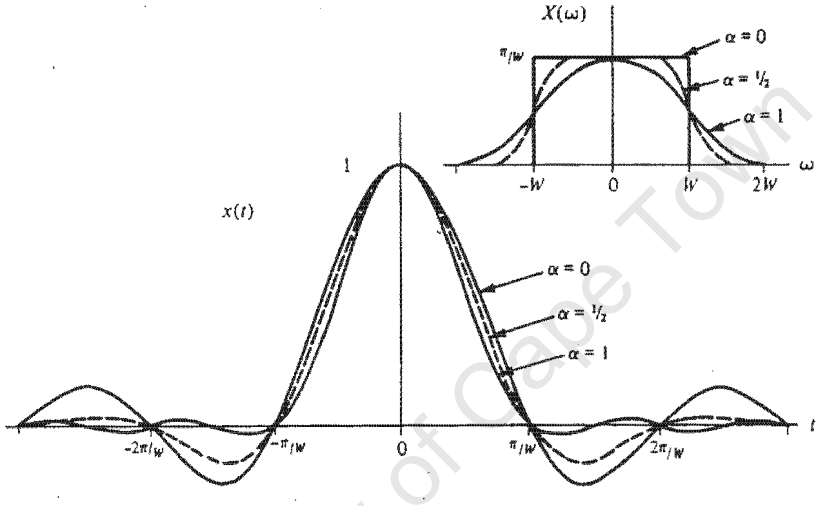


Figure 2.5 - Raised cosine transfer function and impulse response for α = 0, 0.5 and 1.

The transfer function and impulse response for the case of α = 0, 0.5 and 1 is shown in Figure 2.5. Note that the zero crossings of the impulse response are still at multiples of T_S.

2.4.3 PULSE AMPLITUDE MODULATION

The simplest form of baseband digital communication is pulse amplitude modulation (PAM). Consider a lowpass filter with cutoff frequency f_S. As shown in the previous section the impulse response of this filter is a sin(x)/x function. The impulse response has zero-crossings at intervals $T_S = 1/f_S$, except at t=0 where the amplitude is that of the driving impulse. If a train of impulses spaced at intervals T_S is

passed through this filter, the resulting baseband signal is a summation of their corresponding impulse responses. At intervals T_S, however, the resulting signal will have an amplitude dependent only on one impulse value. If the signal is sampled at these intervals, the original impulse train is recovered. Figure 2.6 a) shows a train of 3 symbol impulse values. Figure 2.6 b) shows the timing relationship between the impulse responses, and c) shows the recovered impulse values when the signal is sampled at the correct instants.

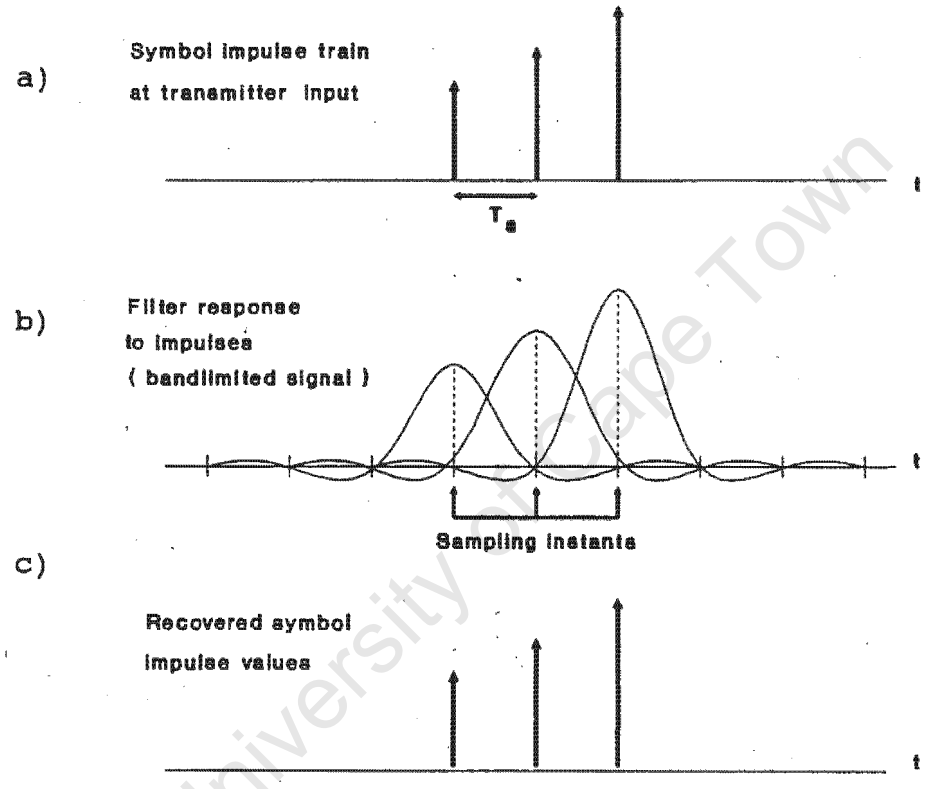


Figure 2.6 - Steps in creating a PAM signal.

Usually a communication system will have lowpass filters at the transmitter and the receiver. The filter transfer function is then divided equally between the transmit and receive filters, so each will have the square-root of the overall response. This yields minimum ISI at the output of the receiver lowpass filter, where symbol detection occurs. It is also critical that the sampling instants are correct, as any deviation away from the zero-ISI points will degrade the system performance. PAM systems are 'memoryless' systems in that there is no correlation between successive symbols in the transmitted data.

2.4.4 BANDWIDTH EFFICIENCY

The bandwidth efficiency of a baseband transmission system is often an important factor in implementing a modulation scheme. Nyquist's first theorem shows that a maximum data rate of $1/T_S$ symbols/sec can be transmitted in a bandwidth of $1/2T_S$ Hz. The maximum theoretically attainable baseband bandwidth efficiency is therefore 2 symbols/sec/Hz. However, two factors degrade the bandwidth efficiency in PAM systems. Firstly, it is impossible to construct a true 'brick-wall' filter, and therefore filters of wider bandwidth have to be employed. Secondly, PAM systems are very sensitive to timing deviations when a filter with a cutoff near the Nyquist frequency is used, and often therefore raised-cosine filters of up to 100% excess bandwidth are employed. The true operational symbol bandwidth efficiency of a PAM system is thus closer to 1 symbol/sec/Hz.

By using multilevel transmission, the bandwidth efficiency of a digital transmission system can be increased. For example, if a binary or M = 2 level signal has a practical bandwidth efficiency of 1 bit/sec/Hz, we can combine successive bits into 2-bit or M = 4 level signals, thus doubling the bit bandwidth efficiency to 2 bits/sec/Hz. The use of multilevel signalling is called **M-ary** signalling. The penalty to pay for this is that M-ary signalling requires a higher signal-to-noise ratio to achieve the same bit error-rate performance as an equivalent-bandwidth binary signal.

2.5 BANDPASS DIGITAL COMMUNICATION

2.5.1 INTRODUCTION

The channel in a communication system may have a bandpass frequency characteristic (such as a telephone line). In this situation it is necessary to modulate a carrier frequency with digital information to enable transmission. Typically the carrier frequency is placed in the centre of the

bandpass characteristic of the channel, allowing full utilization of the channel. There are three basic methods of digital modulation from which all other methods are derived. These are:-

1. **Amplitude modulation** - the amplitude of a carrier is either switched between two (binary) or more (M-ary) values in response to the digital information, or modulated by a baseband PAM signal.

2. **Frequency shift keying (FSK)** - the frequency of the carrier is switched between two or more values and bandpass filtered.

3. **Phase shift keying (PSK)** - the phase of the carrier is switched between two or more values and bandpass filtered.

This project concerns the amplitude modulation of a carrier by a baseband signal, and therefore only this technique will be discussed.

2.5.2 AMPLITUDE MODULATION (AM)

This modulation technique involves multiplying (modulating) a carrier frequency by a baseband digital signal. Consider a baseband 4-level PAM signal f(t). This signal can be used to modulate a carrier cos(wt) to produce the signal

$$g(t) = f(t)\cos(wt)$$

If g(t) is multiplied by cos(wt) again the result is

$$f'(t) = g(t)\cos(wt)$$

$$= 0.5\, f(t) + 0.5\, f(t)\cos(2wt)$$

The result is therefore the original baseband signal plus a component at twice the carrier frequency. Lowpass filtering the signal extracts the original baseband signal. It is possible to modulate a quadrature (90° phase shifted) carrier of the same frequency, i.e. sin(wt), by another signal h(t). If h(t) and f(t) are real-valued signals, the two modulated signals can be added together and transmitted in the same frequency band. The transmitted signal is thus

$$g(t) = f(t)\cos(wt) + h(t)\sin(wt).$$

Multiplying the composite signal by cos(wt) and sin(wt) at the receiver will extract the two signals f(t) and h(t) independently. This technique is known as quadrature AM, or QAM.

A baseband digital signal has a maximum theoretical bandwidth efficiency of 2 symbols/sec/Hz. If this signal is modulated onto a carrier, a double-sideband signal results, degrading the maximum bandwidth efficiency to 1 symbol/sec/Hz. However, if two such modulated signals are added in quadrature as described above, the theoretical maximum bandwidth efficiency is restored to 2 symbols/sec/Hz. The terminology used in this technique of digital modulation is as follows:- a baseband PAM signal having 4 possible symbol levels is referred to as 4-ary PAM. Modulating two such signals in quadrature therefore gives a signal with 16 possible states. This signal is referred to as a 16-QAM signal. Similarly, a PSK signal with 16 possible states is referred to as 16-PSK.

2.6 REFERENCES

[2.1] B. Sklar, 'A Structured Overview of Digital Communications-A Tutorial', IEEE Communications Magazine, pp. 4-17, August 1983.

[2.2] F. Stremler, *Introduction to Communication Systems, 2nd Edition*, Addison-Wesley, Reading, Mass., 1982, pg.509.

[2.2] C. Shannon, 'A Mathematical Theory of Communication', Bell. Sys. Tech. J., vol.27, pp. 623-656, Oct. 1948.

[2.3] H. Nyquist, 'Certain Topics on Telegraph Transmission Theory', Trans. AIEE, vol.47, pp.617-644, April 1928.

[2.4] K. Feher et al, *Telecommunications Measurements, Analysis, and Instrumentation*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1987, pg.11.

[2.5] F. Stremler, *Introduction to Communication Systems, 2nd Edition*, Addison-Wesley, Reading, Mass., 1982, pg.368.

CHAPTER 3. PARTIAL RESPONSE SIGNALLING (PRS)

3.1 INTRODUCTION

Partial response signalling or correlative coding, as it is sometimes called, was invented by Lender in the 1960's [3.1]. It is a data coding technique that uses controlled amounts of ISI to achieve (amongst other things) very convenient signal spectral properties. These spectral properties simplify filter design at the transmitter and receiver of a communication system. A secondary result of this technique is a reduced sensitivity to fluctuations in the transmission rate. Due to the correlation between successive bits or symbols, this scheme also has inherent redundancy which can be used for error detection and correction. In certain applications PRS can achieve higher data rates in a given bandwidth than conventional signalling schemes, sometimes with fewer levels and therefore with better error performance.

As described in Section 2.3.1, Nyquist showed that the theoretical maximum symbol rate in a bandwidth of f_s Hz is $2f_s$ symbols/sec, or equivalently that the maximum possible bandwidth efficiency is 2 symbols/sec/Hz. This is the maximum possible rate that maintains zero ISI in the transmitted signal. Practical limitations on realizable filters often bring the bandwidth efficiency down to 1 symbol/sec/Hz. Lender showed that by introducing correlation between successive symbols, a bandwidth efficiency of 2 symbols/sec/Hz, and sometimes higher [3.2], is possible.

3.2 BASEBAND CLASS 1 PARTIAL RESPONSE SIGNALLING

The best explanation of the partial response signalling technique is by way of example. The specific case of duobinary correlative coding will be discussed. The extension to other correlative schemes follows easily enough.

Duobinary (or 'class 1' partial response) is a binary data transmission scheme. This coding procedure adds a binary data stream to a 1-bit delayed version of itself, so that for an input data stream x_k, with impulse spacing T_s, we will get an output symbol stream y_k where

$$y_k = x_k + x_{k-1}.$$

This is implemented using a transversal delay filter. Note that the symbol output of the delay filter is now a 3-level signal. For $x_k \in \{0,1\}$ we get $y_k \in \{0,1,2\}$. The probabilities of the signal levels are easily deduced : level 0 can only result from the input combination 00, level 1 can result from the combination 01 or 10, and level 2 can only result from the combination 11. The two outer levels therefore have probability of occurrence $P_{0,2} = 1/4$, and the middle level has probability $P_1 = 1/2$.

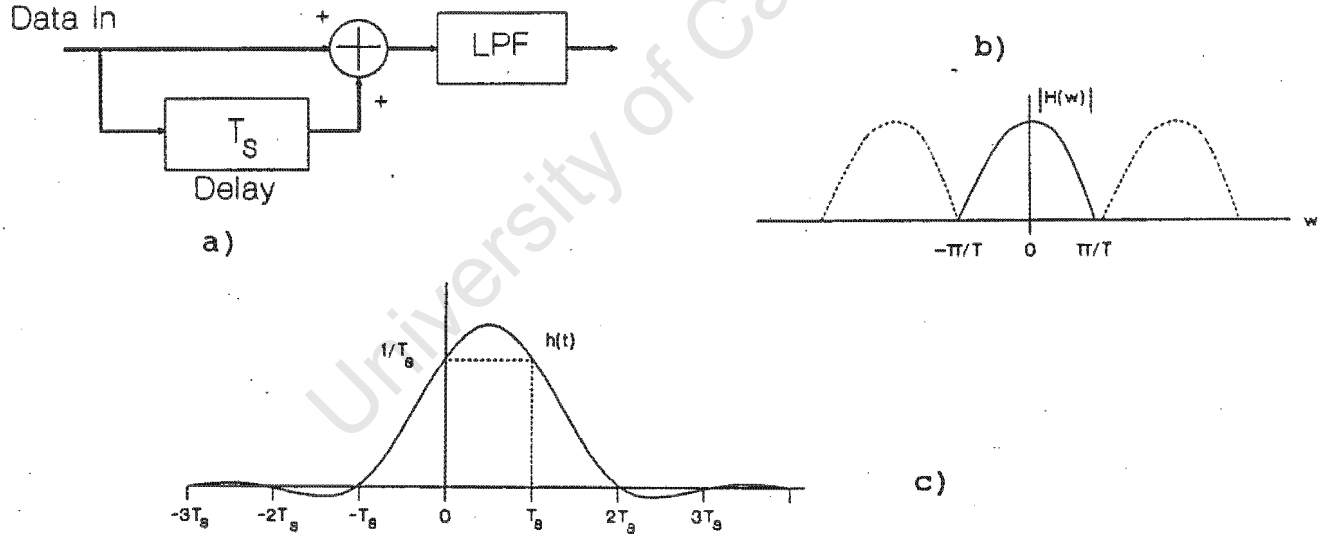


Figure 3.1 - The duobinary process.
a) - Duobinary transversal filter
b) - Filter transfer function.
c) - Baseband duobinary impulse response.

The impulse response of the delay filter is

$$f'(t) = \delta(t) + \delta(t - T_s)$$

This corresponds to the frequency transfer function

$$F'(w) = (1 + e^{-jwt}).$$

This makes sense intuitively as a delay in the time domain results in a linearly increasing phase shift with frequency in the frequency domain. The first null will therefore occur at $1/2T_S$.

Lowpass filtering at the first null (the Nyquist frequency) as shown in Figure 3.1 b) gives the transfer function

$$F(w) = \begin{cases} 2\ e^{-jwTs/2} \cos wT_S/2 & |w| \leq \pi/T_S \\ 0 & \text{elsewhere} \end{cases}$$

The associated impulse response is

$$f(t) = \text{sinc}(t/T_S) + \text{sinc}([t - T_S]/T_S)$$
$$\text{where } \text{sinc}(x) = (\sin x)/x$$

This impulse response is shown in Figure 3.1 c). There are now two non-zero signal values at the sampling instants, as opposed to one in the case of the impulse response of a 'brick-wall' filter. It is therefore seen that the duobinary process adds ISI to a symbol in such a way that it only arises from the preceding symbol. The inherent spectral shaping allows filtering at the Nyquist frequency. This process makes it possible to reach the theoretical bandwidth efficiency limit of 2 symbols/sec/Hz.

A sequence of binary data passing through the duobinary transmission filter is therefore encoded to give convenient spectral shaping. It is then lowpass filtered to isolate the baseband spectrum. The resulting baseband signal has regularly-spaced timing instants where the signal consists only of contributions from two adjacent bits, and takes on

only one of three levels. These instants occur at the Nyquist rate.

The receiver must sample the signal at the correct instants and decode the 3-level symbol correctly, according to the rule

$$x'_k = y_k - x'_{k-1}$$

It is obvious that it is necessary to know the correct value of the preceding bit x'_{k-1} to correctly decode x'_k, or else the propagation of errors would occur. Lender developed a precoding rule that eliminates error propagation and allows bit-by-bit detection at the receiver in PRS systems. For a delay filter function $x(D)$ where D is the delay operator, precoding can be achieved by using the function $[1/x(D)]_{modm}$ where m is the number of levels in the sequence x_k, and modm indicates taking the value modulo-m.

The precoding for duobinary is simple. For a binary input stream x_k the precoded sequence b_k is formed as follows:

$$b_k = (x_k - b_{k-1})_{mod2}$$

The transmitted sequence is then

$$y_k = b_k + b_{k-1} = (x_k - b_{k-1})_{mod2} + b_{k-1}$$

Consider a duobinary transmission scheme. If x_k is 1, y_k will also be 1 irrespective of the value of b_{k-1}. If x_k is 0 then y_k will be either 0 or 2, depending on the value of b_{k-1}. Thus x_k can be decoded on a bit-by-bit basis using the following rule

$$x'_k = y_k \bmod 2$$

So to summarize, for an M-level data source, the following rules apply for duobinary-type partial response signalling:

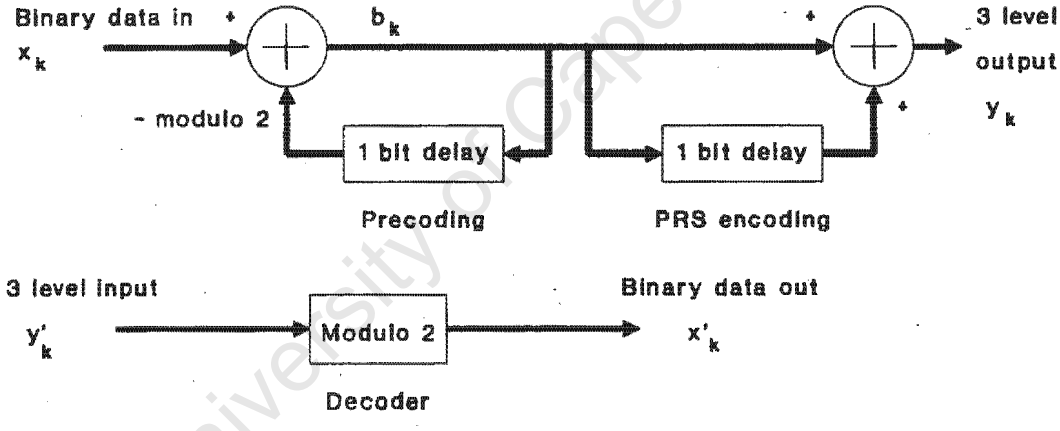
$$\text{precoding} : b_k = (x_k - b_{k-1})_{\text{mod}m}$$

$$\text{coding} : y_k = b_k + b_{k-1}$$

$$\text{decoding} : x'_k = y_k \bmod M$$

Figure 3.2 a) shows the complete duobinary system block diagram, and Figure 3.2 b) shows a sample sequence of xk, bk, yk and y'k mod2 for the duobinary case. The sequence y'k is obtained by sampling the bandlimited signal at the correct sample instants.

a)



b)

x_k		0	1	1	0	1	0	0	0	1	1	0	1	0	1	1	1
b_k	0	0	1	0	0	1	1	1	1	0	1	1	0	0	1	0	1
y_k		0	1	1	0	1	2	2	2	1	1	2	1	0	1	1	1
x'_k		0	1	1	0	1	0	0	0	1	1	0	1	0	1	1	1

Figure 3.2 a) - Duobinary block diagram
b) - example data sequences in duobinary system.

As is shown later, this project employs both 3-PRS, or normal duobinary, and 7-PRS, also called 7-level duobinary. For completeness the coding steps involved in 7-level class 1 partial response signalling are now described. Consider a binary input data stream x_k. The binary bits are grouped into pairs to give the symbol stream a_k. This sequence is thus a 4-level sequence (possible values 0, 1,2,3) and thus **M** in this case is **4**. In the 7-level case, the a_k are precoded as before to give the sequence b_k.

$$b_k = (a_k - b_{k-1}) \bmod 4 \qquad 0 \le bk \le 3$$

The b_k are then passed through the transversal delay filter to give the sequence y_k.

$$y_k = b_k + b_{k-1} \qquad 0 \le yk \le 6$$

Note that yk is a 7-level sequence. The receiver samples values y'k from the received signal, and the original symbols are decoded.

$$a'_k = y'_k \bmod 4$$

Figure 3.3 shows an example sequence of bits passing through a 7-PRS system.

x_k		01	10	10	00	11	10	01	01	11	00	00	10	11	10	01	11	10
a_k		1	2	2	0	3	2	1	1	3	0	0	2	3	2	1	3	2
b_k	0	1	1	1	3	0	2	3	2	1	3	1	1	2	0	1	2	0
y_k		1	2	2	4	3	2	5	5	3	4	4	2	3	2	1	3	2
a'_k		1	2	2	0	3	2	1	1	3	0	0	2	3	2	1	3	2

Figure 3.3 - An example of a bit sequence being encoded and decoded in 7-PRS.

3.2.1 OTHER PARTIAL RESPONSE SYSTEMS

Although this project uses the format of class 1 partial response signalling, there are other PRS schemes, each giving different spectral or pulse shape properties. The general formula for the delay filter in a correlative system (excluding the precoding) is given by the summation

$$F(D) = \sum_{n=0}^{N-1} f_n D_n$$

where N-1 is the no. of delay taps,
D is the delay operator.

Table 3.1, taken from [3.3], shows the baseband spectrums and impulse responses of some partial response systems formed from the 1-D and 1+D operators.

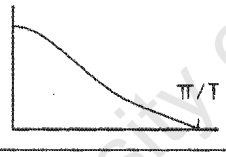
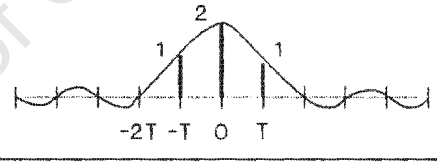
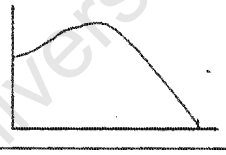
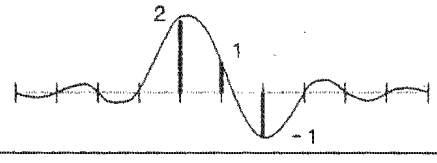
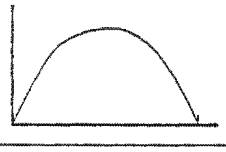
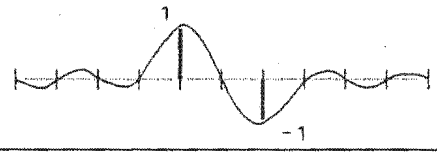
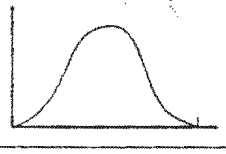
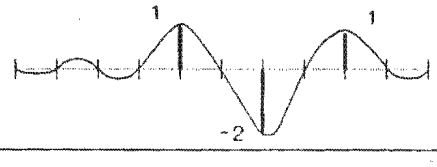
F(D)	| H(w) |	h(t)
(1+D) = 1 + 2D + D^2 (class 2)		
(1+D)(2-D) = 2 + D - D^2 (class 3)		
(1+D)(1-D) = 1 - D^2 (class 4)		
$(1+D)^2(1-D)^2$ = 1 - $2D^2$ - D^4 (class 5)		

Table 3.1 - Partial response systems.

3.3 MODULATING IN QUADRATURE TO GIVE QPRS

The baseband signals described above can of course be modulated onto high frequency carriers. The modulation can take on the form of amplitude, frequency or phase modulation. This project is concerned only with amplitude modulation. Modulating in quadrature allows maximizing the bandwidth efficiency of a digital modulation system. A class 1 7-PRS baseband system has a maximum bandwidth efficiency of 4 bits/sec/Hz. Modulating this signal onto a carrier doubles the signal spectral width, and therefore halves the bandwidth efficiency to 2 bits/sec/Hz. Adding two such signals in quadrature restores the bandwidth efficiency to 4 bits/sec/Hz.

If two 7-level baseband partial response signals are modulated in quadrature, the result is a signal with 49 possible signal states. This is called 49-quadrature PRS, or 49-QPRS. The signal constellation diagram for 49-QPRS is shown in Figure 3.4. The points in the I-Q plane represent the possible signal states at the symbol sampling instants. Two 3-PRS signals can be similarly modulated in quadrature, resulting in a 9-PRS signal.

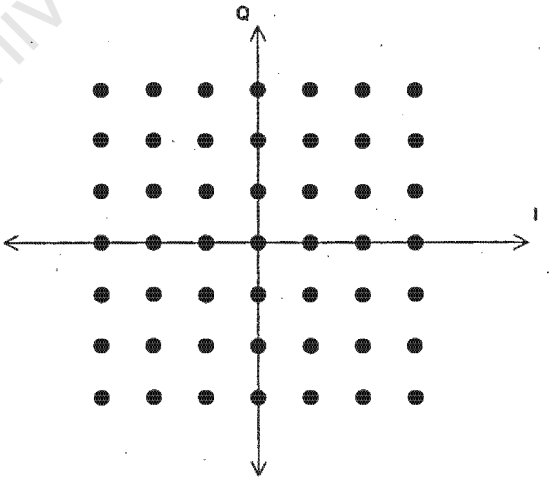


Figure 3.4 - Constellation diagram for 49-QPRS.

3.4 PERFORMANCE CHARACTERISTICS OF PRS

The performance of a baseband digital communication system can be evaluated in a number of ways. Two popular characteristics used in partial response signalling are the bit-error rate versus signal-to-noise ratio measurement, and the speed tolerance (sensitivity of a system to fluctuations in the data transmission rate). These two measurements are described in the following sections, but before discussing them the concept of an eye pattern must be described.

An eye pattern results from truncating the viewed time-base of a signal to one symbol width, and then continually superimposing the signal over this one symbol width. Figure 3.5 shows the eye pattern for binary class 1 partial response signalling.

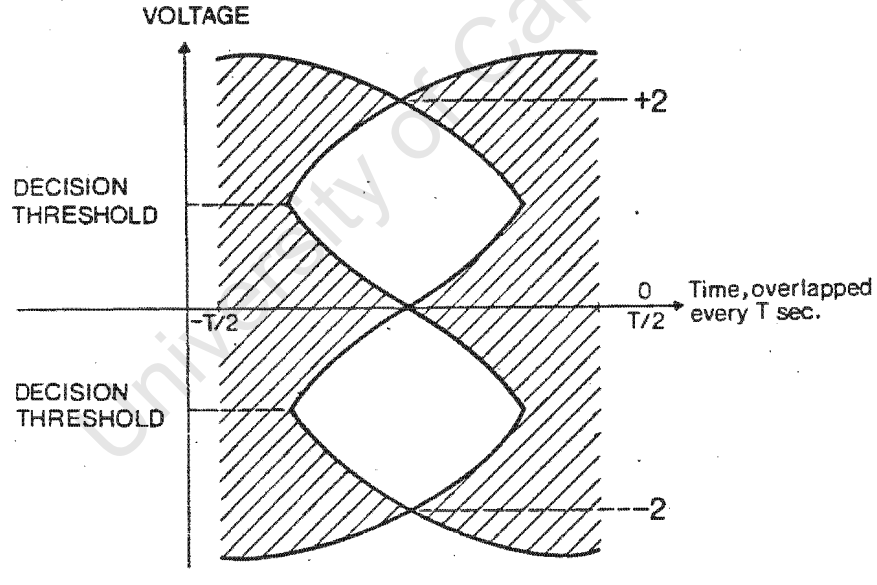


Figure 3.5 - Class 1 partial response eye pattern.

Note that the resultant signal has 3 possible levels at the sampling instants. The eye pattern is easily viewed on an oscilloscope, and the quality of the eye is then a visible indicator of the amount of noise or distortion in the signal. As the quality is degraded, the eye opening will close, and transmission errors may result. Viewing the eye

pattern gives a good qualitative measure of the time domain performance of a system.

Section 3.4.1 compares the theoretical error performance of class 1 PRS to that of other modulation schemes. Section 5.4.2 discusses the speed tolerance of partial response signalling.

3.4.1 ERROR PERFORMANCE OF PRS

The probability of error versus signal-to-noise ratio measurement is used to characterize how well a system performs in the presence of noise. It is perhaps the most frequently used method of comparing modulation schemes against each other.

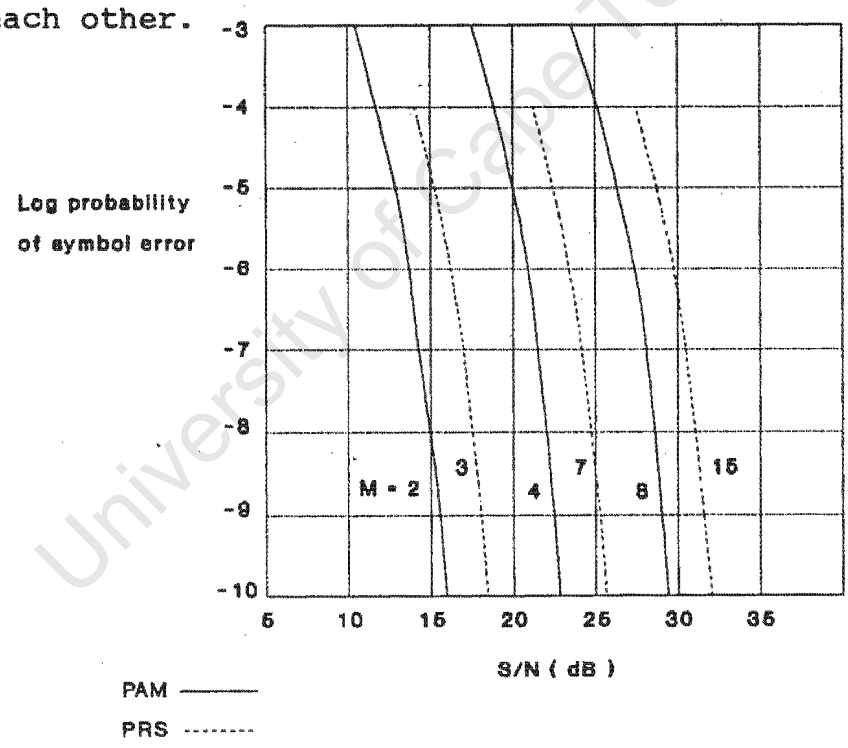


Figure 3.6 - Error performance of baseband PAM and duobinary systems.

Figure 3.6 shows the probability of symbol error versus signal-to-noise ratio for baseband 2, 4, 8 and 16-level PAM and 3, 7 and 15-level duobinary. The RMS signal-to-noise ratio is measured in the Nyquist bandwidth after the receiver lowpass filter [3.4]. The theoretical bandwidth

efficiency of 2-PAM and 3-PRS is 2 bits/sec/Hz. It is seen from Figure 3.6 that for the same probability of error, 3-PRS requires approximately 3dB more signal power. This is misleading, since PAM systems very often use $\alpha = 1$ raised-cosine filters before transmission, which reduces the bandwidth efficiency to close to 1 bit/sec/Hz. 3-PRS, however, can be used with filtering at the Nyquist frequency due to its inherent spectral shaping, and therefore maintains a bandwidth efficiency of close to 2 bits/sec/Hz. The penalty for this is the requirement for a slight increase in transmitted power.

Figure 3.7 shows the symbol probability of error versus carrier-to-noise ratio for various QPRS, QAM and QPSK modulation schemes [3.5]. The QAM and QPRS plots are an extension of those given in Figure 3.6. Again it is seen that there is a slight signal-to-noise advantage of QAM systems over QPRS systems, although QPRS has superior bandwidth efficiency.

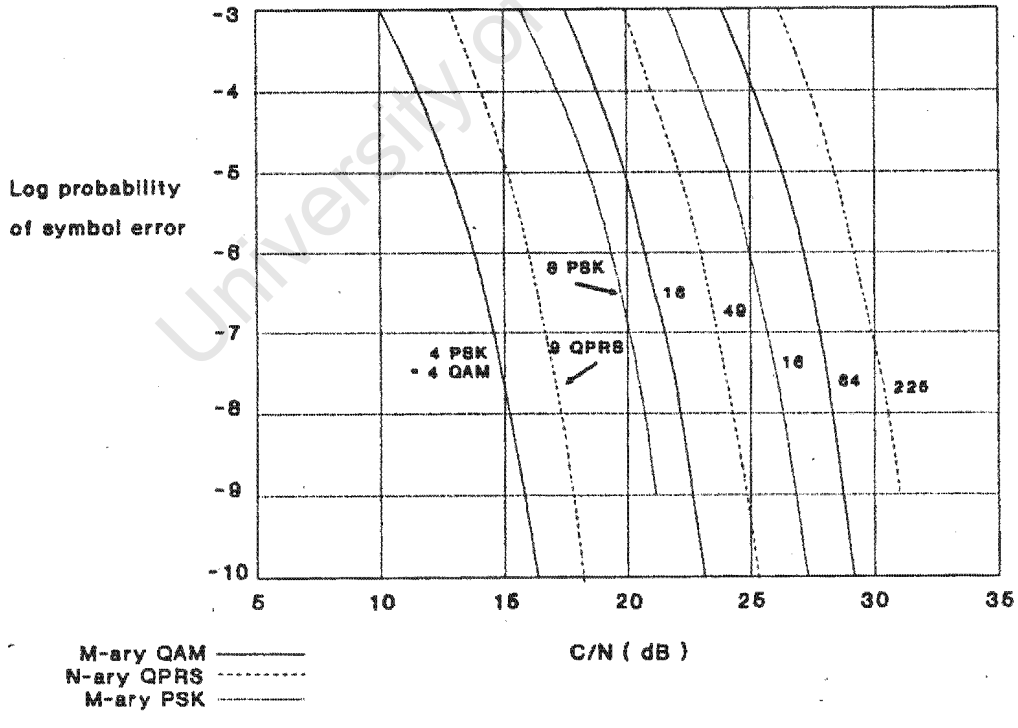


Figure 3.7 - Error performance of QPRS, QAM and QPSK.

The practical bandwidth efficiency of PRS versus memoryless systems is given in Table 3.2 [3.6]. Here it is assumed that the memoryless systems employ $\alpha = 1$ (100% excess bandwidth) filtering.

Bandwidth Efficiency (bits/sec/Hz)	Number of levels in memoryless system with alpha = 1	Number of levels in correlative systems
1	2	
2	4	3
3	8	
4	16	7
5	32	
6	64	15
7	128	
8	256	31

Table 3.2 - Bandwidth efficiency of PRS and memoryless systems.

3.4.2 SPEED TOLERANCE

The speed tolerance of a bandlimited data communication system can be defined as the increase in the data transmission rate at which the smallest eye pattern opening becomes zero. It is thus a measure of the sensitivity of a system to timing variations.

The theoretical rectangular filter ($\alpha = 0$) memoryless system will not tolerate any increase in the data transmission rate, due to the large spectral content of the signal near the cutoff frequency. Partial response schemes have an inherent null at the cutoff frequency, and are therefore more tolerant of increases in the data transmission rate. It has been shown that the $1 + D$ and $1 - D^2$ PRS systems have speed tolerances of 42.5% and 15.5% respectively [3.7]. Class 1 PRS ($1 + D$) is the most robust of all PRS systems in terms of speed tolerance. An interesting plot of speed tolerance versus the roll-off parameter α for binary, $1 + D$, and $1 - D$ systems is shown in Figure 3.8. Here it is seen

that at low values of α the 1 + D system is vastly more speed-tolerant than the binary system.

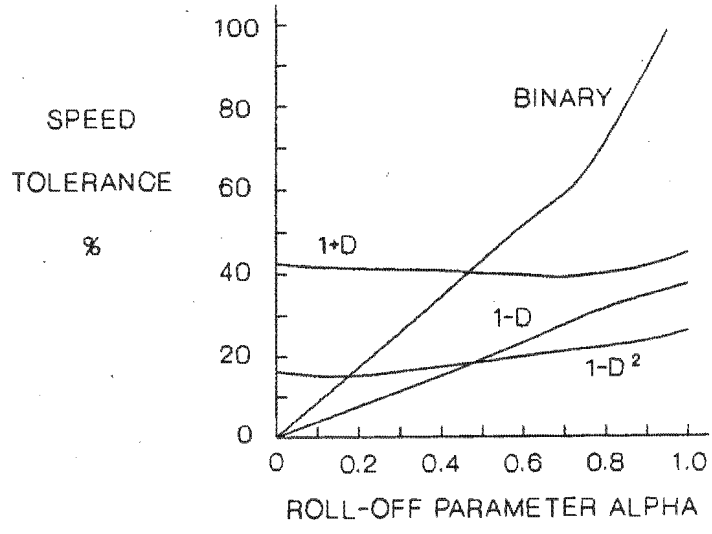


Figure 3.8 - Speed tolerance as a function of the roll-off parameter α.

3.5 SUMMARY

Partial response signalling is a coding technique used in digital communication systems. A source symbol sequence passing through a PRS system is operated on as follows:

1) At the transmitter the symbols are precoded to eliminate error propagation that can occur in finite-memory systems.

2) The precoded symbols are encoded according to PRS rules. In the case of class 1 PRS this involves adding a delayed version of the symbol sequence to itself. This causes convenient spectral shaping with a null in the spectrum at the Nyquist frequency. This is a fundamental benefit of PRS.

3) Lowpass filtering at the Nyquist frequency is performed, giving transmission technique with superior spectral efficiency to memoryless systems. This is the crux of PRS. The baseband signal can be modulated onto a carrier prior to transmission.

4) At the receiver (after demodulation), symbol-by-symbol detection is possible.

3.6 REFERENCES

[3.1] A. Lender, 'The Duobinary Technique for High Speed Data Transmission', IEEE Trans. Commun. Electron., vol.82, pp. 214-218, May 1963.

[3.2] K. Wu & K.Feher, 'Multilevel PRS/QPRS Above the Nyquist Rate', IEEE Trans. Commun., vol.33, no.7, pp. 735-739, July 1985.

[3.3] P. Kabal & S. Pasupathy, 'Partial Response Signalling', IEEE Trans. Commun., vol.23, no.9, pp.921-934, September 1975.

[3.4] P. Peebles, *Digital Communication Systems*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1982. pg.188.

[3.5] K. Feher & Engineers of Hewlett-Packard, *Telecommunications Measurements, Analysis, and Instrumentation*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1987, pg. 30.

[3.6] K. Feher, *Advanced Digital Communications Systems and Signal Processing Techniques*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1987, pg. 328.

[3.7] [6.3] above, pg. 930..

CHAPTER 4. THE MODEM STRUCTURE

4.1 INTRODUCTION

As stated in the introductory chapter, the specifications of the modem are:-

1. Operation over M.1020 conditioned telephone-type 4-wire line.

2. Half-duplex operation at 2400, 4800 and 9600 bps.

3. Use of partial response signalling as a modulation technique.

The primary aim of this project is to investigate the use of partial response signalling (PRS) as the modulation technique in a modem application. As described previously, the choice of PRS was made due to its performance characteristics when operated near the Nyquist limit of data transfer. Reference to the project flowchart in the introductory Chapter 1 shows the research outline.

The proposed modulation and demodulation process can be broken down into several functional steps, as shown in Figure 4.1. The modulation process corresponds to the transmitter and the demodulation corresponds to the receiver. The sequence of events is described as follows:-

1. The binary data from a computer or similar source is split into two channels. These two channels correspond to the in-phase (I) and quadrature (Q) signal components.

2. The binary data in each channel is either converted into 4-level data for 49-QPRS or left in binary format for 3-PRS and 9-QPRS.

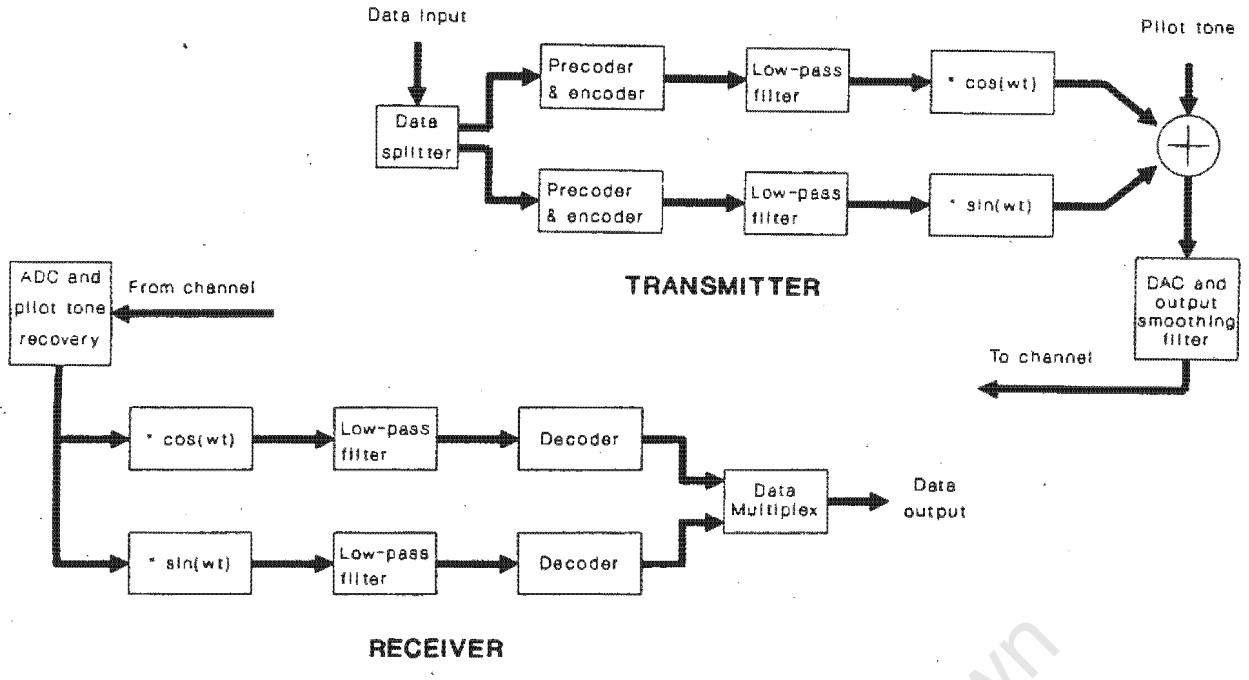


Figure 4.1 - Functional block diagram of modulation and demodulation in the modem.

3. This data is then precoded and encoded according to class 1 partial response coding rules.

4. The resulting symbol sequence is lowpass filtered according to Nyquist's criteria for elimination of ISI. The transmit and receive filters are chosen so that when cascaded they form the desired overall transfer function.

5. The lowpass signal in each channel is modulated onto I and Q carriers for transmission.

6. Finally a pilot tone is added to the composite I and Q signal before transmission. This provides the synchronization information at the receiver. A description of this is given in Chapter 5.

7. The signal passes through a digital-to-analogue converter.

8. At the receiver, the pilot tone is used to recover the carrier synchronization, sampling timing, and symbol sampling synchronization. The signal passes through an analogue-to-digital converter.

9. The signal is demodulated into I and Q lowpass signal paths.

10. The demodulated signals for the I and the Q channels are lowpass filtered to extract the baseband signals.

11. The baseband signals are sampled at the symbol rate to recover the symbol values.

12. The symbols are decoded according to PRS rules.

13. If 7-PRS is being used, the decoded symbols are converted into binary data.

14. The data is multiplexed to form the recovered bit stream.

More detailed descriptions of these steps are given in Chapters 5 and 6, which outline the design of the modem transmitter and receiver. The design is predominantly digital and based on the use of a sophisticated microprocessor chip. Due to the digital architecture, the project was both software and hardware-oriented.

4.2 MOTIVATION FOR USE OF DIGITAL ARCHITECTURE

The term 'digital architecture' implies the use of digitally-based hardware to implement a circuit function. This project involves both digital communications (in the transfer of data from a transmitter to a receiver via a modem) and a digital architecture (in the form of a digitally-based modem design).

Digital design approaches to low-frequency communication systems are gaining popularity over analogue techniques for a number of reasons. The main advantages of using a digital rather than analogue technology are:

1. Digital hardware is very reliable and predictable. Signals can be precisely generated. High noise immunity is achieved.

2. Analogue filters, often of the lumped-element type, require complex design methods, time-consuming development, and usually need to be factory tuned [4.1]. Digital filters are easily implemented and there is no need for factory tuning.

3. Linear phase characteristics result in finite impulse response (FIR) filters [4.2].

4. In the development stage of digital filter design, it is easy to re-program the same hardware for various filter responses.

5. Digital techniques allow many powerful signal processing operations to be performed, e.g. coding, data storage, non-causal filtering etc.

These advantages, and the current trend towards digital circuit design, led to the choice of a digital rather than an analogue circuit design.

4.3 IMPLEMENTATION USING THE MOTOROLA DSP56001 SIGNAL PROCESSOR

4.3.1 CHOOSING THE DSP56001 MICROPROCESSOR

It was decided to use a microprocessor as the basis for the circuit design. The Motorola DSP56001 signal processor was chosen to implement all encoding and decoding, lowpass filtering, modulation and demodulation, and pilot tone addition in the modem design. The processor thus forms the heart of the digital architecture. The important advantages of this microprocessor are [4.3]:-

1. A parallel architecture - the data arithmetic logic unit, address generation unit and program controller operate in parallel. This means that a 24*24 bit multiplication, two address pointer updates, two data moves and an instruction prefetch can be executed simultaneously.

2. A no-overhead hardware DO loop and the parallel architecture allow convolution loops to be executed in the theoretical minimum number of calculations.

3. Modulo address registers allow efficient implementation of circular buffers.

4. The processor operates at high speed (10 mips).

5. A built-in sine wave lookup table is useful for implementing carrier modulation.

Two DSP56001 prototype development boards, one for the transmitter hardware and one for the receiver hardware, along with assembler and downloading software, were purchased from Peralex Electronic Development C.C. in Cape Town. These boards are designed to plug into the expansion port of an IBM-compatible personal computer. All external memory, digital-to-analogue (DAC), analogue-to-digital (ADC)

circuitry, external decoding logic etc. was either wirewrapped onto the development board or built as external circuitry.

4.3.2 BRIEF DESCRIPTION OF THE DSP56001 MICROPROCESSOR

Information given in this section is a summary of that given in [4.2], and its purpose is to briefly describe the features of this microprocessor that are relevant in the modem design. Further information is given in Appendix B.

Data Buses - There are four internal data buses. These are the X and Y data buses (for data transfer to/from X and Y memory spaces), the program data bus (instruction word transfer), and the global data bus (all other transfers, including I/O movement). These are all multiplexed to the external 24-bit data bus.

Address Buses - There are three internal 16-bit address buses. The X and Y address buses reference the X and Y data memory spaces. The program address bus references the program memory space. All three address buses are multiplexed to the external address bus.

Registers - There are two data accumulator registers, A and B. They each consist of two 24-bit data registers (A0,A1,B0,B1) and an 8-bit extension register (A2,B2). There are two other groups of data registers, X and Y, each consisting of two 24-bit registers (X0,X1,Y0,Y1). There are three groups of 8 address registers, each 16 bits wide. These are the pointer registers (R0 - R7), the offset registers (N0 - N7) and the modifier registers (M0 - M7).

X Data memory - The 512-location on-chip X memory is divided into two spaces. The first 256 locations are 24-bit wide RAM, and the second 256 locations contain factory-programmed ROM with μ-law and A-law lookup tables. External X memory (up to 64K) can be added.

Y Data memory - This has the same structure as the X data memory, but the ROM space contains a full 256-point sinewave lookup table.

Program memory - The on-chip program RAM consists of 512 by 24-bit locations. Up to 64K external program memory can be added. There is also a factory-programmed Bootstrap ROM which can be used to download user programs into the program memory space.

Expansion Port - This consists of the 24-bit external data bus, the external 16-bit address bus, and the associated bus control signals. It is used to interface to external RAM, digital-to-analogue converters etc.

Host Interface - This consists of an 8-bit parallel port and a group of host control signals. It is used to interface the DSP56001 microprocessor to a host computer. This is used extensively in this project.

Port C Serial Interface - This consists of a serial communications interface (SCI) and a synchronous serial interface (SSI). These contain all the signals necessary for serial communication and serial data transfer. All these pins can also be programmed as general purpose I/O pins, and are in fact used as I/O pins in this project.

4.3.3 MODEM FUNCTIONS IMPLEMENTED BY THE DSP56001

The modem functions implemented by the DSP56001 are identical to those shown in block diagram form in Figure 4.1. These are all the essential signal processing steps in the modulation and demodulation scheme as previously described. Later chapters give a more detailed look at the different functions and the method of implementation.

4.3.4 USING THE DSP56001 AND THE PERALEX PROTOTYPE BOARD

This section describes how the prototyping card supplied by Peralex was used in the development of the project. A typical use of the development board would be as follows:-

1. Write a Motorola DSP56001 assembly language program to perform the desired process.

2. Assemble the program into bootable DSP56001 program code using the Motorola Cross-Assembler. The Assembler creates a downloadable file of code with the same prefix as the assembler file, but with a '.LOD' filename extension.

3. Write a Pascal host program to control the DSP56001, send data, lookup tables, read results etc. from the DSP56001. This is done by accessing the Host interface on the chip via the PC expansion port.

4. Download the program code to the DSP56001 using the download program written by Peralex. This places the executable code from the '*.LOD' file into the DSP's program memory space.

5. Run the Pascal program, which starts DSP program execution, and transfers data to/from the chip. Lookup tables are usually loaded first via the host interface, and then normal chip operation starts.

6. If necessary (for example in simulation runs) the Pascal program reads resulting data from the chip via the host interface.

The prototyping board is specifically designed so that the microprocessor can be used to run real-time signal processing tasks while being monitored by the Host Pascal program. This gives more control over the development of the project.

4.4 DESIGN OF THE DIGITAL LOWPASS FILTERS

The most critical function the DSP56001 has to perform is digital lowpass filtering. In the transmitter, this involves the filtering of the PRS symbol impulse sequence to produce a baseband signal. In the receiver, it involves filtering the sampled and demodulated received waveform to recover the baseband signal. It is well known that a matched filter pair provide the optimum signal-to-noise performance in a communication system. The transmit and receive lowpass filters used in the modem structure are therefore chosen to be a matched pair, and their cascaded transfer function is thus the overall system transfer function.

The overall system transfer function was chosen to have a raised-cosine response. As mentioned earlier in section 2.4, this is a very popular choice in communication systems due to its practicality in satisfying Nyquist's filter requirement for zero ISI. The problem encountered in choosing this filter response was that the transmit and receive filters each had to have a square-root raised-cosine response to give the desired overall response. The inverse Fourier transform of the root raised-cosine frequency transfer function cannot be solved explicitly, and no text could be found that gave a formula or derivation for it. The inverse Fourier transform of a function F(w) is defined as

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(w) e^{-jwt} dw$$

For a real-valued signal f(t),

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(w) \cos(wt) dw$$

The root-raised cosine transfer function is [4.5]:

$$F(w) = \begin{cases} \sqrt{T} & \text{for} \quad 0 \le |w| \le (1-\alpha)W \\ \sqrt{\left[0.5\left(1 - \sin\left[\frac{\pi}{2\alpha W} \left(|w| - W \right) \right]\right)\right]} & \\ & \text{for} \quad (1-\alpha)W \le |w| \le (1+\alpha)W \\ 0 & \text{for} \quad |w| > (1+\alpha)W \end{cases}$$

A Pascal program called 'SIMPSON.PAS' (see Appendix H) was written to evaluate f(t) for F(w) defined above, using Simpson's rule. The program generates the discrete impulse response values of f(t) for a given filter length and roll-off factor α. These values form the lowpass filter lookup table in the microprocessor.

An important point of note in digital filter design is the relationship between the sampling rate and the symbol width of the filter. Ideally, the symbol width of the filter determines the cutoff 'sharpness' and the sidelobe suppression. The sampling rate, however, determines the position of the spectral replicas in the frequency domain. If the sampling rate is low and the spectral replicas are close to each other, aliasing might occur, even if the filter is wide (in symbol width terms). The design of a digital filter must take these factors into account.

4.5 SAMPLING CONSIDERATIONS

Before continuing, it is necessary to explain the system sampling requirements. The transmitter does not sample an analogue signal as such, but instead it produces the impulse samples which will then be converted into an analogue signal for transmission. The receiver, however, does sample an analogue signal. The three main factors that affect the sampling rate are listed below.

1. Speed capabilities of the DSP56001.

The DSP56001 has an instruction cycle time of 98 nsec. This figure determines how many computations can be done per sampling interval. A high sampling rate implies that the FIR filter length is limited due to the time taken to perform a filter convolution.

2. Spectrum of the transmitted signal.

The transmit carrier frequency is 1800 Hz and the bandwidth of the baseband signal is expected to be 1200 Hz, so the upper frequency limit of the transmitted signal is expected to be 3000 Hz. The sampling frequency must therefore be at least twice as high as this (6000 Hz). However, this frequency is still too low. A non-critical analogue smoothing filter is required at the transmitter, after digital-to-analogue conversion, to eliminate the spectral replicas that arise due to sampling. If the spectral replicas are far apart, it is easy to implement the analogue filter so that its transfer function has negligible effect on the desired signal spectrum. The fundamental aim of using digital filtering is to avoid the problems associated with analogue filtering.

3. The aperture effect.

Before transmission the sampled digital signal is converted to an analogue signal by using a digital-to-analogue converter (DAC). The DAC has a zero-order hold effect on each sample. This causes a distortion of the transmitted spectrum, since the sampled spectrum is now multiplied by a sin(x)/x response due to the zero-order hold [4.5,pg.111]. This is the aperture effect. The first null in the sin(x)/x response is at the sampling frequency, so the higher the sampling frequency, the less the effect of the sin(x)/x factor at low frequencies.

4.6 PROJECT SETUP

It is informative at this stage to give an overview of the project setup, and show how the software and the circuit hardware are interfaced. This gives a clearer picture of the scope of the project and makes it easier to discuss the individual sections of the project.

The project was hardware and software-based, and the experimental equipment was operated largely under the control of a PC. The setup, shown in simple form in Figure 4.2, consisted of separate DSP56001 prototyping cards for the modem transmitter and receiver, external hardware, Pascal software for the PC host programs, and assembler software for the DSP56001 microprocessors. The assembler programs are resident in the DSP56001 processors, the Pascal programs control the DSP56001 mode of operation, and the external hardware conditions the output of the transmitter DSP56001 and the input of the receiver DSP56001.

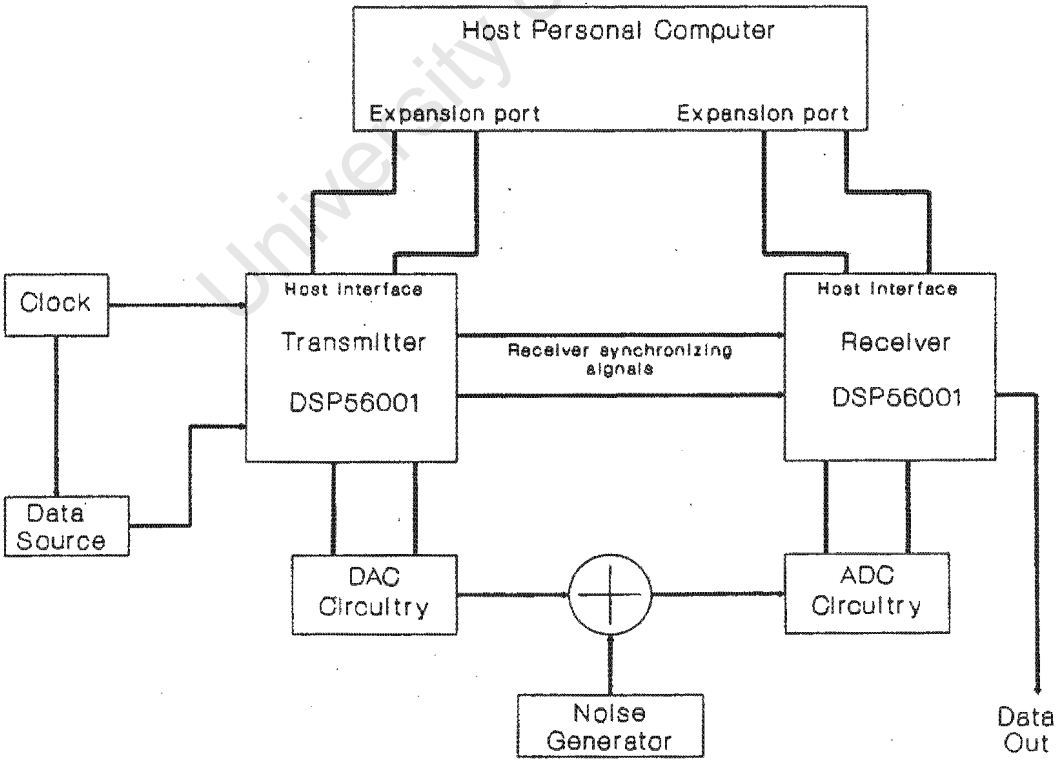


Figure 4.2 - Experimental setup.

4.7 REFERENCES

[4.1] B. Sklar, 'A Structured Overview of Digital Communications - Part 1', IEEE Communications Magazine, August 1983, pp.4-17.

[4.2] A. Oppenheim & R. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1975, pg.237.

[4.3] *Motorola DSP56001 Advance Information*, Motorola Inc., 1988.

[4.4] E Swokowski, *Calculus With Analytic Geometry*, Prindle, Weber & Schmidt, Boston, Massachusetts, 1984, pg.241.

[4.5] F. Stremler, *Introduction to Communication Systems*, Addison-Wesley, Reading, Massachusetts, 1982. pg.368.

CHAPTER 5. THE TRANSMITTER DESIGN

5.1 TRANSMITTER SIMULATION

5.1.1 INTRODUCTION

Before designing the real-time modem transmitter, computer programs were written to simulate the operation of the transmitter (this was also done for the receiver). The objectives of the simulation were:-

1. To find a program structure that would closely resemble that of the real-time transmitter.

2. To simulate the resultant transmitted waveform and signal spectrum, and to determine the effect of different length filter lookup tables and different raised-cosine filter α values.

The simulation software includes a host control Pascal program, a resident assembler program for the DSP56001 microprocessor, and two Pascal programs to view the result of the simulated transmitted waveform. The program 'Simpson.pas' is used to generates a filter response for any values of filter length, α, and points per symbol width.

The simulation software is not described in detail, since it is very similar to the real-time software, which is described in detail in later sections.

5.1.2 TRANSMITTER SAMPLING FREQUENCY

The sampling frequency for the transmitter seemed to be limited to two choices - either 19.2kHz or 38.4kHz. It was desirable to have a sampling frequency that was a 2^n multiple of the highest bit rate (and therefore the two lower bit rates) of 9600 bps. This makes system implementation and control a lot easier, since certain

operations are performed at 2^n multiples of the bit rate. This requirement suggests possible sampling frequencies of 9.6kHz , 19.2kHz, 38.4kHz and 76.8kHz. Sampling at higher rates (and in fact even at 76.8kHz) is impractical due to processor speed limitations. Sampling at 9.6kHz would cause a spectral replica extending down to 6.6kHz, placing too severe requirements on the output analogue smoothing filter. Sampling at this frequency would also cause the aperture effect to play a significant role. This left a choice of either a 19.2kHz or 38.4kHz sampling rate. Both were simulated.

5.1.3 THE HOST PASCAL PROGRAM - TXIQ.PAS

The host Pascal simulation program (called 'TXIQ.PAS') is a control program for the DSP56001 microprocessor. Its functions include downloading the lowpass filter lookup table to the DSP56001, controlling the startup timing of the microprocessor, writing binary data to the DSP56001 and then writing the sampled values of the microprocessor's transmitted 'waveform' to a file.

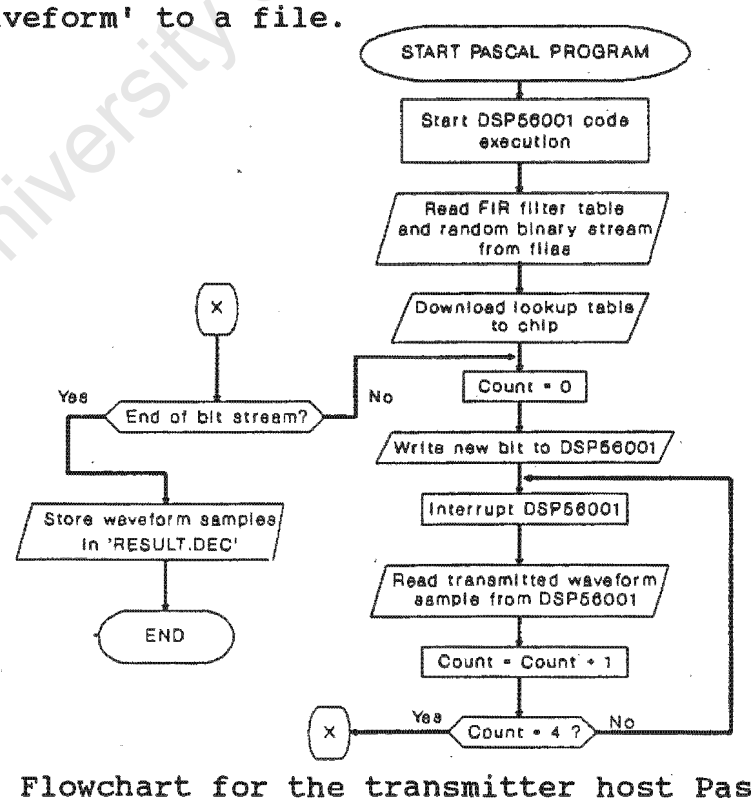


Figure 5.1 - Flowchart for the transmitter host Pascal simulation program.

Figure 7.1 shows the flowchart of the transmitter host Pascal program. The program listing of TXIQ.PAS is given in Appendix F. The program listing for TXIQ.PAS reflects a sampling rate of 38.4kHz. For this sampling rate, the following timing considerations apply:-

1. The DSP56001 is interrupt-driven at 38.4kHz. It must produce output samples at this rate.

2. The bit rate is 9600 bps, so for every bit four interrupts occur. This condition is checked in the flowchart above using the variable 'count'.

3. For a sampling rate of 38.4kHz and a symbol rate of 2400 baud in each channel, the FIR filter must therefore have 38.4/2.4 = 16 samples per symbol interval.

These conditions are seen to correspond to those in the program listings in Appendix F. For 19.2kHz the only differences are that two interrupts will occur for every input bit, and the FIR filter will have 8 samples per symbol interval.

5.1.4 THE DSP56001 ASSEMBLER PROGRAM - 'TXIQ.ASM'

The assembler program contains the program code for the DSP56001 microprocessor. The listing for this program is given in Appendix F. The program was written to closely simulate the operation of the final real-time program. As stated before, the microprocessor has to read in the binary data and split it into I and Q channels, precode and encode the data, lowpass filter the resulting symbol sequence, and modulate the baseband I and Q signals onto quadrature carriers. The program shown in Appendix F only generates a 49-QPRS signal (data rate of 9600 bps), since the generation of the signals for 4800 and 2400 bps are simpler cases of the same program. The program flowchart is shown in Figure 5.2.

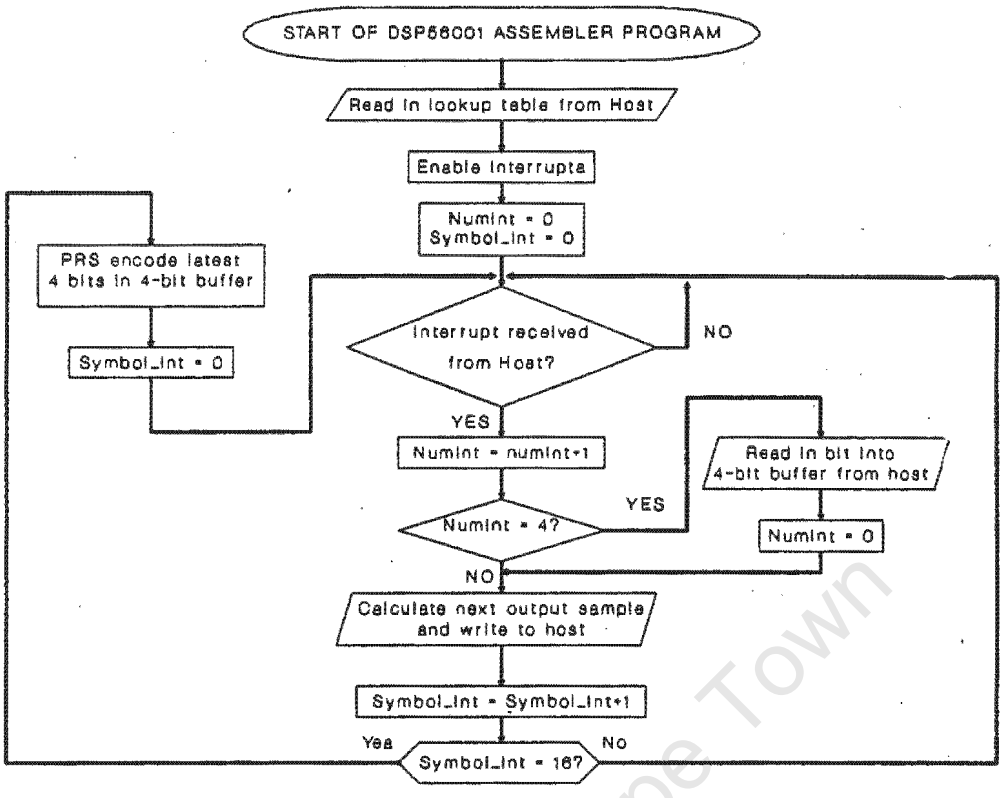


Figure 5.2 - Flowchart for transmitter DSP56001 assembler simulation program.

The DSP56001 reads in the lookup table from the host computer. The DSP56001 is interrupt driven - for a sampling rate of 38.4kHz and a data rate of 9600 bps, the DSP56001 must calculate an output sample for every interrupt and read in a bit once every 38.4/9.6 = 4 interrupts. The symbol rate here is 2400 baud, so a new symbol must be calculated once every 38.4/2.4 = 16 interrupts.

In summary, the simulated transmitter implements 7-level baseband PRS in two channels, and combines them for transmission by modulating these two channels onto orthogonal 1800Hz carriers. This produces a 49-QPRS signal.

5.1.5 TRANSMITTER SIMULATION RESULTS

The transmitter simulations were performed at the software equivalent of 19.2kHz and 38.4kHz sampling rates. A square-root raised-cosine filter (RRCF) response was used as the

transmitter lowpass filter in the simulation. The filter length and the excess bandwidth parameter α were varied to observe different RRCF responses.

The output values from the DSP56001 represent the transmitted signal samples. This transmitted signal is of the 49-QPRS modulation format. These values are stored in a file, and the data in this file is displayed on the computer as an eye pattern. The signal samples are also transformed into the frequency domain to observe the spectral properties of the signal.

The simulation results are given for the two sampling rates, followed by a brief discussion of the results. The discussion includes a motivation for the choice of certain signal parameters for the real-time transmitter.

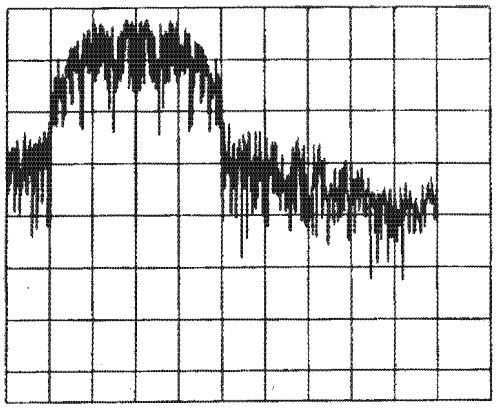
SIMULATION AT 19.2 KHZ SAMPLING RATE

It was shown in section 5.1.2 that for a sampling rate of 19.2kHz, the filter lookup table must have 8 points per symbol width. This parameter is therefore constant for all the filter responses. The following root raised-cosine filters were used in the simulation:-

1. 256 points - 32 symbols wide, $\alpha = 0$
2. 256 points - 32 symbols wide, $\alpha = 0.2$
3. 320 points - 40 symbols wide, $\alpha = 0$
4. 320 points - 40 symbols wide, $\alpha = 0.2$
5. 512 points - 64 symbols wide, $\alpha = 0$
6. 512 points - 64 symbols wide, $\alpha = 0.2$

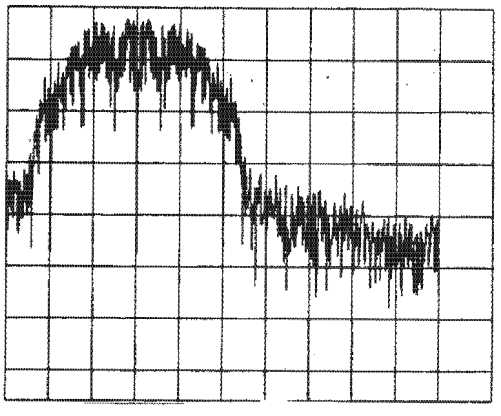
The transmitted power spectra for these filters are shown in Figure 5.3 a) - f). The eye patterns for cases 4 and 6 are shown in Figure 5.3 g) and h).

600Hz / div.



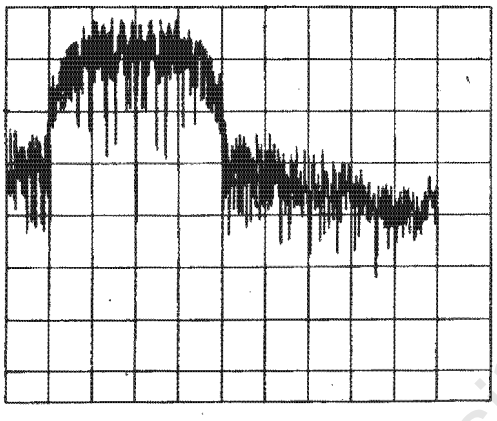
10dB / div.

600Hz / div.

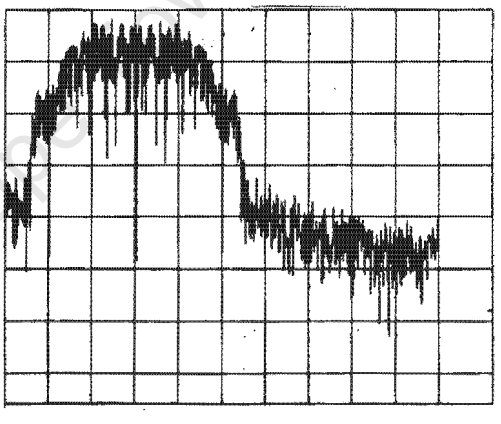


a) - 256 points, α=0.

b) - 256 points, α=0.2.

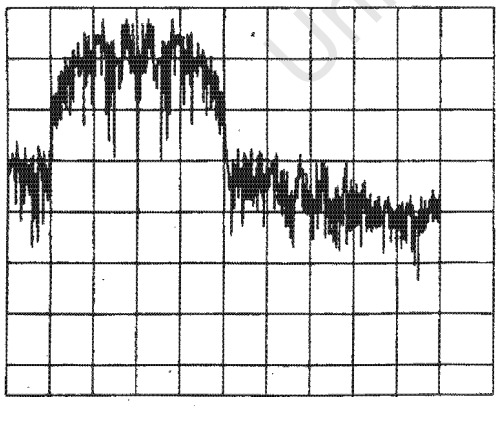


10dB / div.

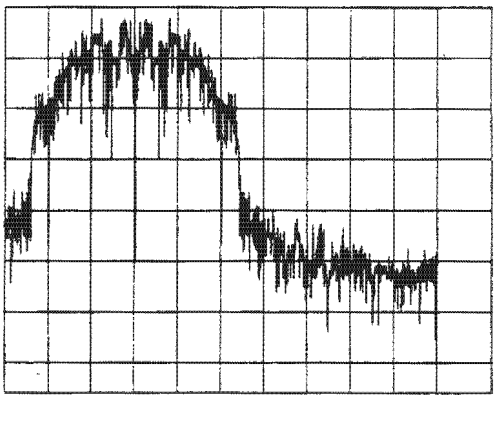


c) - 320 points, α=0.

d) - 320 points, α=0.2.

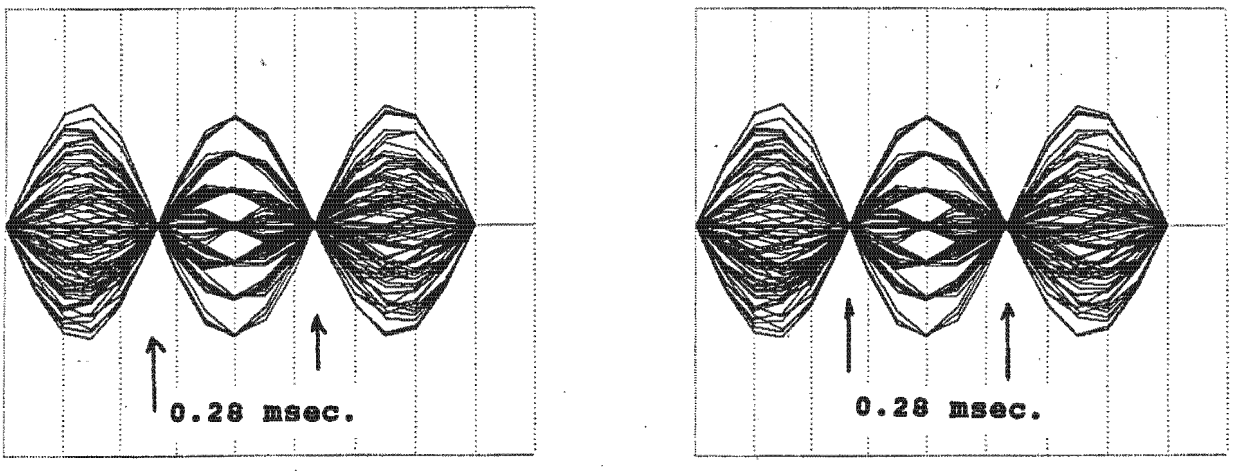


10dB / div.



e) - 512 points, α=0.

f) - 512 points, α=0.2.



g) - 320 points, α=0.2. h) - 512 points, α=0.2.

Figure 5.3 - Transmitted power spectrum and eye pattern simulation results for 19.2kHz sampling.

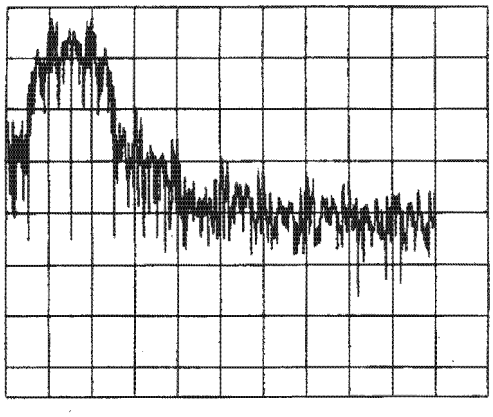
SIMULATION AT 38.4KHZ

Here the filter lookup table must have 16 points per symbol. The following root raised-cosine filter responses were used in the simulation:-

1. 256 points - 16 symbols wide, α = 0
2. 256 points - 16 symbols wide, α = 0.2
3. 320 points - 20 symbols wide, α = 0
4. 320 points - 20 symbols wide, α = 0.2
5. 512 points - 32 symbols wide, α = 0
6. 512 points - 32 symbols wide, α = 0.2

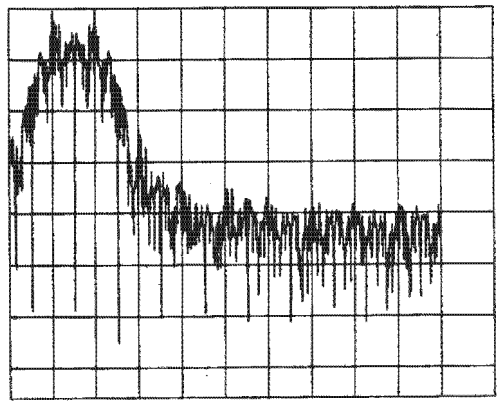
The transmitted power spectra are shown in Figure 5.4 a) to f). The eye diagram for case 4 is shown in Figure 5.4 g). There is no noticeable difference in the eye diagrams corresponding to cases 1 to 6 above, therefore only case 4 is shown.

1200Hz / div.



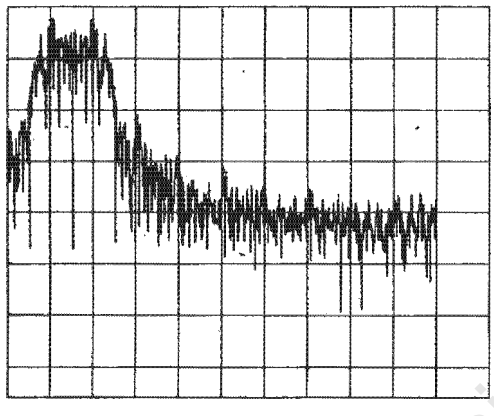
10dB / div.

1200Hz / div.

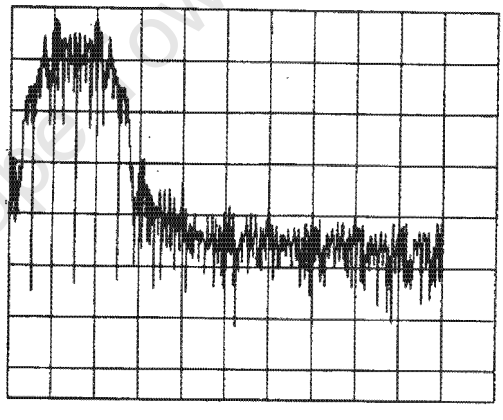


a) - 256 points, α=0.

b) - 256 points, α=0.2.

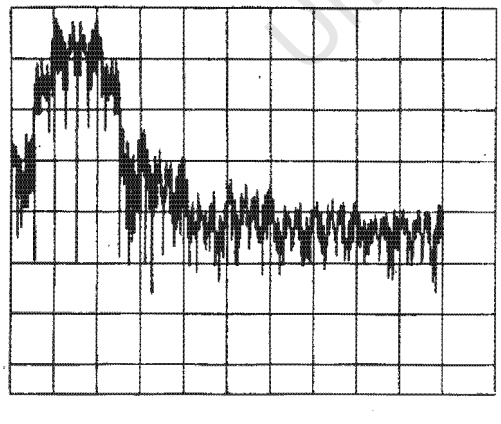


10dB / div.

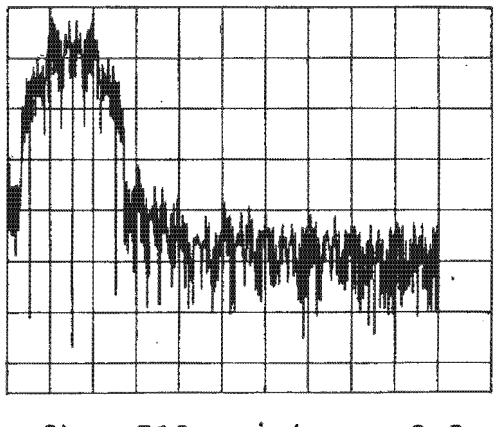


c) - 320 points, α=0.

d) - 320 points, α=0.2.

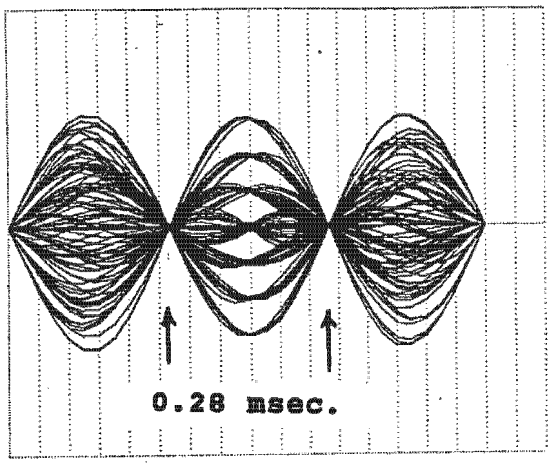


10dB / div.



e) - 512 points, α=0.

f) - 512 points, α=0.2.



g) - 320 points, α=0.2.

Figure 5.4 - Transmitted power spectrum and eye pattern simulation results for 38.4kHz sampling rate.

BRIEF DISCUSSION OF SIMULATION RESULTS

There are two basic trends that can be observed from the spectral diagrams given above. Firstly, for a given filter length, an increase in the excess bandwidth coefficient α leads to a slightly wider main lobe, but higher suppression of the sidelobes. Compare for example the frequency plots given in Figure 5.4 a) and b). Secondly, if α is kept constant and the filter length is increased, the mainlobe width is held constant but the sidelobe suppression is improved. This is shown when comparing Figures 5.4 b) and 5.4 f). These trends are as expected. It is expected that the use of an FFT here to obtain the signal spectra shows a worse filter response than would be obtained in real time, due to the truncated nature of the FFT.

In a real-time application the length of the filter table is limited by the time available for the DSP56001 to perform one convolution pass to produce a baseband signal. Since the parameter α is independent of the filter length, it is desirable to have as long a filter as possible, and then set α accordingly. At this point it was decided to use a transmitter sampling rate of 38.4kHz, assuming that a root raised-cosine filter could be implemented to give sufficient sidelobe suppression, and not excessive mainlobe widening.

5.2 TRANSMITTER HARDWARE

5.2.1 INTRODUCTION

The transmitter hardware circuitry is interfaced to the transmitter microprocessor and performs clock generation, pseudorandom data generation and digital-to-analogue conversion. The different sections of the transmitter are discussed in the following text. The transmitter circuit diagrams are given in Appendix A. In several instances in sections 5.2.1 to 5.2.5, it is assumed that various signals are provided by the transmitter DSP56001 microprocessor for the external hardware. The generation of these signals are explained in Section 5.3, which deals with the transmitter software.

5.2.2 EXTERNAL MEMORY FOR THE DSP56001

The transmitter DSP56001 required external program and data RAM to be connected to the microprocessor P and X memory space. The extra P memory is required since the assembler program for the transmitter DSP56001 was larger than the available on-chip P memory. The external data RAM is necessary to store the filter lookup table, which typically has more than 256 tap weights, making it impossible to store as a contiguous block in the on-chip RAM. The memory chips used were HM65728s.

The program and data external memory are mapped onto the memory spaces P:$1000 - P:$17FF and X:$1000 - X:$17FF respectively (Section 5.2, which describes the transmitter software, explains how these memory spaces are accessed). Figure A.1 shows the circuit diagram for the external program and data memory. The memory and the decoding logic are wire-wrapped onto the prototyping board development area.

5.2.3 CLOCK GENERATING CIRCUIT

The clock signals are required for the sampling interrupt for the microprocessor and the multiple-rate clock for the pseudorandom bit sequence (PRBS) generator. It was explained in section 5.1.2 that the sampling frequency of the system is 38.4kHz. The PRBS generator must be clocked at rates of 2400, 4800 and 9600Hz, all of which are integral denominators of 38.4kHz. The clock circuit is thus very simple, as all clock frequencies are related by powers of 2.

A very popular circuit utilizing an inverter gate is used to construct a 6.144MHz crystal oscillator. Obviously the 6.144MHz frequency must be divided down to get the desired clock signals. The 38.4kHz clock is achieved by suitable connection of a 74HC160 decade counter and four D-type flip-flops. Cascading four more flip-flops results in the desired frequencies of 9.6kHz, 4.8kHz and 2.4kHz.

Changing the bit rate requires that the processor be interrupted by the host so that it can modify certain parameters for the new bit rate. The clock rate for the PRBS generator is therefore made selectable under software control from the DSP56001 processor. The bit rate is selected by asserting the processor Port C pins 4 and 5, which are programmed for general purpose output. These pins are connected to an analogue multiplexer, along with the clock signals. The overall clock circuit diagram is given in Figure A.2.

5.2.4 DIGITAL-TO-ANALOGUE CONVERTER

The DSP56001 performs the signal processing stages up to and including baseband modulation, the net result being a sequence of sampled data values representing the transmitted signal waveform. It is therefore necessary to interface a digital-to-analogue converter (DAC) to the data bus of the processor. A 12-bit DAC (4096 levels), the DAC1221 is used

in this circuit. This device has a settling time of 500nsec, and can therefore be used at a sampling frequency of 38.4kHz (T_s = 26μsec). The DAC1221 is configured for bipolar operation, the input/output relationship of which is shown in Table 5.1.

DIGITAL INPUT												V_{out}
1	1	1	1	1	1	1	1	1	1	1	1	V_{ref} X 1022/1024
1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	$-V_{ref}$

Table 5.1 - DAC bipolar input/output relationship.

The DSP56001 microprocessor uses two's complement notation to represent binary numbers, whereas the DAC1221 uses the relationship shown in Table 5.1. It is therefore necessary to convert from two's complement to the DAC format. Referring to Table 5.1, it is seen that conversion is achieved by simply inverting the MSB of the two's complement 12-bit number to get to the DAC format.

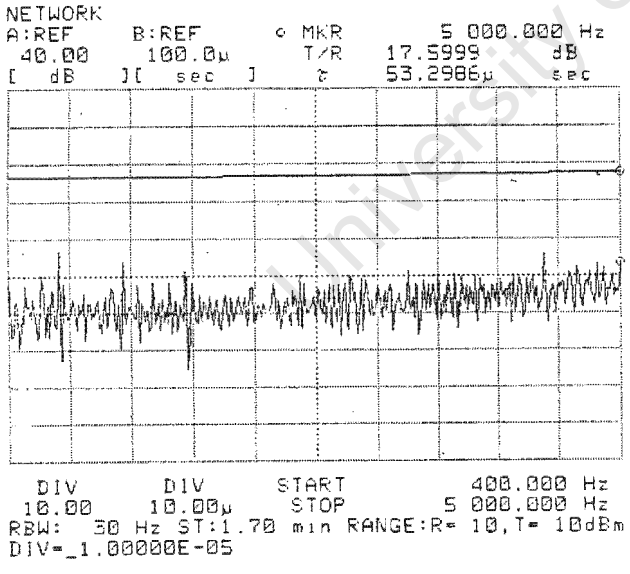
The circuit diagram for the digital-to analogue converter is shown in Figure A.3. The DAC is memory-mapped onto the data bus of the transmitter. A set of latches interfaces the data inputs of the DAC to the DSP56001 data bus. The quantized output of the digital-to-analogue converter is taken to a Butterworth smoothing filter, which filters off the higher frequency spectral replicas.

5.2.5 OUTPUT ANALOGUE SMOOTHING FILTER

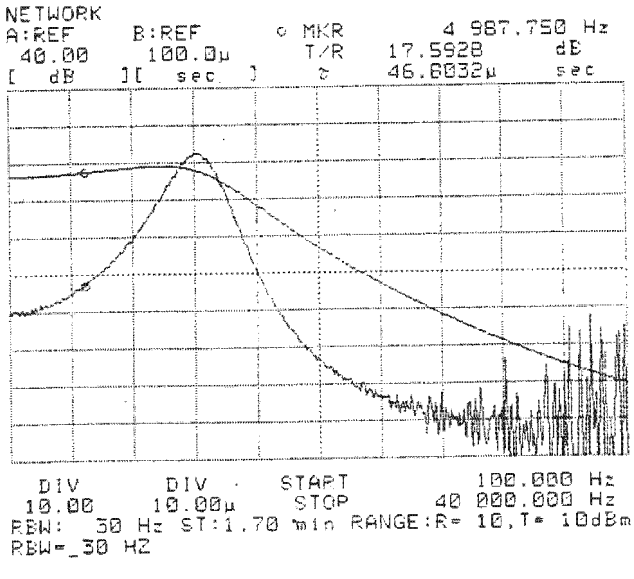
The output of the DAC circuit described in the previous section is a quantized signal. Due to the sampled nature of the signal, spectral replicas appear at multiples of the sampling frequency (38.4kHz). These spectral replicas must be removed by analogue lowpass filtering before transmission, to avoid interference effects and to contain

the transmitted energy to within the desired frequency range.

If the sampling frequency of the system is high enough, the design of this analogue lowpass filter is non-critical. A 6th order Butterworth filter was chosen for this filter function. With a designed cutoff frequency at 12kHz, the phase characteristic in the region 0 - 4kHz is approximately flat. The suppression at 15kHz is also good (actual measurements are given later). A voltage-controlled voltage-source (VCVS) active circuit implements the filter design. The use of an active circuit eliminates the need for inductors in the filter and the VCVS circuit is easy to implement. It is fairly sensitive to component values [5.1], but the design here does not require exact frequency characteristics. Figure A.4 in Appendix A shows the Butterworth filter circuit. Figure 5.5 a) and b) show the measured filter transfer function and group delay characteristic.



a)



b)

Figure 5.5 - Measured Butterworth filter transfer function and group delay characteristic.
a) - 400Hz to 5kHz.
b) - 100Hz to 40kHz.

5.2.6 PSEUDORANDOM BIT SEQUENCE (PRBS) GENERATOR

REQUIREMENT FOR A RANDOM DATA SOURCE

The transmitter requires a random data source for the binary input sequence. The random property of the data is vital to produce a continuous-envelope signal spectrum for transmission. An illustrative example to show the effect of repetition of a signal in the time domain is that of a non-return to zero (NRZ) bit stream. A truly random NRZ signal will have a continuous sin(x)/x spectrum [5.2]. If the bit sequence is not random and is repeated every n bits, the spectrum is now not continuous, and in fact spectral lines appear at multiples of the repetition rate.

The implication of this result in this project is that a periodic data source will produce a periodic analogue output signal from the transmitter. This signal will therefore have its energy concentrated at discrete frequencies spaced at intervals n/T_r, where T_r is the period of repetition. This situation can lead to false results when taking noise performance measurements, and must therefore be avoided.

PRBS IMPLEMENTATION

A PRBS generator of sufficient length can be used as a data source in a test situation. A PRBS generator produces a bit sequence of a predefined finite length. The longer the bit sequence, the closer the approximation to a random data source. A common method of implementation is to cascade a sequence of m shift-registers. The output of register m and some register n are modulo-2 added and fed back to the input of register 1. If the right n is chosen, the output will produce a sequence of 2^m-1 bits long.

For a PRBS generator of 22 registers, modulo-2 addition of registers m=22 and n=21, with the result fed back to register 1, will result in a bit sequence of repetition

length $2^{22}-1 = 4194303$. The repetition period of the modem transmitter output needs to be calculated now to quantify the effect of the periodicity of this data source. If the modem is operating at 9600 bps, the output will repeat after every 4194303 bits, or after $T_r = 4194303/9600$ seconds. Thus the repetition period is $T_r = 437$ seconds, and the output signal will have spectral lines at frequency intervals of 0.0022 Hz. This closely approximates an ideal random data source.

The PRBS generator circuit is shown in Figure A.5. Modulo-2 addition is implemented using the logic EXOR function, as shown in the diagram. A second feedback circuit from registers 2 and 3, giving a repetition length of $2^3-1 = 7$ bits, is also included in the circuit. This short sequence makes it easier to debug the circuit in the development stage, as it is impractical to monitor a long bit sequence. Switch S1 is included to 'kickstart' the circuit in the event of the all-zero state occurring at start-up time.

5.3 REAL-TIME SOFTWARE

5.3.1 INTRODUCTION

The transmitter software includes the Host Pascal program and the DSP56001 assembler transmitter program. Both the Pascal and the assembler real-time programs are very similar to the simulation programs described in section 5.1, although more detailed program descriptions are given here. The software can be thought of as the 'brains' of the transmitter, since it implements the more important transmitter functions. These functions are shown in block diagram form in Figure 5.6. The software and the DSP56001 produce signal samples that are converted by the DAC circuitry to analogue format for transmission. The transmitter DSP56001 also provides various clock and decoding signals for the transmitter and receiver hardware.

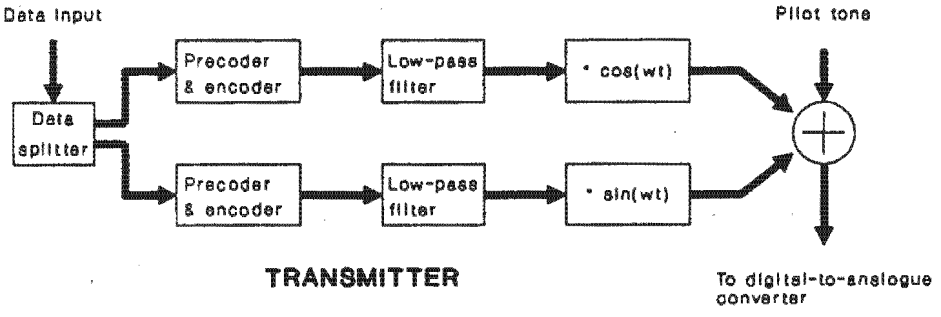


Figure 5.6 - Transmitter functions implemented by the DSP56001.

The Pascal program is described first, as its function is largely supervisory. The assembler program, which is resident in the DSP56001, performs the signal processing tasks and is under the control of the Pascal program.

5.3.2 THE HOST PASCAL PROGRAM - 'TRANSMIT.PAS'

INTRODUCTION

The Pascal program is the supervisory program for both the transmitter and the receiver. It is described almost wholly in this chapter, with occasional reference being made to it from Chapter 6, which describes the receiver design. The Pascal program allows control over the operation of the experimental modem by performing the two following functions:-

1. Transfers the lowpass filter lookup tables to the transmitter and receiver DSP56001 chips. These tables are stored in the computer memory in the form of files.

2. Programs the DSP56001 transmitter (and receiver) for a specified bit rate. The bit rate is selectable from the computer keyboard, and can be changed during transmission.

PASCAL PROGRAM OPERATION

The program listing for this program (TRANSMIT.PAS) is given in Appendix E. The structure of the program is best introduced and described by the flowchart in Figure 5.7.

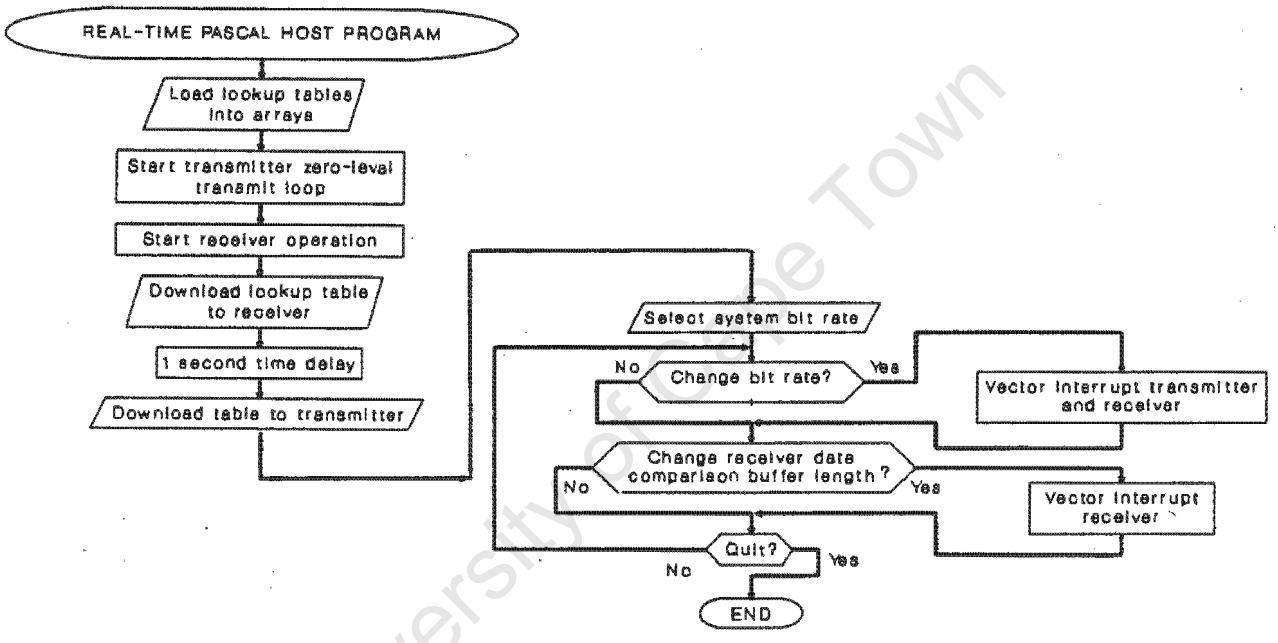


Figure 5.7 - Flowchart for real-time Host Pascal program.

The sequence of events in the program, with reference to Figure 5.7, is as follows:-

1. Read the filter lookup table from a file into Pascal arrays.

This is done using the procedure 'read_from_file'. The lookup table contains a root raised-cosine response, and is thus used for both the transmitter and the receiver. The program listing in Appendix E is for the case of a transmitter sampling rate of 38.4kHz, and a receiver sampling rate of 19.2kHz. The receiver lookup table is half

the length of the transmitter, as it is the same width in symbol intervals, but has half the number of samples per symbol. In the given listing the transmitter filter is 22 symbols wide with 16 points per symbol.

2. Start the transmitter DSP56001 program.

Starting the transmitter DSP56001 program is done by using the procedure 'DSP_Go'. After an initialization routine, the transmitter DSP56001 enters a loop that transmits a continuous zero-level output. This is done so that the receiver can calibrate the d.c. offset due to the analogue-to-digital converter at its input stage. The transmitter DSP56001 waits in this state until flagged by the Pascal program.

3. Start the receiver DSP56001 program.

This is done as above using the routine 'DSP_Go'. The receiver assembler program enters a loop to wait for the lowpass filter lookup table to be downloaded into its memory.

4. Download the receiver filter lookup table.

The root raised-cosine lookup table for the receiver is downloaded to the DSP56001 via the host interface in the procedure 'lookup_to_receiver'.

5. Wait for 1 second.

A 1 second delay occurs in the Pascal program to allow the receiver DSP56001 time to read the zero-level d.c. offset, which is currently being transmitted from the transmitter DSP56001. The Pascal program then flags the transmitter, which exits the zero-level transmit loop.

6. Select the bit rate and download the transmitter lookup table.

This is done in the routine 'lookup_to_transmitter'. A value is written to the transmitter DSP56001 instructing the chip to operate at either 2400, 4800 or 9600 bps. The transmitter lookup table is downloaded to the DSP56001 in the same manner as in point 4 above. The transmitter then enters its operational state and starts signal transmission. By this stage the receiver is also operational.

7. Loop until 'quit' selected.

The Pascal program now loops through an option-checking routine called 'bit_rate_change' until the 'q' key is pressed, which terminates program operation. The routine 'bit_rate_change' allows changing the system bit rate and changing the length of the data comparison buffer in the receiver DSP56001 (this is described fully in the receiver section). Selecting a bit rate causes the Pascal program to vector an interrupt to a specified location in the transmitter and receiver DSP56001 processors. The assembler interrupt procedure then changes the necessary parameters in the microprocessors for the new bit rate or data comparison buffer length. This is discussed in the assembler program description in section 5.3.3.

5.3.3 THE DSP56001 ASSEMBLER PROGRAM - 'TRANSMIT.ASM'

INTRODUCTION

The transmitter DSP56001 assembler program forms the core of the transmitter modulation process. The majority of the signal processing functions are performed by the DSP56001, from coding the binary input data to outputting modulated waveform sample values. An important software consideration is the use of the memory spaces in the DSP56001. Figure 5.8 shows how the memory space of the DSP56001 is utilized by the assembler program.

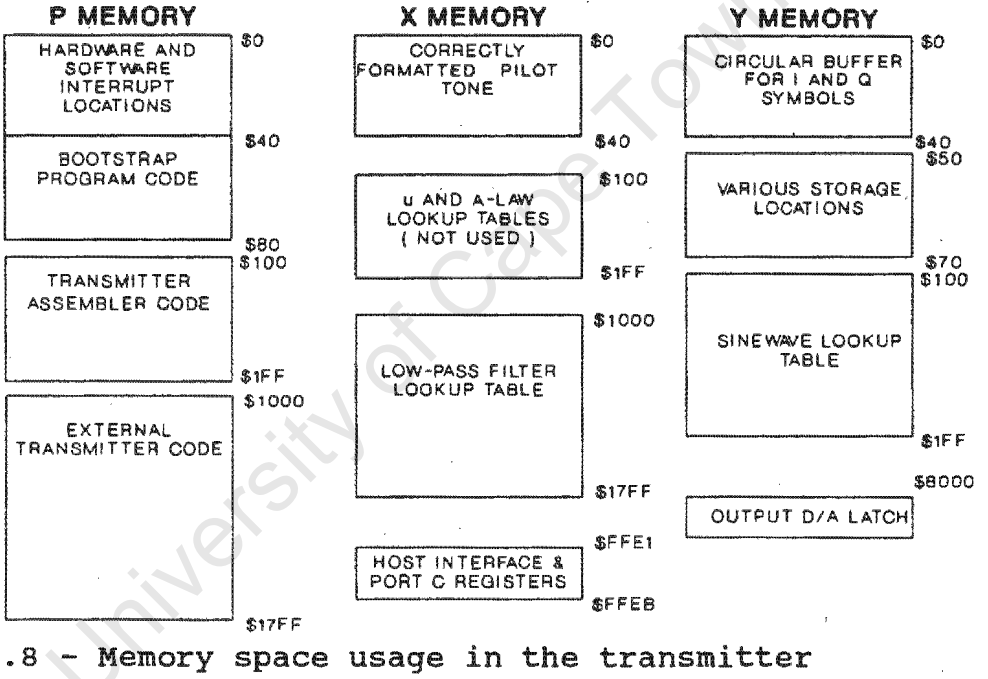


Figure 5.8 - Memory space usage in the transmitter DSP56001.

TRANSMITTER ASSEMBLER PROGRAM OPERATION

The transmitter assembler program and the DSP56001 perform the signal processing functions shown earlier in Figure 5.6. These functions are described briefly here before a detailed program description is given. The DSP56001 reads in the binary data from the PRBS generator at either 2400, 4800 or 9600 bps. This is converted into a quaternary (4-level) symbol sequence, and split into I and Q channels. The symbols in the I and Q channels are coded according to class

1 PRS coding rules, and stored in one buffer (with the I and Q values interlaced). This buffer is the same symbol length as the filter lookup table. The symbol sequences for the I and Q channels are convolved with the lowpass filter lookup table at the sampling rate (here it is 38.4kHz) to give two baseband signal values. These two values are multiplied by I and Q sinewave carriers of 1800Hz, and the resulting values are added to give a QPRS waveform sample value. This is written to the digital-to-analogue circuitry, where it is suitably converted for transmission.

The listing for the program 'TRANSMIT.ASM' is given in Appendix C. As in the case of the Pascal program, the assembler program is best described by the use of flowcharts. The program flow is divided into 2 sections, the initializing routine and the execution routine.

TRANSMITTER INITIALIZING ROUTINE

The flowchart for the initializing routine is shown in Figure 5.9.

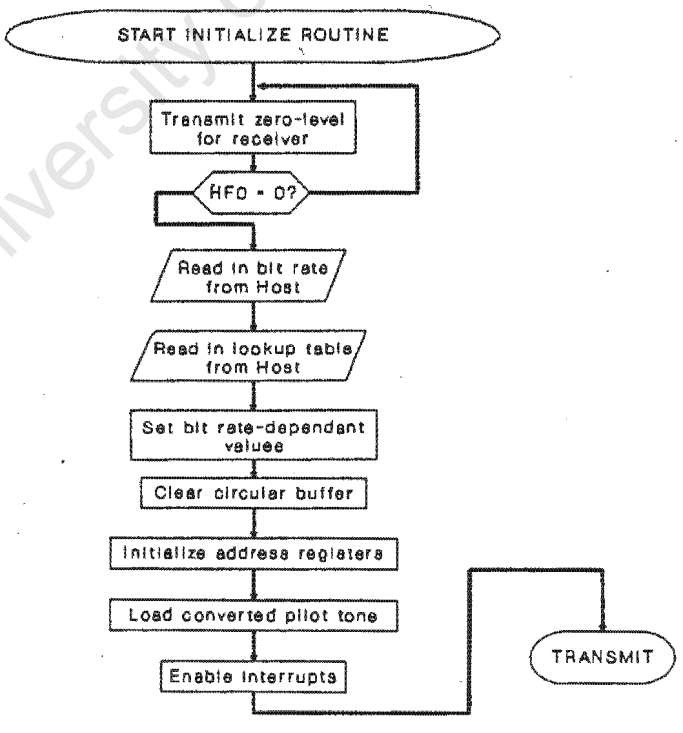


Figure 5.9 - Flowchart for transmitter assembler program initializing routine.

The sequence of events, with reference to Figure 5.9, is as follows:-

1. Transmit zero level for receiver.

This is done using the subroutine 'level'. The DSP56001 transmits a zero level until flagged by the host. This is done to allow the receiver to read the zero level d.c. offset at its input.

2. Read in the bit rate and the lookup table.

This is done in the subroutine 'in'. The host computer writes the bit rate to the DSP56001 via the host interface data register. The bit rate indicator is read in from the host interface and stored in the Y memory location Y:baud.

A looping routine then loads the lookup table into the DSP56001 via the host interface. The table is stored in external X memory starting at location X:$1000.

3. Set bit rate-dependant values.

This is done using the subroutine 'init'. Before the various interrupts are enabled to allow transmitter operation, certain parameters that depend on the bit rate must be set. Port C is configured as general purpose I/O .

4. Clear the circular buffer.

The circular buffer contains the PRS-encoded multilevel symbol stream for both the I and the Q channels. The total length of the circular buffer is therefore twice that of the lookup table (in terms of symbols). Thus for a lookup table that is 32 symbols long (or 512 points), the circular buffer will span 64 memory locations. The operation of the buffer will be explained later. The buffer occupies the lower

section of Y memory from location Y:$00 up to Y:$3F (a maximum of 64 locations).

5. Initialize the address registers.

The following address registers are used:-

r0 - lookup table pointer.
r1 - counter to count interrupts.
r2 - I carrier pointer.
r3 - pilot tone pointer.
r4 - circular buffer pointer.
r5 - not used.
r6 - Q carrier pointer (90° shifted from I carrier).
r7 - pilot tone pointer.

6. Load converted pilot tone.

This is done in the routine 'pilot'. The correctly formatted values for the pilot tone are stored in Y memory locations Y:$0 - Y:$3F (64 locations). Pre-formatting these values from the on-chip sinewave lookup table saves calculation time when writing to the DAC. The pilot tone frequency is 600Hz. For a sampling rate of 38.4kHz the formatted tone lookup table must therefore have 38.4/0.6 = 64 samples. These samples are read from the on-chip 256-point table in Y memory, converted into the right format, and stored in the new Y memory location.

7. Enable interrupts.

This is done in the routine 'enable'. This is the final initialization step, as enabling the interrupts allows transmission to proceed. The Interrupt Priority Register is set to enable the /IRQB interrupt, which is used for 38.4kHz sampling, and specify the priority of the Host Command interrupt.

TRANSMISSION ROUTINE

The DSP56001 is now ready for transmission. The flowcharts for signal transmission are shown in Figure 5.10. Points X and Y indicate where the two flowcharts join each other.

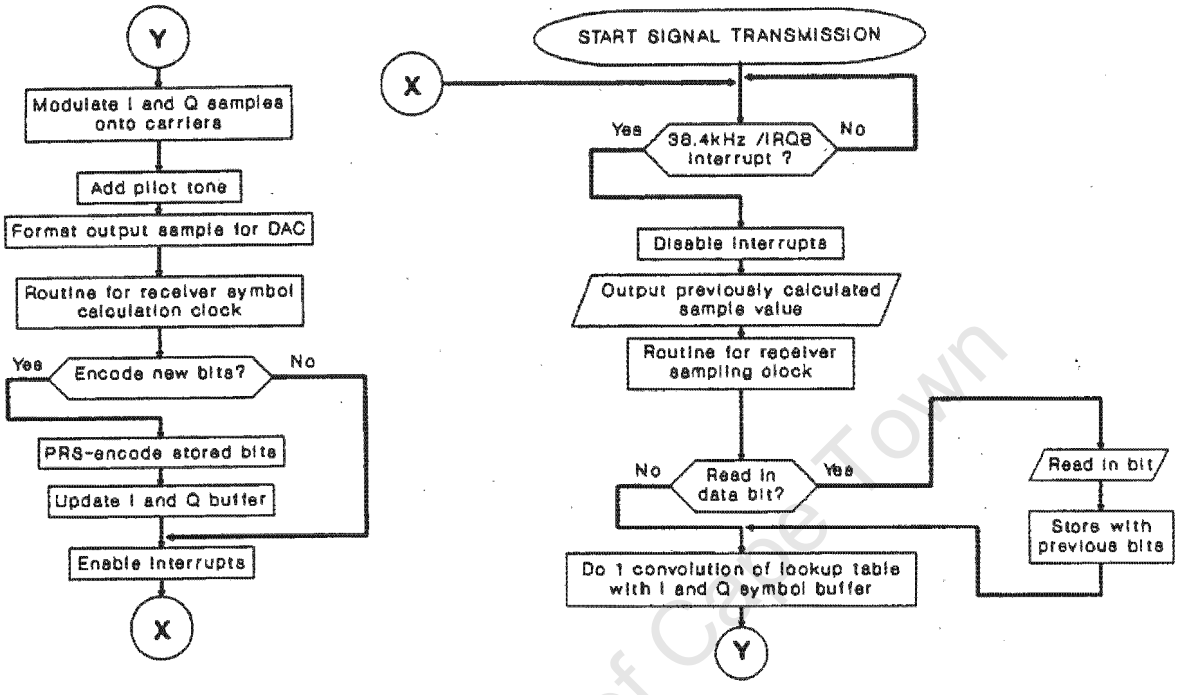


Figure 5.10 - Flowchart for transmitter assembler transmission program .

The sequence of events, with reference to Figure 5.10, is as follows:-

1. Wait for 38.4kHz /IRQB interrupt.

The transmitter waits in an endless loop (called 'main') until the /IRQB interrupt is activated. This 38.4kHz signal is supplied by the clock generating circuit described in section 5.2.3. The interrupt causes a jump to the interrupt service routine 'calc'.

2. Disable interrupts.

This is the first step in the interrupt service routine. It prevents multiple interrupts from occurring if the interrupt signal is noisy.

3. Output previously calculated sample value.

The transmitted waveform sample value calculated in the previous interrupt cycle is written to the DAC circuitry. This is done using the routine 'out'. The previous value is stored in Y:numout. Writing the previous value as opposed to calculating and then writing the present value ensures that no timing jitter is introduced between samples, since the program calculation time is variable.

4. Routine for receiver sampling clock.

Two of the clock signals for the receiver are generated in the routine 'out'. These include the carrier reset clock (Port C bit #7) and the 19.2kHz sampling clock (Port C bit #8). These signals simulate the clocks that would be provided by a phase-locked loop at the receiver. A low level on the carrier reset clock indicates that the receiver I carrier (used for demodulating the received signal) must be reset to its zero position and the Q carrier to a 90° shifted position. The carrier clock is set depending on the transmitter carrier position.

5. Must a data bit be read in?.

The data rate is either 2400, 4800 or 9600 bps. The sampling rate is 38.4kHz. Therefore a bit must be read into the DSP56001 from the PRBS generator every 16, 8 or 4 interrupts, depending on the bit rate. This is taken care of in the subroutine 'count'.

If a bit must be read in, program flow continues at the subroutine 'read'. The bit is read in from the PRBS output, and has to be combined with previous bits for coding into the PRS-formatted symbols, depending on the bit rate. 2400 bps transmission, or 3-PRS, requires 1 bit/symbol. A new symbol is therefore calculated for every bit that is read in. 4800 bps, or 9-QPRS, requires 1 bit/symbol per channel.

This means that the I and Q symbols are calculated after every 2 bits that are read in. 9600 bps, or 49-QPRS, requires 2 bits/symbol per channel. The I and the Q symbols are calculated after every 4 bits read in. The relevant bits are stored in the buffer location Y:store.

6. Do 1 convolution of the lookup table with the I and Q symbol buffer.

The symbol buffer contains the PRS-encoded symbol impulse values (3-level, I channel for 2400 bps, 3-level I and Q channel for 4800 bps, 7-level I and Q channel for 9600 bps) that need to be lowpass filtered to produce a baseband signal. This is done by convolving the symbol impulse values with the lowpass filter lookup table. The convolution process is best described by the sequence shown in Figure 5.11. In this example, the lookup table is taken to be 6 symbols long with 4 points per symbol, to simplify the explanation. The actual lookup table in the assembler program is up to 32 symbols long with 16 points per symbol, but exactly the same method of convolution is used.

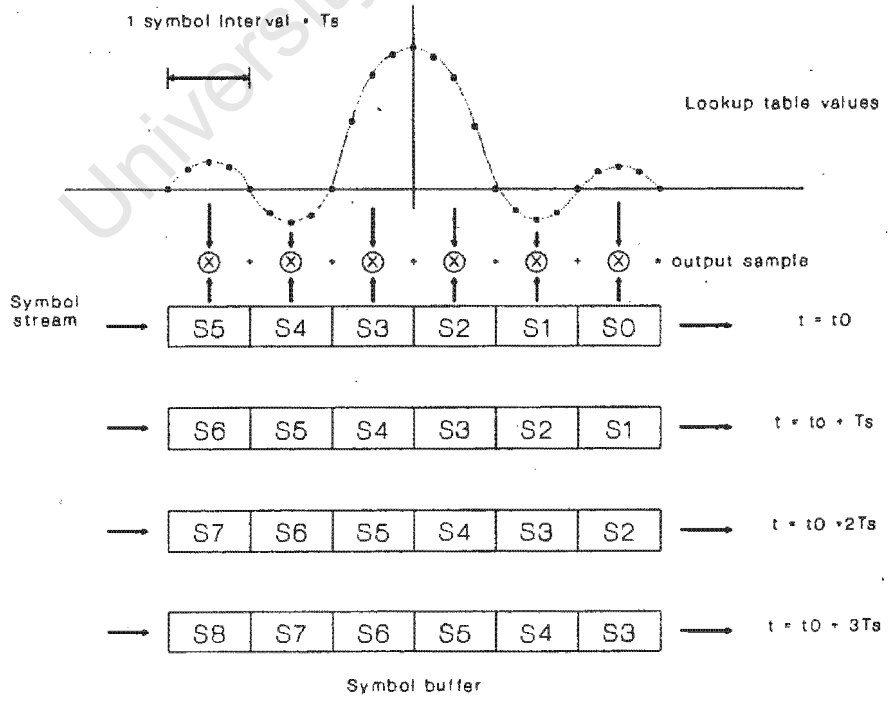


Figure 5.11 - Convolution of the lowpass filter lookup table with the PRS symbol stream.

To produce a baseband signal, each impulse symbol value Sn in the symbol buffer must be convolved with the entire lookup table. For one impulse, this would involve multiplying the impulse value by the first lookup table value, and outputting the resultant sample value. At the next sample instant, the impulse value is multiplied by the next lookup table value, and that result becomes the output sample. Thus for each sampling instant, one multiplication is performed. The resultant waveform would just be an amplitude-scaled version of the lookup table, which is of course the impulse response. Note that the symbols here are impulses, as they only span one sample width and not one symbol width (in this example there are 4 samples per symbol interval).

For a sequence of impulse symbol values, the situation is slightly more complicated. Looking at Figure 5.11, it is seen that the lookup table spans 6 symbol intervals. Therefore at any one time there will be 6 symbol impulse values in the buffer that have to be convolved with the lookup table. At time t = t0, symbols S5, S4, S3, S2, S1 and S0 are multiplied by the 1st, 5th, 9th etc. samples respectively. The products are summed to give the filter output at t = t0. At t = t0 + Ts/4, the whole system is shifted by 1 sample instant. S5 is now multiplied by the 2nd lookup table sample, S4 is multiplied by the 6th sample, and so on. At time t = t0 + Ts, however, a new symbol S6 enters the buffer and the oldest symbol S0 is lost. The multiplication process repeats itself again starting at the 1st lookup table sample.

Thus each symbol impulse value is convolved completely with the lookup table. The individual impulse responses are summed to give the composite baseband output signal. In the assembler program listing in Appendix C, the lookup table is 22 symbols long and there are 16 points/symbol. Therefore 22 multiplications are performed and the results summed for each output sample in each channel. This gives a total of 44

multiplications per baseband output sample that are performed for each convolution pass. This is done in the convolution routine 'calc'.

7. Modulate I and Q samples onto carriers.

The calculated baseband I and Q samples need to be modulated onto quadrature 1800Hz carriers before transmission. For a sampling rate of 38.4kHz, an 1800Hz carrier needs 21.33 samples per cycle, or 64 samples per 3 cycles. The on-chip sinewave lookup table is 256 points long, therefore to get the required 64 samples in 3 cycles (768 points), every 12th point in the sine table is read to generate the carriers. Multiplying the I and Q baseband samples that result from point 6 above by the correct values in the sine table results in a modulated quadrature signal.

8. Add pilot tone.

The pilot tone is intended to provide the receiver with the synchronizing information, assuming a phase-locked loop (PLL) is being used at the receiver. In this project however, the synchronizing signals are supplied directly by the transmitter. This was due to the time limitations on the project. The pilot tone is described here nevertheless for completeness.

The transmitter requires the 3 synchronizing signals:-

- the 19.2kHz sampling clock.
- the 2.4kHz symbol calculation clock.
- the 1800Hz carrier reset clock.

If a 600Hz pilot tone is added to the transmitted signal with the correct phase, all three of the above clock signals can be extracted unambiguously at the receiver. When the received signal with the pilot tone is demodulated at the receiver, the resultant tones are at (1800 - 600) = 1200Hz

and (1800 + 600) = 2400Hz. The 2400Hz signal is completely filtered out by the lowpass filter at the receiver. The 1200Hz signal is at the edge of the lowpass filter band and is thus partially filtered. The zero-crossings of the 1200Hz signal, however, coincide with the symbol sampling instants, causing minimal interference in the recovered baseband signal. This is shown in Figure 5.12.

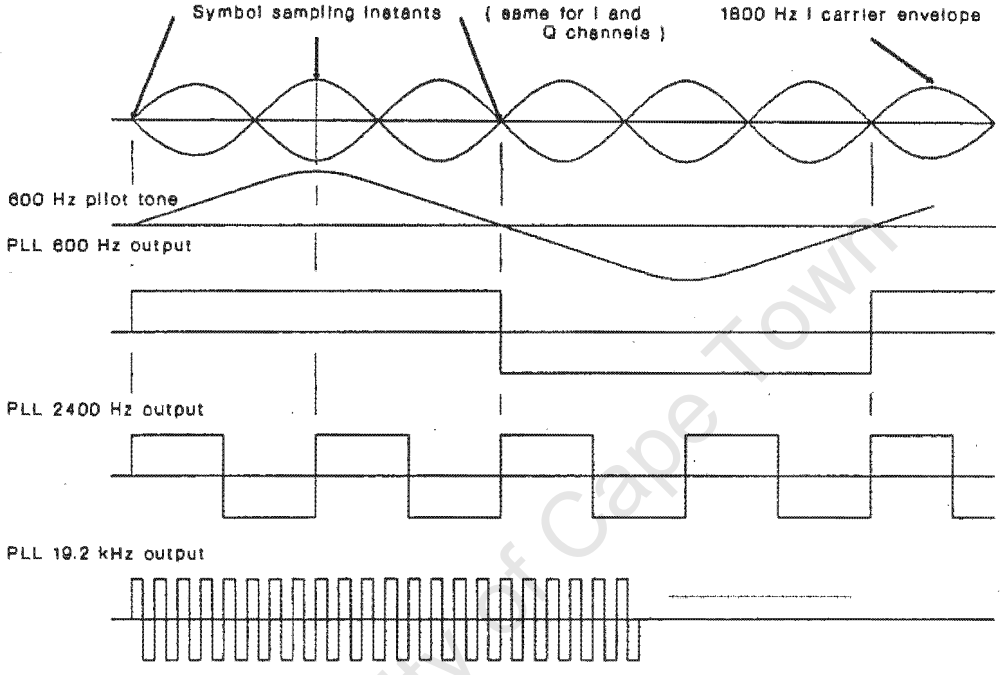


Figure 5.12 - Pilot tone addition and timing signal extraction.

9. Format output sample for DAC.

The resulting modulated signal value, with the pilot tone value added to it, needs to be formatted before it is compatible with the 12-bit DAC. The formatting technique is described in section 6.2.3. The formatted number is stored in Y:numout.

10. Routine for receiver symbol calculation clock.

The receiver must calculate the received symbol values at the correct instants. The symbols rate is 2400 baud, so a 2400Hz clock is provided by the transmitter for that purpose. This occurs in the routine 'sym', which also checks

if a new symbol must be calculated for storage in the buffer. The transmitter sampling rate interrupt occurs at 38.4kHz, so the 2400Hz clock is generated by setting a Port C pin once every 16 interrupts, and clearing the pin shortly afterwards. This pin is connected to the receiver.

11. Must the new bits be encoded?

The memory location Y:store stores the most recent 1, 2 or 4 bits read in from the PRBS generator. The symbol rate is 2400 baud irrespective of the bit rate, so new symbols must be calculated at a rate of 2400Hz. The /IRQB interrupt occurs at 38.4kHz, so a new symbol must be calculated every 38.4 / 2.4 = 16 interrupts. This condition is checked in the routine 'sym'.

12. Encode the new bits according to Partial Response coding rules.

The bits stored in Y:store are PRS-encoded in the routine 'code'. This routine codes the bits according to the current bit rate of either 2400, 4800 or 9600 bps. The coding procedure is as follows:-

1. Isolate the I channel bit(s).

2. Add an offset to the bit(s) for calculation purposes. For 2400 and 4800 bps, offset is 2 and for 9600 bps offset is 4.

3. The previous precoded symbol is subtracted from this.

4. The result is taken modulo-2 for lower bit rates, modulo-4 for 9600 bps. The last three steps complete the precoding for partial response signalling.

5. The result is stored for use in the next I channel calculation.

6. Add the past stored symbol algebraically to the result of step 4. This completes the encoding for partial response signalling.

7. The result is an integral number between 0 and 6 for 9600 bps or between 0 and 2 for 4800 and 2400 bps. Subtracting 3 from the 7-level signal level-shifts the baseband signal to the range -3 to +3. Subtracting 1 from the 3-level signal shifts the baseband signal to the range -1 to 1.

8. This value, the partial response coded symbol, is stored in the next location in the circular buffer.

9. Steps 1 to 8 are repeated for the Q channel.

10. The circular buffer now contains the new I and Q symbol values. If the bit rate is 2400 bps, there is no Q channel, and the buffer contains only I channel symbols. For a data rate of 2400 bps the buffer is half the length of the buffer in the case of the higher bit rates.

13. Enable interrupts.

The interrupt service routine is now complete, and the /IRQB and host interrupts are re-enabled by writing the appropriate value to the Interrupt Priority Register. The program flow returns to the loop 'main', and the DSP56001 microprocessor waits for the next 38.4kHz /IRQB interrupt.

5.4 SUMMARY

The transmitter consists of both hardware and software. The software consists of two programming structures - the host Pascal program and the DSP56001 assembler program. The Pascal program controls the system operation by setting the bit rate, downloading the desired lookup table, and starting the system operation. The assembler program is the 'brains' of the system and performs most of the signal processing operations to produce a modulated bandlimited signal for transmission. The hardware supplies the system clock, the bits for transmission, and converts the digital signal from the DSP56001 into analogue format. The hardware also filters off the spectral replicas that result from sampling.

5.5 REFERENCES

[5.1] P. Horowitz & W. Hill, *The Art of Electronics*, Cambridge University Press, Cambridge, 1986, pg. 157.

[5.2] F. Stremler, *Introduction to Communication Systems*, 2nd. Edition, Addison-Wesley, Reading, Massachusetts, 1982, pg. 534.

CHAPTER 6. THE RECEIVER DESIGN

6.1 RECEIVER SIMULATION

6.1.1 INTRODUCTION

As in the case of the transmitter, the operation of the receiver was simulated before the actual hardware was built or real-time software was written. The purpose of the receiver simulation was slightly different to that of the transmitter simulation. The main function here was to verify the demodulation process, and to get a working assembler program structure that could be used in the real-time program. The program listings reflect the case of a receiver sampling rate of 19.2kHz. The sampling frequency is not as critical in the receiver as in the transmitter, since the aperture effect does not affect the receiver.

The steps taken in performing the receiver simulation are very similar to those in the transmitter simulation. The steps taken were:-

1. Generate a root-raised cosine lookup table for the transmitter using the program 'SIMPSON.PAS'.

2. Run the transmitter simulation using this lookup table.

3. Download the receiver assembler program to the receiver DSP56001.

4. Download the lookup table to the receiver DSP56001.

5. Write the waveform values from the transmitter output file to the receiver, and read the result of the demodulation process from the receiver into a different file.

The simulation programs are only described in a broad sense, to illustrate their operation. The real-time programs in later section 6.3 are described in detail.

6.1.2 THE HOST PASCAL PROGRAM - 'RXIQ.PAS'

The Pascal program written for the simulation is called 'RXIQ.PAS', and a listing of this program is given in Appendix G. It is very similar in structure to the transmitter host Pascal simulation program 'TXIQ.PAS'. Figure 6.1 shows the flowchart for this program.

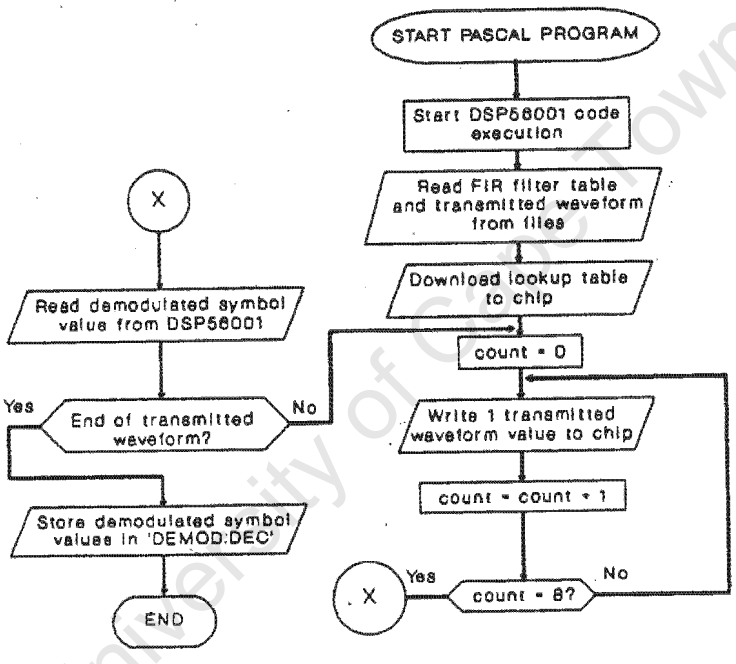


Figure 6.1 - Flowchart for receiver host Pascal simulation program 'RXIQ.PAS'.

The program listed reflects a bit rate of 9600 bps (49-QPRS), and a sampling frequency of 19.2kHz. The listing also reflects the case of a simulated transmitter frequency of 38.4kHz. The receiver lookup table is half the length of the transmitter lookup table since the receiver sampling frequency is half that of the transmitter. The symbol width of both filters is the same.

6.1.3 THE DSP56001 ASSEMBLER PROGRAM - 'RXIQ.ASM'

The receiver simulation assembler program is called 'RXIQ.ASM', and the program listing is given in Appendix G. The program flowchart is shown in Figure 6.2.

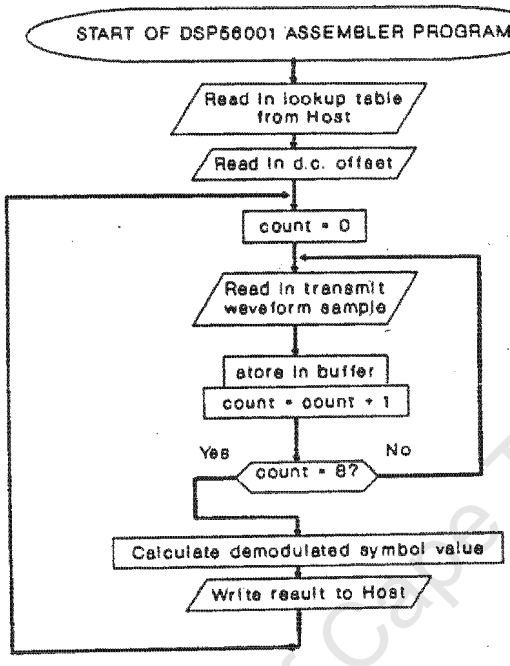


Figure 6.2 - Receiver assembler simulation program flowchart.

The DSP56001 reads in the transmitted waveform samples from the host computer via the host interface, demodulates and lowpass filters them, and outputs the sampled baseband result back to the host. It is not necessary to lowpass filter every sample, as the only output values of interest are the results at the baseband symbol sampling instants. The received waveform contains 8 points/symbol (corresponding to a 19.2kHz sampling rate), therefore the symbol outputs will occur once every 8 samples.

After the lookup table has been read in and stored in external memory, the DSP56001 reads in the d.c. offset of the transmitted waveform. The d.c. The offset arises from the analogue-to-digital converter in the receiver, which has an input range of 0 to 5V.

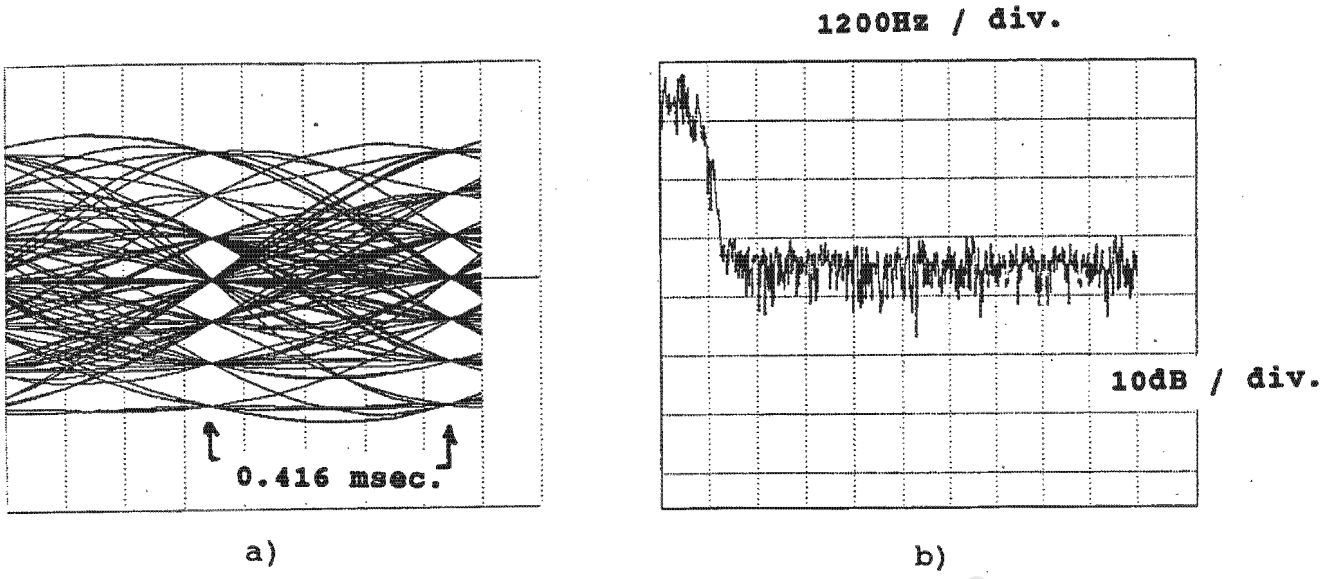
The samples are then read in from the host, multiplied by a carrier for demodulation, and stored in a sample buffer. This buffer is the same length as the lookup table. After every 8 samples read in, a lowpass filtered output value (which is the demodulated and filtered baseband symbol value) must be calculated. This requires doing one convolution pass of the lookup table with the input buffer. The program in Appendix G demodulates only the I channel, as the Q channel is a simple extension of the same method of demodulation.

6.1.4 RECEIVER SIMULATION RESULTS

It was stated in the preceding section that it is not necessary to filter every received sample to recover the transmitted symbols. For the purposes of displaying the results in this text, however, the receiver simulation was modified to produce all the samples of the demodulated I channel baseband signal. The receiver simulates a sampling rate of 19.2kHz. Only one receiver filter characteristic is displayed here - a root raised-cosine filter of length 160 with α = 0.2. Figure 6.3 a) shows the recovered baseband signal, and Figure 6.3 b) shows the baseband signal spectrum obtained by taking the FFT of the baseband signal.

BRIEF DISCUSSION OF RECEIVER SIMULATION RESULTS

Observing the results displayed in Figure 6.3, it is seen that the receiver software structure operates correctly. The symbol sampling instants are clearly seen in the eye pattern. There is minimal ISI at the receiver sampling instants, illustrating the effect of cascading two root raised-cosine filters. The baseband signal spectrum shows sideband suppression of approximately 30dB. A sampling rate of 19.2kHz thus appears to be sufficient for the receiver, and is therefore chosen for the real-time receiver.



a) b)

Figure 6.3 - Receiver simulated results for 160 length filter, α = 0.2.
a) Baseband 7-level output.
b) Baseband spectrum.

6.2 RECEIVER HARDWARE

6.2.1 INTRODUCTION

The receiver hardware is less complex than the transmitter hardware, and consists of two major sections. These are the analogue-to-digital converter and the wideband noise generator. The noise generator is used for test purposes. The receiver DSP56001 also has external X memory wire-wrapped on to it, but this is identical to the X memory connected to the transmitter DSP56001. The hardware is described in the following three sections. The relevant circuit diagrams are given in Appendix A.

6.2.2 EXTERNAL X MEMORY FOR THE DSP56001

This is identical to the X memory configuration of the transmitter. The memory occupies the space X:$1000 - X:$17FF. This memory is used to store the lowpass filter lookup table.

6.2.3 RECEIVER ANALOGUE-TO-DIGITAL CONVERTER AND OUTPUT DAC

The circuit diagram for the receiver analogue-to-digital converter (ADC) and output DAC are shown in Figure A.6 in Appendix A.

INPUT ANALOGUE-TO-DIGITAL CONVERTER

The received analogue signal from the transmitter must be converted to digital data before the receiver DSP56001 can process it. The ADC chip used in the receiver is the National ADC0820. This is an 8-bit high speed microprocessor compatible ADC chip. The chip uses half-flash conversion techniques to give a conversion time of 1.6 μsec, which implies an upper sampling frequency limit of between 500 and 600kHz. As stated in section 6.1, the receiver sampling rate is 19.2kHz, which is well below the limit of the ADC0820. The 8-bit resolution gives a signal-to-quantization noise of 48dB. This quantity is sufficiently higher than the expected signal-to-noise ratio due to the channel to induce errors at the receiver. The chip is memory-mapped to the location Y:$2000. The data pins are connected directly to the external data bus of the DSP56001. Subsequently reading from Y:$2000 will read these bits into the DSP56001. These connections are shown in Figure A.6.

The analogue input range to the ADC0820 is 0 - 5V, so the a.c. signal from the transmitter must be both level-shifted and attenuated accordingly. This is done very simply by using an op-amp in the inverting configuration. The different modulation formats (i.e. 3-PRS, 9-QPRS and 49-

QPRS) have different peak amplitudes, and must therefore be scaled differently. The 3-PRS, 9-QPRS and 49-QPRS signals are scaled using separate potentiometers. The level-shifting is achieved by connecting the -15V supply to the inverting input of the op-amp via a 100k pot and a 100k resistor. Adjusting the pot sets the op-amp output d.c. bias to +2.5V.

The input to the ADC0820 is thus a signal in the range $0 < Vin < 5V$. The digital output to the DSP56001 is 8-bit data that can be sampled at 19.2kHz.

RECEIVER LOWPASS FILTER OUTPUT DAC

The final result of the receiver's signal processing operations is a baseband symbol sample value. It is desirable to view the pre-detection lowpass filter output to assess the quality of the recovered signal. The receiver therefore requires a digital-to-analogue converter for this purpose.

The DAC chosen for this function is the AD558. As in the case of the transmitter DAC, the AD558 is a memory-mapped write-only peripheral, occupying the memory location Y:$8000. The DAC has 8-bit resolution, and the settling time for a full-scale step is 800nsec, making it suitable for the 19.2kHz sampling rate. The chip is permanently enabled as its inputs come from a latch that connects it to the DSP56001 data bus. The AD558 is configured to give an analogue output range of $0 < Vout < 2.56V$. This is shown in Figure A.6.

6.2.4 WIDEBAND NOISE GENERATOR

A wideband noise generating circuit was constructed for the purpose of error-rate testing at the receiver. The circuit diagram for this is shown in Figure A.7. The noise source is derived from reverse-biasing the base-emitter junction of a BC238 transistor into breakdown mode. The resulting current

through the narrow PN junction produces a noise voltage. The noise is both thermal noise, due to the base spreading resistance, and shot noise due to the quantized nature of the current through the intrinsic base resistance. In theory both produce white Gaussian noise.

The noise voltage at the BC238 base-emitter junction is amplified by a transistor stage and the use of a wideband LM6361 op-amp. The output of this op-amp is fed to another LM6361 in inverting mode. The received signal from the transmitter is also fed to the inverting input of the second LM6361, configured to give the signal a gain of -1. The noise input has a potentiometer adjustment to give further noise gain. The circuit output is therefore the signal plus adjustable variance near-white Gaussian noise. Figure 6.4 shows the frequency spectrum of the generated signal over the range 0 - 10kHz.

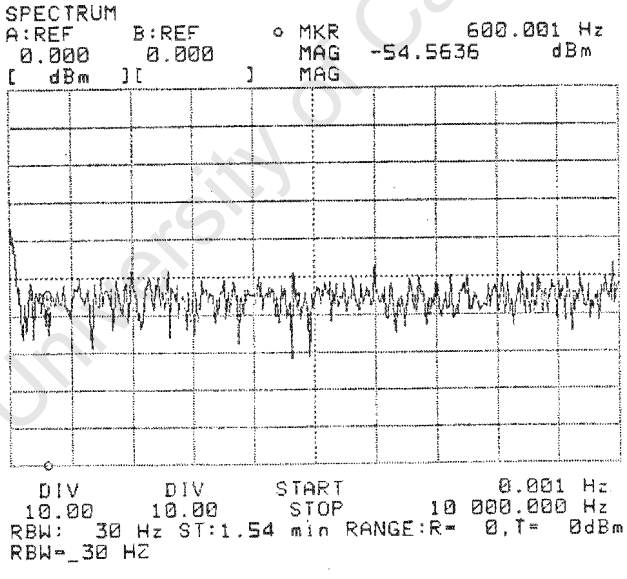


Figure 6.4 - Noise generator spectrum 0 - 10kHz.

The output of the first LM6361 is not truly white Gaussian noise. This is due to two factors:-

1. Noise Bandwidth

The noise is obviously not of infinite bandwidth. This is not problematic since the receiver must only see a flat spectrum in the receiver noise bandwidth. The receiver noise bandwidth for a modulated spectrum is the double-sided Nyquist bandwidth for a root raised-cosine filter [6.1], which coincides to the frequency range 600 - 3000Hz.

2. Noise Crest Factor

In a truly Gaussian noise source, there is a finite probability of occurrence of infinite amplitude voltages. The noise source of Figure A.6, however, limits the peak noise excursions by clipping at the power rails. Limiting is also caused by the slew rate limitations of the op-amp. The noise source thus said to have a finite crest factor. The crest factor of a waveform is defined as the ratio of the peak signal value to the RMS signal value. As an example, for the circuit above, assuming an RMS voltage of 1V and a peak voltage of 14V, the crest factor would be 14 = 22dB.

6.3 REAL-TIME SOFTWARE

6.3.1 INTRODUCTION

The receiver software consists of an assembler program for the DSP56001 and the host Pascal program that is used for both the transmitter and the receiver. The Pascal program has been covered in section 5.3.2, but the receiver-oriented functions will be explained again. The assembler program performs the signal processing functions of the receiver, which include sampling the received signal, demodulating the I and Q channels down to baseband, lowpass filtering, thresholding, and decoding.

6.3.2 THE HOST PASCAL PROGRAM - 'TRANSMIT.PAS'

The relevant points of the program are described again here, as they relate to the assembler program description in the following section.

The receiver lookup table is half the length of the transmitter lookup table, i.e. 176 points. The bit rate is selectable from the computer keyboard. The receiver contains a data comparison buffer for the bits from the PRBS generator. This is used for counting errors, and its operation is described in the next section. The length of this buffer is also selectable from the keyboard.

6.3.3 THE DSP56001 ASSEMBLER PROGRAM - 'RECEIVE.ASM'

INTRODUCTION

The functions implemented by the receiver software in the DSP56001 are shown in block diagram form in Figure 6.5.

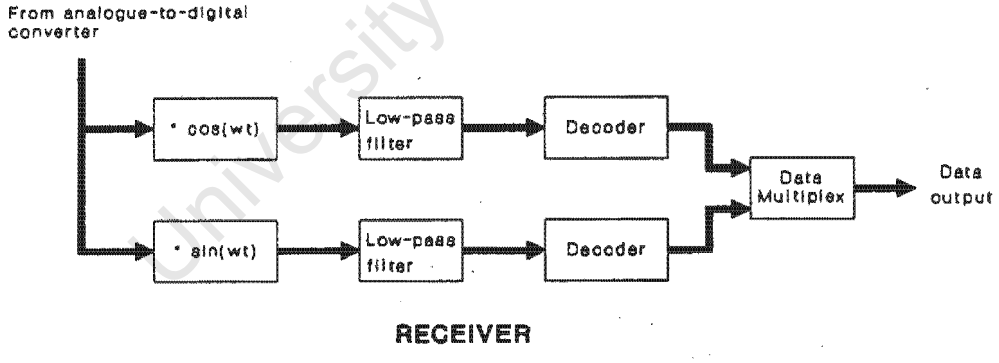


Figure 6.5 - Receiver functions implemented by the DSP56001.

The DSP56001 is interrupt-driven, i.e. it performs the signal processing tasks when it receives the /IRQA and /IRQB interrupt signals. The /IRQA interrupt occurs at the sampling rate of 19.2kHz, and causes sampling, demodulation and decoded bit output. The /IRQB interrupt occurs at 2.4kHz, and causes baseband symbol calculation and decoding. The interrupt /IRQA has higher priority than /IRQB as it

often occurs while /IRQB is being serviced, and must be serviced immediately. Figure 6.6 a) and b) show the receiver assembler program structure in flowchart form.

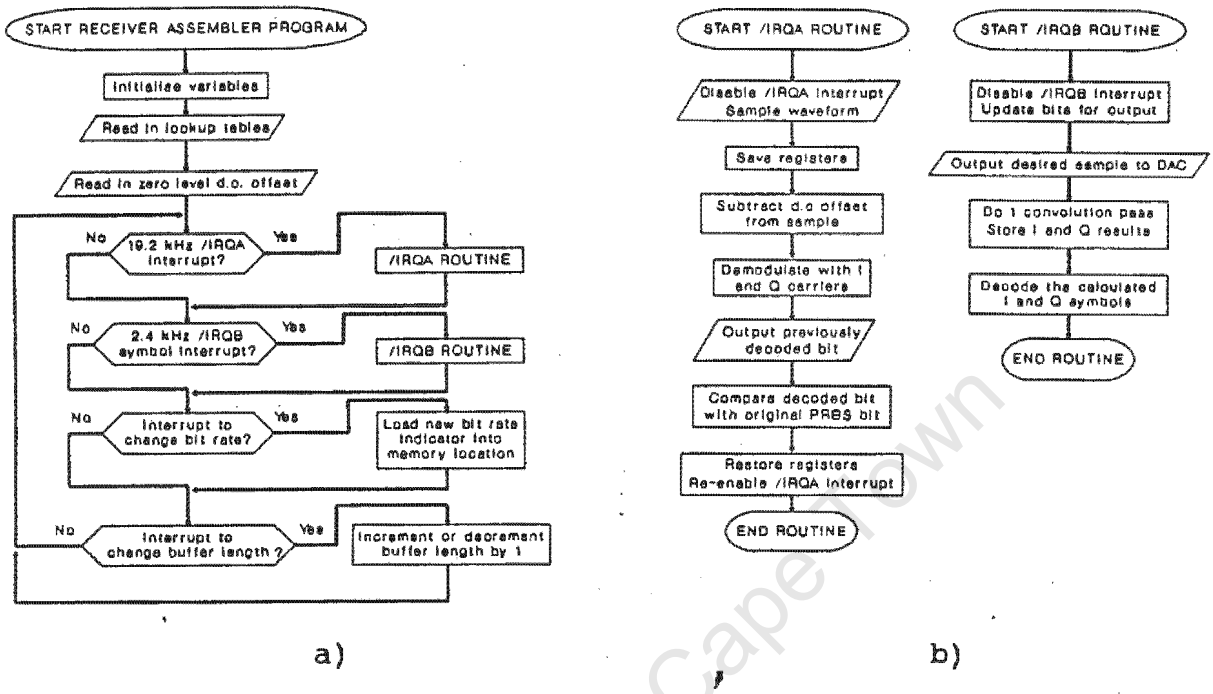


Figure 6.6 - Receiver DSP56001 assembler program flowcharts.
a) - Flowchart for main body.
b) - Flowcharts for interrupt service routines.

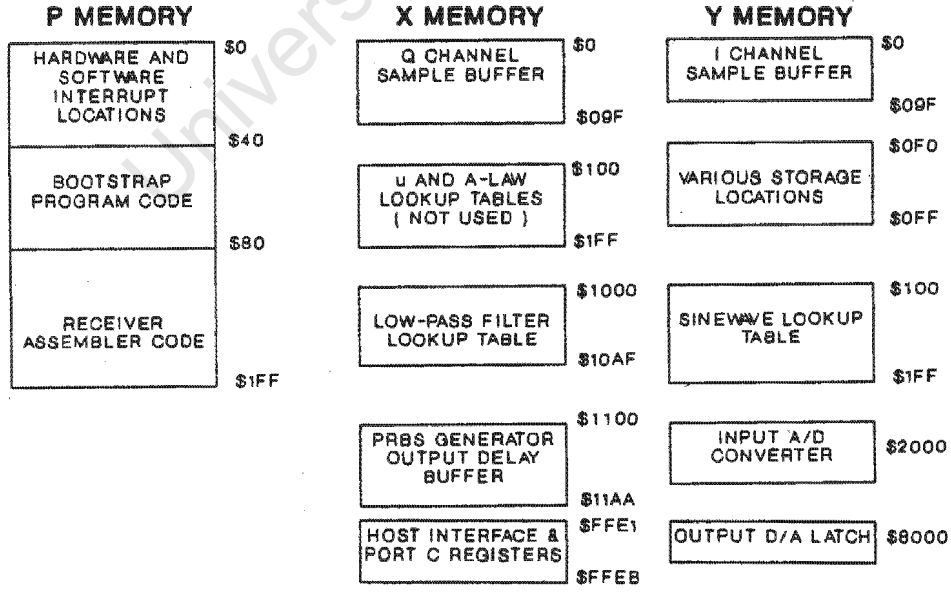


Figure 6.7 - Receiver DSP56001 memory map.

The memory map for the receiver DSP56001 is shown in Figure 6.7. This clarifies the buffer and peripheral storage locations of the receiver. The program structure is now described with reference to the flowcharts given in Figure 6.6. The description will be in two sections - one corresponding to Figure 6.6 a) and one corresponding to Figure 6.6 b).

RECEIVER ASSEMBLER MAIN PROGRAM OPERATION

1. Initialize variables.

Port C is configured as general purpose I/O. The address registers are used as follows:-

r0 - pointer for the PRBS delay buffer for error checking.
r1 - pointer for the filter lookup table.
r2 - pointer for the Q channel demodulated sample buffer for use in the convolution routine.
r3 - pointer for the Q channel demodulated sample buffer for reading and storage routine.
r4 - I channel carrier pointer.
r5 & r6 - same as r2 and r3, but for the I channel buffer.
r7 - Q channel carrier pointer.

The final step in initialization is to enable the interrupts. This is done by writing an appropriate value to the Interrupt Priority Register.

2. Read in lookup table.

The root raised-cosine lookup table is read into the DSP56001 from the Host program in the routine 'table'. In the program shown in Appendix G the table has 176 taps - half that of the transmitter. The table is stored in X:$1000 - X:$10AF.

This routine also clears the I and Q channel buffers in X and Y memory. The buffers are the same length as the lookup table, as convolution here requires point-by-point multiplication.

3. Read in zero-level d.c. offset from input ADC.

The analogue-to-digital converter at the receiver input has an input voltage range 0 - 5V. A transmitter zero-level therefore results in approximately 2.5V at the ADC input, which in turn results in half the full-scale 8-bit digital output of the ADC being read by the receiver. This zero level must be subtracted from every sample that is read from the ADC. This removes the d.c. bias from the signal and allows correct demodulation of the received waveform. The zero-level is read in while the transmitter is in its zero-transmit loop.

Reading data from the ADC is done in four steps. Firstly the ADC /CS (chip select) is enabled by reading from Y:$2000. Secondly the /RD (read) signal is then activated by clearing Port C bit #0. Thirdly a delay is introduced while the ADC performs the conversion. Finally the actual ADC output is read in to the data bus by again reading from Y:$2000.

4. Is there a 19.2kHz /IRQA interrupt pending?

The /IRQA interrupt service routine causes signal sampling, and outputs the decoded bit stream. This is explained further on.

5. Is there a 2.4kHz /IRQB interrupt pending?

The /IRQB interrupt service routine causes lowpass filtering of the demodulated signal to calculate the received symbol value. The symbols occur at 2.4kHz. The symbols are then decoded to recover the bit stream. This is also explained further on.

6. If necessary, change the bit rate.

The system bit rate is changed under control of the host Pascal program. To change the bit rate the host vectors an interrupt to the relevant program location in the DSP56001. The interrupt service routines for these three interrupt locations are called 'low', 'mid' and 'high', and they simply write a value to the location Y:bitrate indicating the current bit rate.

7. If necessary, change the PRBS buffer length.

The data from the PRBS generator is stored in a delay buffer for comparison with the decoded bit stream. This allows error checking of the decoded bit stream. The correct delay length is variable, due to start-up synchronizing, and therefore it must be selectable. The interrupt service routine 'shftu' increments the buffer length by 1, and the routine 'shftd' decrements the length of the buffer by 1.

/IRQA INTERRUPT SERVICE ROUTINE OPERATION

The flowchart for the /IRQA interrupt service routine is shown in Figure 6.6 b). This interrupt occurs at the sampling rate of 19.2kHz, with the 19.2kHz clock being provided by the transmitter. The routine is called 'readin' in the program listing. The service routine does the following:-

1. Disable /IRQA interrupt and sample the received waveform.

The /IRQA interrupt is disabled to avoid multiple interrupts from a noisy interrupt source. The /IRQB is still enabled as it has lower priority than /IRQA and will not interrupt during the /IRQA service routine. The received signal is sampled in the beginning of the routine by reading from the ADC.

2. Save registers.

The /IRQA interrupt often occurs while the /IRQB interrupt is being serviced. It has higher priority than the /IRQB interrupt, so it is serviced while the /IRQB routine is being executed. At the beginning of the /IRQA service routine, the values of the registers used in the routine must be saved, and at the end of the routine they must be restored for the continuation of the /IRQB service routine. The register values are stored in a section of on-chip Y memory.

3. Subtract d.c. offset from signal sample.

The sample read in from the ADC has a d.c. offset component, which must be subtracted from all the samples read in from the ADC. This allows correct demodulation.

4. Demodulate the sample with I and Q carriers.

The a.c. sample value contains both the I and the Q channel information. Multiplying the value by the corresponding coherent 1800Hz I and Q values will correctly demodulate the value into its I and Q baseband plus higher frequency values.

The address register r4 points to the I value in the sinewave lookup table, and register r7 points to the Q value. The r7 register pointer is 90° shifted from the r4 register to maintain correct quadrature demodulation. The lookup table is 256 points long, so 90° corresponds to an offset of 64 locations.

The results of the demodulation are stored in the I and Q sample buffers in Y and X memory respectively. These buffers are 176 locations long, the same length as the lowpass filter lookup table.

5. Output previously decoded bit.

The decoded bits (decoded in the /IRQB service routine) are stored in location Y:bitsout, which contains an 8-bit value. The format of this value depends on the bit rate. The receiver sampling rate is 19.2kHz, so if the data rate is 9600 bps, a decoded bit must be output once every two interrupts. Alternatively, the 9600 bps data rate can be achieved by outputting the same bit for two 19.2kHz interrupts, then outputting the next bit twice in the next two interrupts, etc. As an example, the bit sequence 00110011 transmitted at 19200 bps is the same as the sequence 0101 transmitted at 9600 bps. Thus if the operational bit rate is 9600, any decoded bit will be stored as two duplicated bits and output at 19.2kHz. The 8-bit location Y:bitsout thus contains 4 decoded bits. These are output bit-by-bit, and a new value loaded after every 4 decoded bits.

Similarly, for 4800 bps transmission, Y:bitsout contains 2 decoded bits, each represented by 4 duplicated bits. For 2400 bps transmission, Y:bitsout contains 1 decoded bit, represented by 8 duplicated bits. Thus outputting the bits of Y:bitsout at a constant rate of 19.2kHz will produce the right bit rate depending on how the bits are stored.

6. Compare decoded bit with original PRBS bit

By comparing the decoded bit stream to a sufficiently delayed version of the original bit stream the errors in the system can be counted. The original bit stream is produced by the PRBS generator, so reading the data from the PRBS source and storing it in a circular buffer in the receiver produces a delay which allows data comparison with the decoded bit stream.

The transmitted PRBS data bit is read and stored in the delay buffer in external X memory, pointed to by address register r0. After storing a new bit in the buffer, register r0 points to the last element in the circular delay buffer. The bit value at this location is output via Port C pin #4, allowing the delayed version of the PRBS sequence to be viewed with the decoded bit sequence on a logic analyzer or oscilloscope. The decoded bit and the last bit in the buffer are compared, and if they differ a pulse is transmitted on Port C pin #6 by setting pin #6 and then clearing it a short while later. When doing error counts, a frequency counter is connected to this pin and the pulse count is accumulated.

It should be noted that in the case of 2400 bps transmission, eight error pulses will occur for every data bit that is in error, due to Y:bitsout containing eight replicated bits. For 4800 bps transmission, four error pulses will occur for every data bit error, and for 9600 bps two error pulses will occur for every data bit that is in error. The final error count therefore had to be divided by the right value to give the correct number.

7. Restore registers and re-enable /IRQA.

The register values that were saved at the beginning of the service routine are restored at the end of the routine, and the /IRQA interrupt is re-enabled by writing an appropriate value to the Interrupt Priority Register.

/IRQB INTERRUPT SERVICE ROUTINE OPERATION

This interrupt service routine, called 'calc', performs lowpass filtering on the demodulated I and Q samples in the sample buffers. It also decodes the filtered baseband symbol values into a bit stream. The transmitter lowpass filter and the receiver lowpass filter form a matched filter pair, each having the square-root response of the overall desired channel response. To perform lowpass filtering on the

received signal requires convolution of the received signal samples with the filter lookup table.

If a complete continuous baseband output was required at the receiver output, a full 176-point convolution would have to be performed on both the I and the Q channel buffers every time a sample interrupt occurs. This is not possible since the signal is sampled at 19.2kHz and there is insufficient processor time for the two convolutions. The only baseband points of interest, however, are at the symbol sample instants, which contain the transmitted data information. If the baseband signal eye pattern is viewed on an oscilloscope, these symbol instants would be the points in the centre of the eyes. The symbols occur at 2400Hz, so a convolution for both channels only has to be performed once every 1/2400 seconds. This convolution must occur when the required symbol sample is in the location in the buffer that corresponds to the centre sample of the lookup table. This is the requirement for minimum ISI. The transmitter provides this /IRQB clock signal with the correct phase. The service routine does the following:-

1. Disable /IRQB interrupt and update bits for output.

The /IRQB interrupt is disabled to prevent multiple interrupts occurring due to a noisy signal. The /IRQA interrupt is disabled only while the baseband symbol value (3 or 7 levels) is written to the receiver DAC.

The location Y:newbits contains the decoded bits from the previous symbol calculation in the service routine 'calc'. At the beginning of the current service routine the value in Y:newbits is transferred to Y:bitsout. The bits in Y:bitsout are then transmitted in the /IRQA routine, as described above.

2. Output desired sample to DAC.

The result of the previous I channel lowpass filtering (or convolution) in the receiver is stored in Y:oldvalI, and is written to the receiver DAC where it can be observed on an oscilloscope. This allows visual monitoring of the action of the receiver.

3. Do 1 convolution pass and store the I and Q channel results.

This is the section of the /IRQB routine that executes the lowpass filtering of the demodulated received signal. The received signal is centred on the 1800Hz carrier, so demodulation results in signal replicas at baseband and at 3600Hz. The upper frequency replica at 3600Hz is removed by lowpass filtering the demodulated signal. This also filters out other unwanted high frequency components (>1200Hz) that may have arisen at the receiver input.

The lowpass filtering process involves convolving the filter lookup table with the I and Q sample buffers. This occurs when a symbol sample is at the location in the I and Q buffers that corresponds to the peak value in the filter lookup table. The convolution is achieved by point-by-point multiplication of the lookup table values with the sample buffer values. Ideally the resultant output should be one of three levels for 2400 bps and 4800 bps transmission, or one of seven levels in 9600 bps transmission (for a noisy signal this is not true).

There are some considerations in the convolution process. Firstly, the convolution loop could get interrupted by the /IRQA interrupt while it is still in progress. The /IRQA routine loads new samples into the I and Q sample buffers, thus overwriting the oldest samples in the buffers. This means that the convolution must multiply the oldest values in the buffers first, to avoid calculation errors.

An example of convolution using a lookup table 6 symbols long with 4 points per symbol is shown in Figure 6.8. With reference to the diagram, the newest samples are written into the buffers by overwriting the oldest samples. Thus the I and Q buffers will have, for example, the newest value in the left-most buffer location. The next location to the right stores the oldest value, the third location stores the second oldest value etc., all the way across to the right-most buffer location, which stores the second most recent value. The samples in the buffer are multiplied by the corresponding values in the lowpass filter lookup table, and the products are summed to give the baseband symbol value. Note that in this example a convolution is performed only once for every 4 samples that are read in. A convolution is only performed when a sampling instant coincides with the peak value of the lookup table. In the actual receiver, a convolution is performed once every 8 samples.

The method of multiplying the oldest buffer value by the second value in the lookup table and not the last is justified, since the last lookup table value is the same as the second, the second-last is the same as the third, etc. The result of the convolution is the baseband I and Q symbol values.

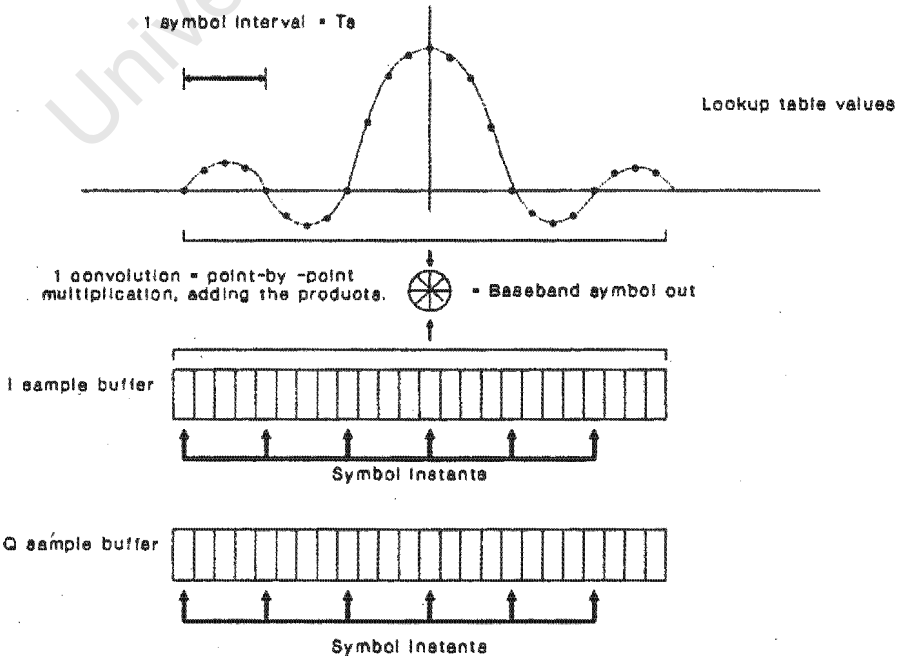


Figure 6.8 - Receiver convolution pass.

4. Decode the calculated baseband I and Q symbols.

The baseband symbols contain the received data information, which is dependent on the symbol value. The symbols must be thresholded to decide which symbol level has been received. The simplest form of thresholding is to place the decision thresholds midway between the ideal received symbol levels. This is not optimum since it takes no account of the probability of occurrence of the signal levels. Implementing optimum thresholds is difficult since it requires adaptive monitoring of the noise variance. This is out of the scope of the project.

The method for determining the actual threshold values is very simple. The receiver output in noise-free operation, which was a 3-level or 7-level signal (depending on the modulation format), is taken from the DAC output and viewed on the oscilloscope. The full-scale DAC output is 2.56V, so the 8-bit value corresponding to the baseband symbol levels can be easily calculated from the ratio $V_{out}/2.56V$. The threshold values are then calculated as half the distance between the signal levels. Table 6.1 shows the decision thresholds obtained using this method.

	THRESHOLD LEVELS			OUTPUT BITS	
SYMBOL	49-QPRS	9-QPRS	3-PRS	3-LEVEL	7-LEVEL
3	$52	$2A	$31	0	10
2	$31				01
1	$10				00
0	$FFFFF0			1	11
-1	$FFFFCF			0	10
-2	$FFFFAE				01
-3					00

Table 6.1 - Decision thresholds for receiver symbol detection.

The thresholds are in two's complement notation. Note that for 9-QPRS and 3-PRS there is only one threshold value. This is because the two outer levels (-1 and 1) in the 3-level signal contain exactly the same information, so taking the absolute value of the signal results in a binary (0 and 1) value and allows the use of only one threshold.

Depending on the bit rate indicator value stored in Y:bitrate, one of the routines 'l_out', 'm_out' or 'h_out' is used to threshold the baseband symbol. The routines 'l_out' and m_out' are for 2400 and 4800 bps respectively, and check if the symbol value is above or below the threshold, and set the decoded bit values accordingly. The routine 'h_out' thresholds and decodes the 9600 bps symbols. This is done by starting at the highest threshold and checking if the symbol value is larger than the threshold. If not then the symbol value is checked against the next lowest threshold, etc. If a threshold is found where the symbol value is larger than the threshold, the output bits are set accordingly and the checking process is discontinued. If not then the lowest level has been received, and the bits are then set to the corresponding value. This is performed for both the I and the Q values.

At the end of the routine, the I and Q decoded bits are combined into one 8-bit word and stored in Y:newbits. The I channel bits occupy the least-significant 4 bits, and the Q channel occupies the most-significant 4 bits. The /IRQB interrupt is re-enabled and program flow returns to the routine 'loop'.

6.4 SUMMARY

The receiver, like the transmitter, consists of both hardware and software. The hardware converts the received analogue signal into digital format for the receiver DSP56001. The DSP56001 performs the demodulation of this signal to produce baseband symbols. These are written to a DAC for viewing on an oscilloscope. The software consists of the same host Pascal program as the transmitter, and an assembler program for the DSP56001 microprocessor. Receiver synchronization signals are provided by the transmitter. The receiver design incorporates hardware and software to enable the running of bit error-rate tests.

Figure 6.9 shows the transmitter and receiver hardware constructed for this project. Figure 6.10 shows the complete experimental setup for the transmitter and receiver.

Figure 6.9. - Transmitter and receiver hardware.

Figure 6.10 - Modem transmitter and receiver experimental setup.

6.5 REFERENCES

[6.1] K. Feher & Engineers of Hewlett-Packard, Telecommunications Measurements, Analysis and Instrumentation, Prentice-Hall, Englewood Cliffs, New Jersey, 1987, pg.200.

CHAPTER 7. EXPERIMENTAL RESULTS

7.1 INTRODUCTION

This chapter describes measurements that have been taken at various points in the modem design to quantify the system's performance. For completeness it is reiterated here that the term 'modem' in this context refers to a data modulator and demodulator structure only, and not to a functional device operating over telephone lines.

The chapter focuses on three aspects of system performance. These are the following:-

1. The eye pattern qualities.

2. The signal spectral qualities.

3. The system bit error rate versus signal-to-noise ratio performance.

The transmitter results are discussed first, and include measurements of the transmitted eye patterns and transmitted signal spectra corresponding to four different lowpass root raised-cosine filters. The receiver results follow, and include measurements of the demodulated and filtered baseband eye pattern and the corresponding spectrum. This is followed by derivations and measurements of the bit error-rate performance of the system in the presence of AWGN. Each section ends with a discussion of the results.

There are certain system parameters and specifications that must be summarized before describing the actual results. These serve to define the constraints of the measurements that were taken. The system parameters are:-

- The transmitter sampling rate is 38.4kHz.
- The receiver sampling rate is 19.2kHz.

- The receiver synchronization signals are supplied directly by the transmitter, i.e. the clock recovery at the receiver is bypassed.
- The transmitter and receiver are connected directly, the communication path being a short length of cable. The 'channel' therefore has negligible effect.

7.2 TRANSMITTER RESULTS

The following measurements were taken at the transmitter output:-

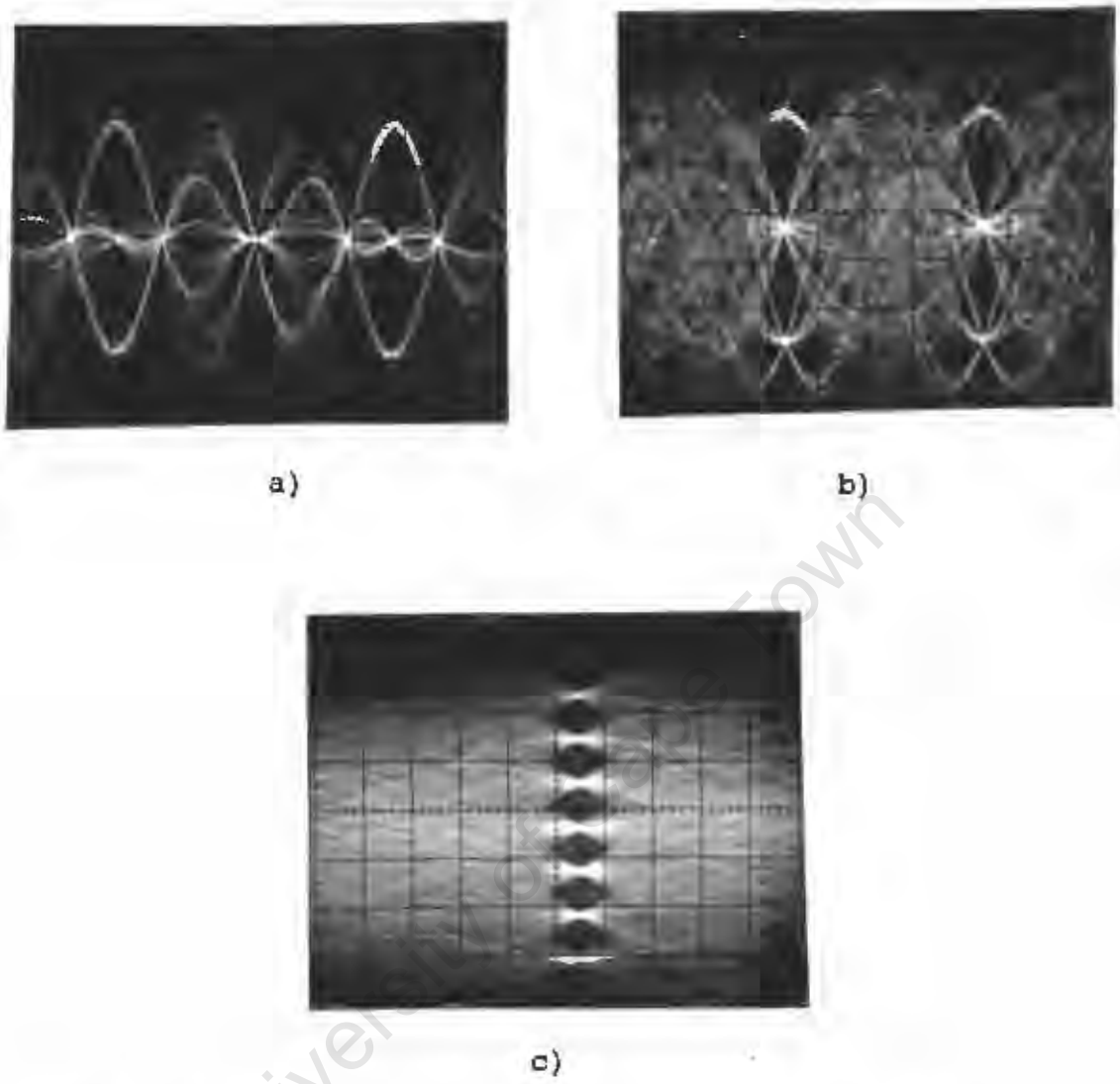
1. Measurements of the transmitted eye pattern for the three operational bit rates of 2400, 4800 and 9600 bps.

2. Measurements of the transmitted signal spectrum for four values of raised-cosine filter roll-off factor α.

These readings are more illustrative than quantitative, since they do not give a measure of the overall system performance. They do, however, show that the transmitted signal conforms to the required specifications in terms of spectral occupancy. They also demonstrate the accuracy that can be achieved by using a digital architecture.

7.2.1 TRANSMITTED EYE PATTERNS

The transmitted signal is a modulated, bandlimited waveform that takes on the format of either 3-PRS, 9-QPRS or 49-QPRS. These three modulation schemes correspond to data rates of 2400, 4800 and 9600 bps respectively. Figures 7.1 a), b) and c) show photographs of the transmitted eye patterns for 2400 bps, 4800 bps and 9600 bps. These patterns are generated using a 352 point root raised-cosine filter with $\alpha = 0.1$.



a) b)

c)

Figure 7.1 - Eye patterns for the transmitted signal.
a) - 2400 bps transmission using 3-PRS.
b) - 4800 bps transmission using 9-QPRS.
c) - 9600 bps transmission using 49-QPRS.

Figure 7.1 a) illustrates the relationship between the symbol instants and the carrier frequency. This signal only has an I component, so the carrier envelope is easily seen. This photograph shows two eye openings very clearly. The carrier frequency is 1800Hz and the symbol rate is 2400 baud (irrespective of the modulation scheme), so the baseband symbol instants occur once every 1800/2400 = 0.75 carrier cycles. The photograph thus shows a total of three symbol

instants - the third symbol instant is not seen since it appears at the carrier zero-crossing in the centre of the oscilloscope screen. The generated signals are very clean - a result of using digital signal processing methods.

The carrier envelope is not as easily seen in Figures 7.1 b) and c), since the signal in both cases is a composite I and Q channel signal. b) shows two eye locations, with one eye corresponding to the I channel and the other eye corresponding to the Q channel. The eyes of one channel are superimposed on the carrier zero-crossing eyes of the other channel, as described in the previous paragraph. The slight amount of ISI that is visible is not a result of signal degradation. It arises from the use of the root raised-cosine filter in the transmitter, which does not comply to Nyquist's second theorem for zero ISI. The cascaded transmitter/receiver filter pair does, however, comply with the zero ISI condition.

To illustrate the zero ISI introduced by the raised-cosine filter, the transmitter software was modified to transmit the I channel baseband signal for 2400 bps using a 352 point, 22 symbol wide, α = 0.1 raised-cosine lowpass filter. The resulting eye pattern is shown in Figure 7.2. There is negligible ISI at the symbol sampling instants.

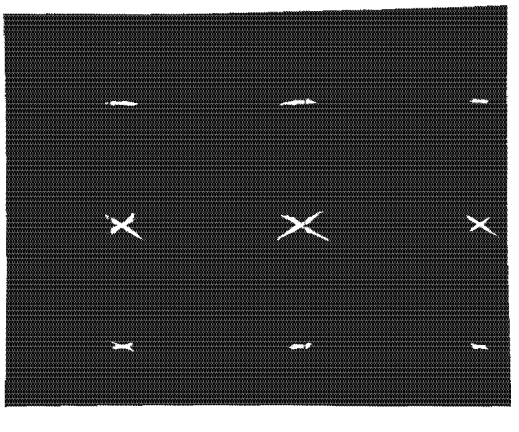
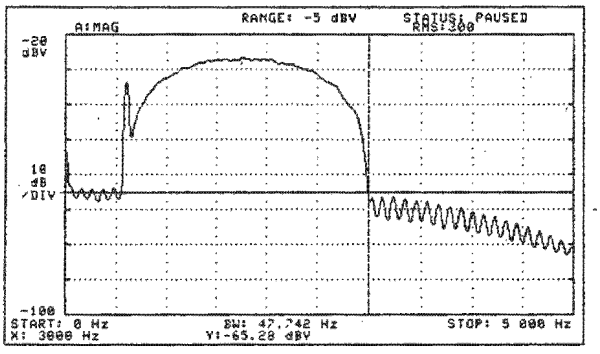


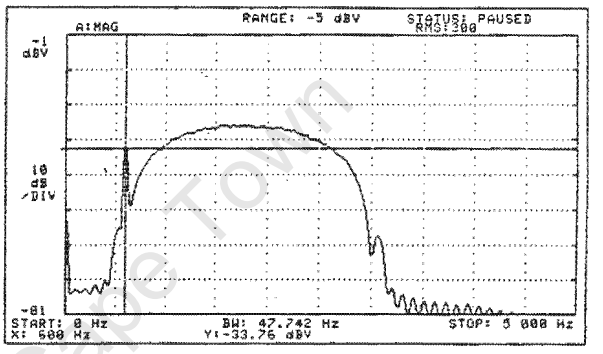
Figure 7.2 - Baseband eye pattern for α = 0.1 raised-cosine filter.

7.2.2 TRANSMITTED SIGNAL SPECTRA

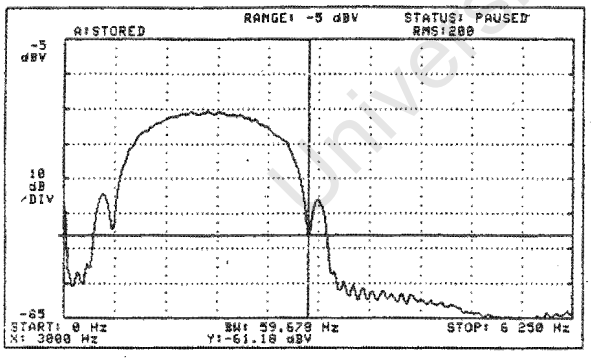
The transmitted spectrum is shown here for four filter characteristics. In each case the root raised-cosine filter is 352 points long, corresponding to a width of 22 symbols. Figure 7.3 a) shows the transmitted spectrum for $\alpha = 0.03$, b) $\alpha = 0.1$, c) $\alpha = 0.2$ and d) $\alpha = 0.3$. All signals except c) have a pilot tone inserted in the spectral null at 600Hz.



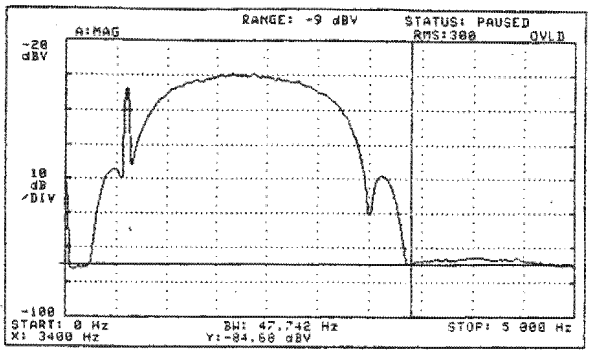
a)



b)



c)



d)

Figure 7.3 - Transmitted signal spectra.

a) $\alpha = 0.03$

b) $\alpha = 0.1$

c) $\alpha = 0.2$

d) $\alpha = 0.3$.

These filters are seen to have a vastly improved transfer characteristic over those given in the transmitter simulation in section 5.1.4. This is due to the truncated nature of the FFT used in the simulations. Again it is seen that a higher value of α leads to a wider mainlobe, but better suppression of the sidelobes. For α = 0.03, the out-of-band at 3100Hz is nearly 40dB. For α = 0.3, the out-of-band suppression at 3500Hz is seen to be approximately 55dB. This improved suppression is at the expense of a wider mainlobe. The case of α = 0.1 shows a good compromise. The mainlobe occupies the region 400Hz to 3150Hz. This is not critical since the modem application is for use over leased 4-wire M.1020 lines for full-duplex operation, and thus there are no adjacent signals to present interference problems. The out-of-band suppression for α = 0.1 is approximately 48dB. The α = 0.1 filter was therefore chosen as the lowpass filter to use in the bit error-rate measurements. It is noted that the usable bandwidth in a dedicated M.1020 line might be greater than the region 400Hz to 3150Hz. Using a value of α greater than 0.1 , e.g. α = 0.3, might be possible.

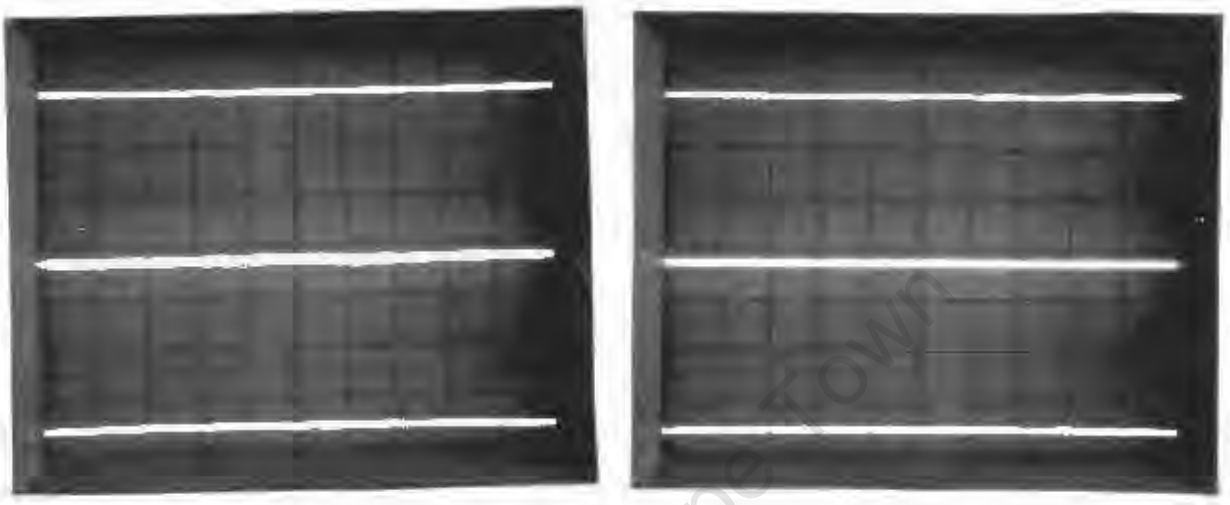
7.3 RECEIVER RESULTS

7.3.1 RECEIVER TIME DOMAIN OUTPUT

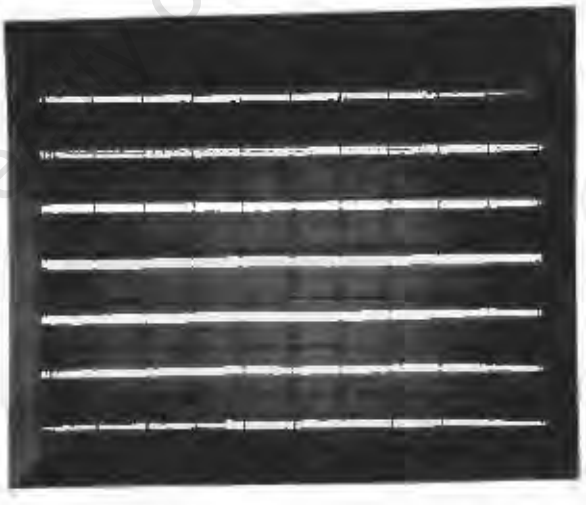
The receiver samples the transmitted signal at 19.2kHz, and outputs baseband symbol samples at 2400Hz. These received baseband symbols are the result of demodulating and lowpass filtering the received signal at the symbol instants. The symbol values are written to the receiver DAC for display on an oscilloscope.

Figures 7.4 a), b) and c) are photographs of the recovered I channel baseband samples at the symbol instants for 2400 bps, 4800 bps and 9600 bps respectively. These correspond to a receive root raised-cosine filter of 176 points (22 symbols wide) with α = 0.1. This is the same characteristic

as the transmitter filter, but the sampling frequency (and therefore the points/symbol ratio) is halved. Figure 7.4 illustrates the receiver output for the case of no added noise. The displayed signal is the pre-detection signal, i.e. it is taken before the thresholding and decoding steps in the receiver.



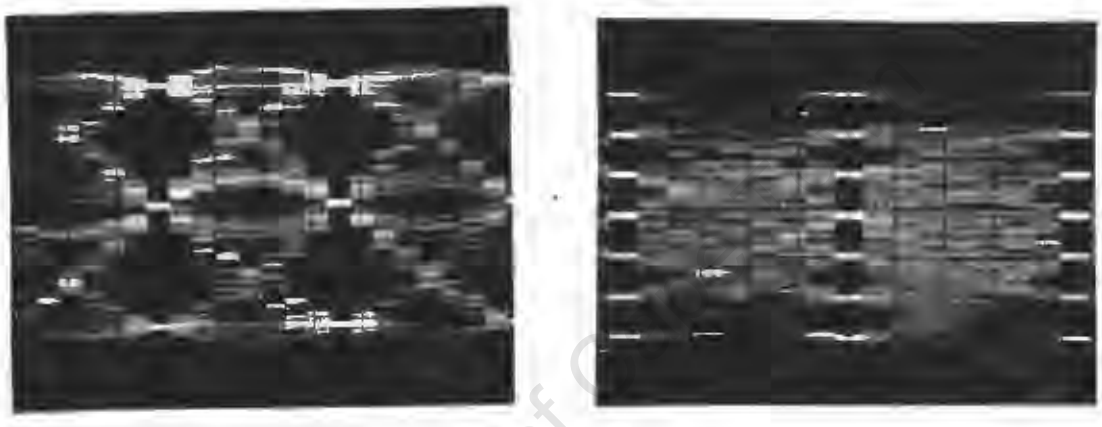
a) b)



c)

Figure 7.4 - Noiseless receiver symbol-instant outputs after lowpass filtering.
a) - 2400 bps.
b) - 4800 bps.
c) - 9600 bps.

The receiver DSP56001 software was modified to output the demodulated and lowpass filtered (baseband) I channel signal to the receiver DAC. This required a very minor program alteration. Figures 7.5 a) and b) are photographs of the recovered baseband signal for 2400 bps and 9600 bps transmission respectively. The quantized nature of the signals is due to connecting the oscilloscope directly to the DAC output, i.e. no smoothing filter is used at the receiver.



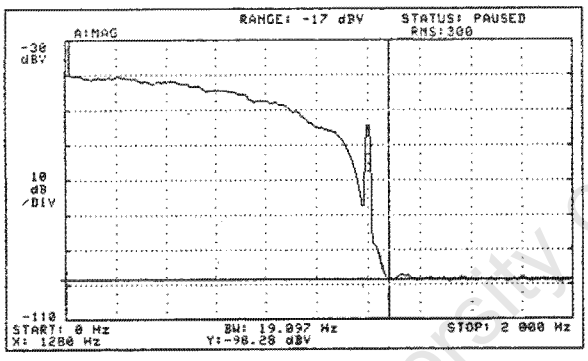
a) b)

Figure 7.5 - Recovered baseband signals at receiver.
a) - 2400 bps.
b) - 9600 bps.

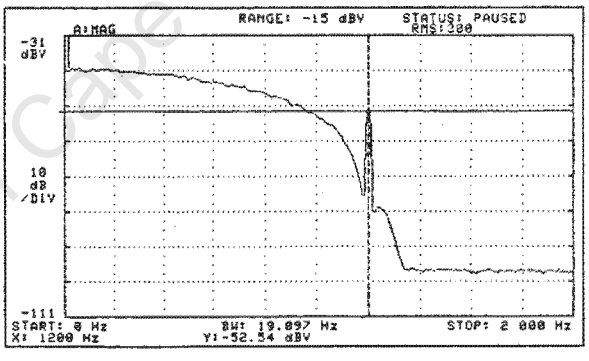
The oscilloscope traces shown in Figure 7.4 are a result of outputting only the symbol samples that appear in the centre of the eyes in Figure 7.5. It is noted that the eye width for the case of 9600 bps transmission is much smaller than that in the case of 2400 bps transmission. A higher bit rate therefore requires more accurate sampling at the symbol instants than a lower bit rate. As before, it is noted that digital methods have allowed a precise signal to be achieved.

7.3.2 RECEIVER RECOVERED BASEBAND SPECTRA

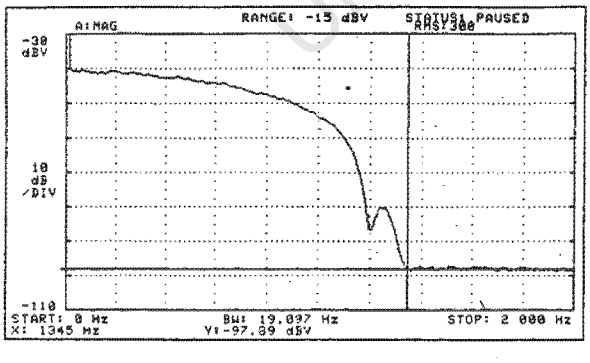
The signals shown in Figure 7.5 in section 7.3.1 are taken at the output of the receiver DAC. These signals are a sampled version of the recovered baseband signal, with the DAC applying a zero-order hold effect to the samples. The zero-order hold operation shapes the sampled signal spectrum by a sin(x)/x shape. The null in the sin(x)/x shape is at 19.2kHz (the sampling frequency), which is sufficiently far from the baseband spectrum to cause spectral distortion. The measured baseband spectrum can therefore be assumed to closely approximate the true recovered baseband signal spectrum. The receive filter is again a root raised-cosine response with 176 points.



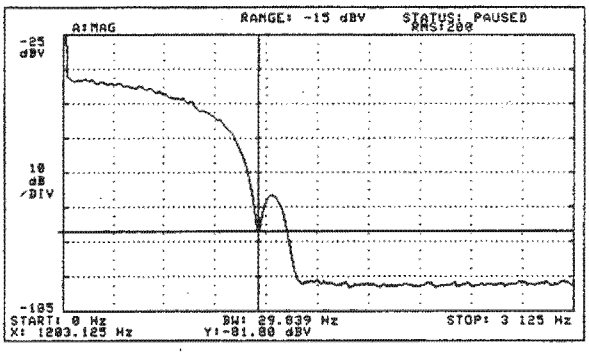
a)



b)



c)



d)

Figure 7.6 - Recovered baseband signal spectra.

a) α = 0.03, pilot tone.
b) α = 0.1, pilot tone.
c) α = 0.1, no pilot tone.
d) α = 0.2, no pilot tone.

Figure 7.6 shows the recovered baseband spectra for a) α = 0.03, pilot tone added, b) α = 0.1, pilot tone added, c) α = 0.1, no pilot tone and d) α = 0.2, no pilot tone.

It is seen in Figure 7.6 for α = 0.1 that the out-of-band frequencies above 1345Hz are approximately 58dB below the peak spectral power. The α = 0.1 value causes minor spectral widening, shown by the width of the sidelobe above 1200Hz. This is very good performance (albeit for a nearly ideal situation), and the spectrum can be compared to results obtained in [7.1], where filters with α = 1 are used.

7.4 BIT ERROR RATE MEASUREMENTS

7.4.1 INTRODUCTION

As mentioned in section 3.4.1, the probability of error versus signal-to-noise characteristic gives a measure of how well a system performs in the presence of noise. Some texts, for example [7.2], show plots of the theoretical probability of symbol error (P_e) versus E_b/N_0 for various M-ary modulation schemes. The term E_b is defined as the average energy of a modulated bit, and N_0 is the average noise spectral density at the input of the receiver. Other texts show plots of symbol P_e versus SNR, where the average signal and noise power are measured at the output of the receiver lowpass filter. An example of this is given by Feher [7.3].

The tests performed in this project, however, measured the *bit* error rate of the system. This makes comparison of the measured results with theory difficult. For this reason, a derivation is presented here to calculate the bit P_e versus SNR for 3-level and 7-level class 1 partial response signalling. Here the signal-to-noise ratio is specified as the ratio of the average signal power to the average noise power at the output of the lowpass filter. The measured results can therefore be compared to this characteristic to evaluate the practical performance of the system.

7.4.2 PROBABILITY OF ERROR IN DUOBINARY SYSTEMS

In the following derivations, the possible effects of intersymbol interference due to channel phase distortion are ignored, and it is assumed that additive white Gaussian noise (AWGN) is present.

3-PRS error performance

The derivation follows the lines of a derivation given by Peebles [7.4]. In modulated duobinary transmission there are three possible signal states, so it is assumed that polar transmission is employed with possible receiver signal output states **-A**,0,+**A**. The probabilities of occurrence of the outer levels (+/-**A**) are $P_A, P_{-A} = 1/4$ and the probability of the inner level (0) is $P_0 = 1/2$, as shown earlier in Section 3.2.

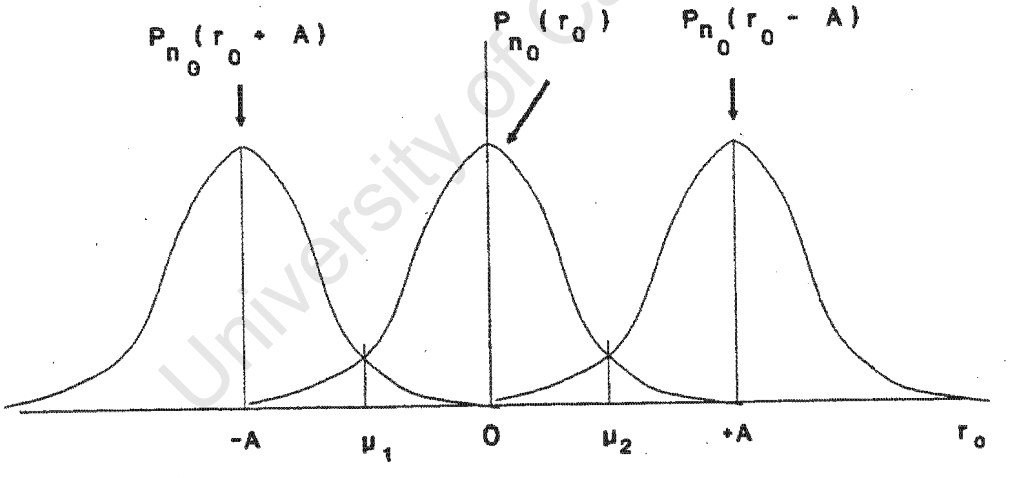


Figure 7.7 - Probability density functions of receiver output in duobinary case.

For this calculation the decision thresholds are set at half the distance between sample values. We assume input AWGN with probability density $P_{n0}(n0)$ and variance σ_0^2 , which means that the output noise is also Gaussian in distribution. The resulting probability densities associated

with the 3 duobinary levels at the sample instants, at the receiver output, are shown in Figure 7.7. The two decision thresholds are denoted μ_1 and μ_2.

The probability of a source symbol **A** being detected at the receiver as a symbol other than **A** (i.e. an error) is denoted as Pe_A. Suppose level **A** is transmitted. An error occurs if the receiver output r_o satisfies $\mu_1 < r_o < \mu_2$. If a **-A** is sent, an error also occurs if r_o falls in this same region. If 0 is sent, an error occurs if $r_o < \mu_1$ or $r_o > \mu_2$. The average error probability is then

$$Pe_{ave} = Pe_A.P_A + Pe_{-A}.P_{-A} + Pe_0.P_0$$

$$= 2.Pe_{-A}.P_{-A} + Pe_0.P_0 \qquad \text{since } P_A = P_{-A}$$

$$\text{where} \qquad P_{-A} = 1/4$$
$$P_0 = 1/2$$

$$= 2\left[\frac{1}{4}\int_{\mu_1}^{\mu_2} (\sqrt{(2\pi)}\sigma)^{-1}\, e^{-(r0+A)^2/2\sigma^2}\, dr_0\right]$$

$$+ 2\left[\frac{1}{2}\int_{\mu_2}^{\infty} (\sqrt{(2\pi)}\sigma)^{-1}\, e^{-(r0)^2/2\sigma^2}\, dr_0\right]$$

The limits μ_1 and μ_2 in the first term are due to the fact that if the noise on the **-A** level is large enough to cause a signal transition over μ_2, the symbol will be correctly decoded anyway since **-A** and **A** convey the same data. The factor of 2 in the second term arises from the symmetry of the Gaussian curve centred on message 0. If we make the substitutions $x = r_0/\mu_2$, $y = (r_0 + \mathbf{A})/\mu_1$ and $z = (r_0 + \mathbf{A})/\mu_2$ we get

$$Pe_{ave} = \int_{\mu_2/\sigma}^{\infty} (\sqrt{(2\pi)})^{-1} e^{-x^2/2}\, dx$$

$$+ \frac{1}{2}\left[\int_{(\mu_1+\mathbf{A})/\sigma}^{\infty} ((\sqrt{(2\pi)})^{-1} e^{-y^2/2}\, dy - \int_{(\mu_2+\mathbf{A})/\sigma}^{\infty} (\sqrt{(2\pi)})^{-1} e^{-z^2/2}\, dz \right]$$

$$= \text{Erfc}(\ \mu_2/\sigma\ ) + 1/2\ \text{Erfc}(\ (\mu_1 + \mathbf{A})/\sigma\ ) - 1/2\ \text{Erfc}(\ (\mu_2 + \mathbf{A})/\sigma)$$

If we take $\mu_1 = -\mathbf{A}/2$ and $\mu_2 = \mathbf{A}/2$ then we get

$$Pe_{ave} = 3/2\ \text{Erfc}(\ \mathbf{A}/2\sigma\ ) - 1/2\ \text{Erfc}(\ 3\mathbf{A}/2\sigma\ )$$

The average signal power is $S = (1/4)\mathbf{A}^2 + (1/4)(-\mathbf{A})^2$, so $S = \mathbf{A}^2/2$. The noise power is $N = \sigma^2$, so $A/2\sigma = \sqrt{(S/2N)}$. Thus the result can be written

> for 3-level class 1 PRS with decision thresholds midway between the signal amplitudes, the probability of a bit error is
>
> $$Pe_{ave} \approx 3/2\ \text{Erfc}\ [\ \sqrt{(\ S/2N\ )}\ ]$$

For 7-level class 1 partial response, the derivation for average probability of error follows the same path as the above derivation. The calculations are lengthy, so only the result is given here.

> For 7-level class 1 PRS with decision thresholds midway between the signal levels, the probability of a bit error is
>
> $$Pe_{ave} \approx 2.8\ \text{Erfc}\ [\ \sqrt{(\ 0.1\ S/N\ )}\ ]$$

9-QPRS and 49-QPRS error performance

The probability of bit error for a quadrature modulated system (QPRS) is calculated as follows:

The probability of a bit error in the I channel is denoted as P_{eI} and the probability of error in the Q channel is denoted as P_{eQ}. The probability of both the I and the Q channel bits being correctly decoded is P_c, where

$$\begin{aligned} P_c &= (1 - P_{eI})(1 - P_{eQ}) \\ &= 1 - 2P_{eI} + P_{eI}^2 \quad \text{since } P_{eI} = P_{eQ}. \\ &\approx 1 - 2P_{eI} \quad \text{for } P_{eI} < 10^{-2} \end{aligned}$$

The overall probability of a bit error is therefore

$$P_{be} \approx 2P_{eI}. \quad \text{for } P_{eI} < 10^{-2}$$

For class 1 9-QPRS the overall probability of a bit error is

$$P_{be} = 3\ \text{Erfc}\ [\ \sqrt{(\ S/2N\ )}\ ]$$

For class 1 49-QPRS the overall probability of a bit error is

$$P_{be} = 5.6\ \text{Erfc}\ [\ \sqrt{(\ 0.1\ S/N\ )}\ ]$$

These equations give the theoretical curves that the measured results are compared with.

7.4.3 MEASURED BIT ERROR RATES FOR 3-PRS, 9-QPRS AND 49-QPRS

7.4.3.1 MEASUREMENT SETUP

The receiver circuit is designed to count the errors incurred in transmission by comparing the decoded bit stream to a correctly delayed version of the original data stream. The comparison is done at software level in the receiver DSP56001 processor. A Port C pin is used as an output clock for the error count. If an error is detected, a narrow clock pulse is output on this pin. The pin is connected to a frequency counter which counts the errors by counting these clock pulses. White Gaussian noise is added to the received signal using the noise generating circuit described in section 6.2.4. The measurement procedure is as follows:

1. The noise level is set to zero and the transmitter is connected to the receiver via the noise generating circuit.

2. The system is started up, and the output signal levels (as shown in Figure 7.5) from the receiver are measured on an oscilloscope. From this the average signal power is calculated.

3. The transmitter is disconnected from the noise generator, and the noise power from the generator is increased to some level.

4. The RMS value of the noise is measured at the receiver lowpass filter output. This is done using a Hewlett-Packard digitizing oscilloscope.

5. The transmitter is reconnected to the noise generator, and the combined signal plus noise is then input to by the receiver.

6. The bit error rate is measured on the frequency counter.

7.4.3.2 BIT ERROR RATE RESULTS

The signal levels for polar 3-PRS occur at **-A**, 0 and **A**. The measured value for **A** was 1V. The average signal power is then $S = A^2/2 = 0.5W$ (assuming a 1 ohm load). A range of RMS noise values were measured at the receiver lowpass filter output (output taken from the receiver DAC), and the corresponding BER value was read. The results are shown in Table 7.1. Figure 7.8 shows the plot of theoretical and measured bit error rates for 2400 bps using 3-PRS.

RMS Noise (mV)	Signal-to-noise ratio (dB)	Bit Error rate
281	8.02	3.92 X 10E-1
223	10.02	2.01 X 10E-1
205	10.75	4.8 X 10E-2
155	13.18	1.19 X 10E-2
125	15.02	2.04 X 10E-3
108	16.35	4.61 X 10E-4
98.6	17.29	1.58 X 10E-4
84.2	18.48	5.04 X 10E-6
74.5	19.55	2.9 X 10E-8

Table 7.1 - 2400 bps bit error rate and SNR measurements.

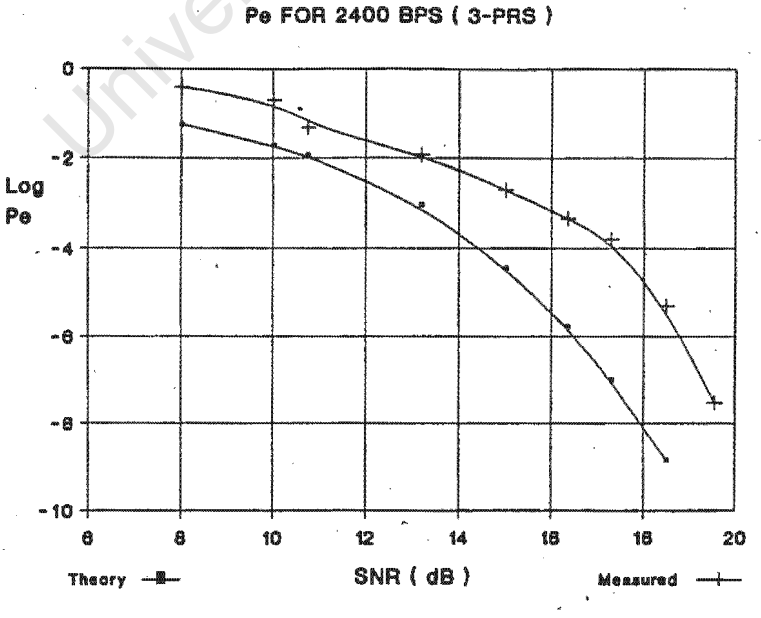


Figure 7.8 - Bit error rate performance for 2400 bps transmission.

The signal levels for 9-QPRS occur at **-A**, 0 and **A** for both the I and the Q channels. The measured value for **A** was 0.88V. The average signal power is S = $\mathbf{A}^2/2$ = 0.387W per channel. The measured RMS noise values and BER are given in Table 7.2. Figure 7.9 shows the theoretical and measured bit error rates for 4800 bps.

RMS Noise (mV)	Signal-to-noise ratio (dB)	Bit Error rate
295	6.48	2.67 X 10E-1
239	8.31	1.63 X 10E-1
178	10.87	1.11 X 10E-1
142	12.83	4.40 X 10E-2
112	14.89	1.0 X 10E-2
95	16.31	3.48 X 10E-3
94	16.41	2.30 X 10E-3
91	16.61	1.30 X 10E-3
79	17.91	6.0 X 10E-4
74	18.49	2.56 X 10E-4
69	19.10	7.9 X 10E-5
65	19.62	3.7 X 10E-5
60	20.31	6.0 X 10E-6
57	20.76	1.28 X 10E-6
54	21.23	1.7 X 10E-7

Table 7.2 - 4800 bps bit error rate and SNR measurements.

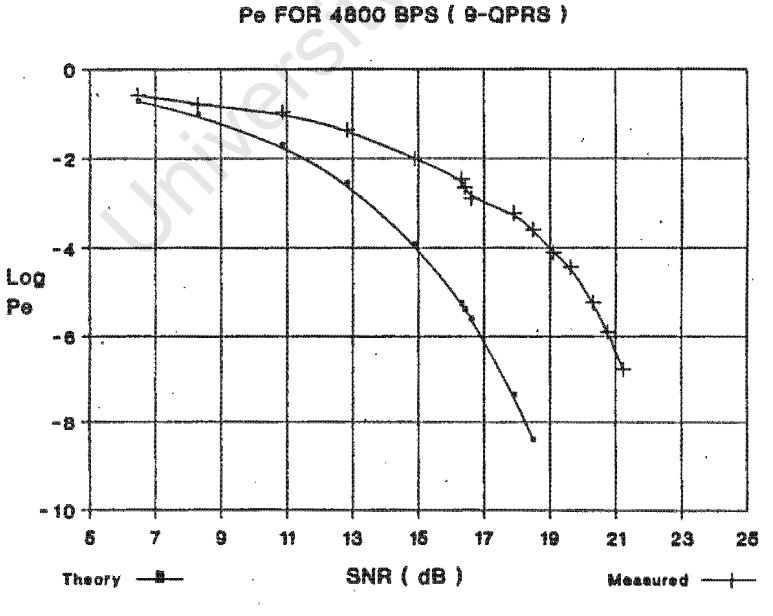


Figure 7.9 - Bit error rate performance for 4800 bps transmission.

The signal levels for 49-QPRS occur at ±3**A**, ±2**A**, ±**A** and 0 for both the I and the Q channels. The measured value for **A** was 0.32V. The average signal power is therefore $S = 0.125(3\mathbf{A})^2 + 0.25(2\mathbf{A})^2 + 0.375(\mathbf{A})^2 = 0.256W$ per channel. The measured RMS noise values and BER are given in Table 7.3. Figure 7.10 shows the error rate performance for 9600 bps transmission.

RMS Noise (mV)	Signal-to-noise ratio (dB)	Bit Error rate
65.8	17.72	1.11 X 10E-1
54.0	19.43	6.50 X 10E-2
47.0	20.64	2.20 X 10E-2
42.5	21.51	1.4 X 10E-2
38.0	22.49	5.70 X E-3
35.0	23.20	2.60 X 10E-3
31.0	24.25	1.40 X 10E-3
26.0	25.78	2.1 X 10E-4
22.5	27.03	1.4 X 10 E-5
19.3	28.37	1.22 X 10E-6
12.5	32.14	< 1 X 10E-11

Table 7.3 - 9600 bps bit error rate and SNR measurements.

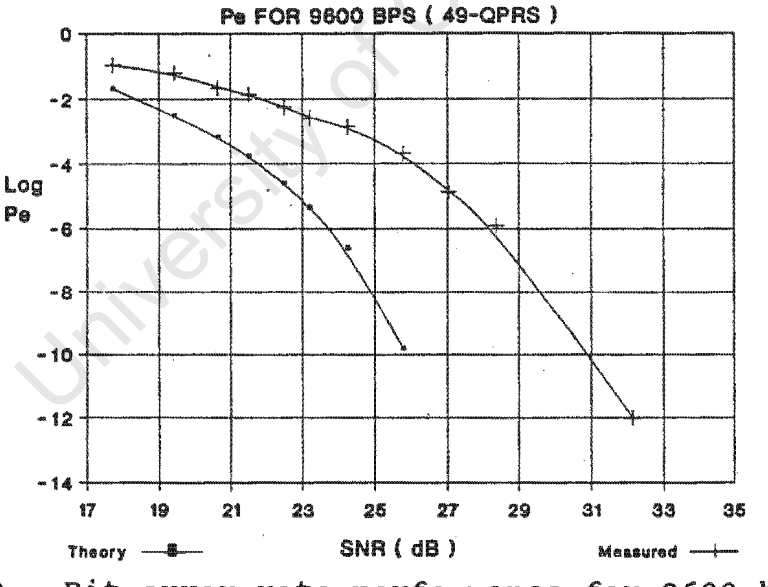


Figure 7.10 - Bit error rate performance for 9600 bps transmission.

Referring to Figures 7.8, 7.9 and 7.10 it is seen that the measured error curves have the characteristic 'waterfall' shape of the theoretical error curves. There is no measurable error floor, i.e. the error rate does not reach a minimum value asymptote. These two results are promising.

For all three transmission rates it is seen that there is some SNR performance degradation of the measured bit error rates. This is approximately 2 - 3dB for the 2400 bps case, 3 - 4dB for the 4800 bps case, and 4 - 5dB for the 9600 bps case. Often there is a substantial difference between the measured and theoretical performance of a given modulation scheme. An example is given by Feher [10.4]. The reason for the degradation in this project is not fully understood, however. It is presumed that the degradation can be attributed to one of two possibilities. These are:-

1. The discrepancy is due to a measurement error, implying that the system actually gives better performance than indicated.

2. There are inherent factors in the modem design that degrade the error performance as compared to theoretical predictions.

With reference to point 1 above, the following observations are made. The method of measuring the signal power at the output of the receiver lowpass filter is assumed to be correct, as it is a very simple and accurate measurement. The measured values of noise power, however, could be misleading. A Hewlett-Packard digitizing oscilloscope was used to measure the RMS noise voltage. This instrument provides a numerical average of the sampled RMS noise values. If the noise crest factor is low, however, the measured RMS noise voltage does not correspond to the variance of the Gaussian probability density function. This would introduce some error into the measurement. For high values of SNR, this error would become increasingly smaller, and the measured curve would approximate the theoretical curve. As the results show, this is not the case, which indicates that the crest factor is not significant at high SNR. The difference between the measured and the theoretical results therefore cannot be easily explained by measurement errors.

With reference to point 2, the following observations are made. With no noise added, the eye patterns and measured spectra of the transmitted and received signals are seen to be very clean. This implies that there is little inherent degradation in the generated signal. The timing of the synchronizing signals at the receiver did not depend on the action of a phase-locked loop, and thus were not affected by the presence of noise. This implies that the receiver is operating in a near-ideal situation.

Thus it is expected that the overall error performance of the modem should be superior to the measured results. It is suspected that a cause for the discrepancy could be the non-Gaussian characteristics of the noise source.

7.5 REFERENCES

[7.1] F. Davarian & J. Sumida, 'A Multipurpose Digital Modulator', IEEE Comm. Magazine, Feb. 1989, pp. 36-45.

[7.2] K. Feher, *Advanced Digital Communications Systems and Signal Processing Techniques*, Prentice Hall Inc., Englewood Cliffs, New Jersey, 1987, pg. 328.

[7.3] F. Stremler, Introduction to Communication Systems 2nd Edition, Addison-Wesley, Reading, Massachusetts, 1982, pg. 614.

[7.4] K. Feher, *Advanced Digital Communications Systems and Signal Processing Techniques*, Prentice Hall Inc., Englewood Cliffs, New Jersey, 1987, pg. 369.

CHAPTER 8. CONCLUSIONS

This chapter is divided into three sections. The first section provides an overview of the project by giving a brief summary of Chapters 2 - 7. The second section draws conclusions based on the experimental results of the project. The chapter ends with recommendations for future work, based on the conclusions that have been drawn.

8.1 CHAPTER SUMMARY

The necessary background theory for the modem design is presented in Chapter 2. The chapter dealt with two main fields - digital communications and digital signal processing. It was shown that Nyquist's theorems for the elimination of ISI are fundamental for effective digital communications. The PAM technique was described as a simple baseband communications example.

Chapter 3 introduced the theory of partial response signalling, in particular class 1 PRS, which was used in this project. It was shown that PRS systems have inherent spectral properties that allow transmission at the Nyquist rate. The spectral efficiency of PRS was the motivating factor for its choice as a modulation scheme.

The modem structure was described in Chapter 4, with particular emphasis given to the digital architecture used in the project. The Motorola DSP56001 microprocessor was presented as the core of the modem design.

Chapters 5 and 6 describe the design steps for the modem transmitter and receiver respectively. Both chapters discussed the preliminary simulations, the hardware design and the real-time software used in the experimental work.

The results of the experimental work were presented in Chapter 7. This included plots of the modulated andü demodulated signal spectra, eye pattern photographs and bit error rate measurements. A short discussion of the results was given after each section.

8.2 PROJECT CONCLUSIONS

An experimental modem structure intended for use over CCITT Recommendation M.1020 circuits has been developed and investigated. The modem uses class 1 partial response signalling as a modulation technique. This technique allowed successful data transmission in a near-Nyquist bandwidth. The modem is designed to operate at data rates of 2400, 4800 and 9600 bps, using 3-PRS, 9-QPRS and 49-QPRS modulation respectively. A DSP56001 microprocessor was used to implement the majority of the signal processing steps in the modem.

The results presented in Chapter 7 attempted to quantify the system in terms of three concepts - the time domain characteristics (eye patterns), the signal spectral properties, and the bit error rate versus signal-to-noise ratio performance.

The use of a digital architecture has allowed very precise and predictable signal characteristics to be achieved. This was largely due to the use of high-order linear-phase digital filtering. The eye patterns displayed minimal ISI, and the signal spectra showed excellent out-of-band attenuation of approximately 50dB, thus complying with the bandwidth requirements of the M.1020 line. It was found that the optimum digital lowpass filters for the transmitter and the receiver were square-root raised-cosine filters with roll-off factor $\alpha = 0.1$. In a real application it might not be necessary to have such a sharp roll-off since the modem would operate over a leased line. It is therefore concluded

that the time and frequency domain characteristics of the modem structure appear to be excellent.

The measured bit error rate performance of the modem is slightly inferior to the theoretically predicted bit error rate performance. The tests were performed for all three bit rates of 2400, 4800 and 9600 bps. As the bit rate increased, the degradation of the measured results became more severe. It was noted, however, that practical systems often show degraded performance when compared to theoretical predictions, and that this degradation can be of the order of 2 - 5dB, as was the case in this project. The actual cause for the degradation in this case is not known, but it is suspected that the noise source was not Gaussian in nature. The measured bit error rate results showed that there was no measurable error floor, which indicates that the experimental system is not marginal. Based on these results it is concluded that the system delivers acceptable error performance.

It is expected that there would be added degradation of the error performance if a phase-locked loop is used for timing synchronization at the receiver. This could introduce timing jitter and sampling discrepancies into the system. The pilot tone scheme would appear to provide good synchronization.

8.3 RECOMMENDATIONS

Based on the research work done on this project and the conclusions drawn in the previous section, the following recommendations are made:-

1. Further development of this modem structure is encouraged to investigate its viability as an operational, marketable device. This recommendation is based on the promising error performance of the modem structure and the very precise signal characteristics that digital circuit techniques can attain.

2. Continued investigation should be directed primarily at adaptive equalization and timing synchronization methods at the receiver. This is a very critical aspect of modem design.

3. The technique of using pilot tones to carry the synchronization information must be investigated. It is expected that a pilot tone would provide a stable signal for a phase-locked loop at the receiver.

4. The use of optimal decoding methods at the receiver should be investigated. This would improve the error performance of the modem.

5. If further investigations of this modem structure are conducted, digital techniques should be employed wherever possible. The stability and flexibility of a digital architecture, coupled with the ease of digital circuit implementation, are particularly well-suited to this design application.

APPENDICES

A - TRANSMITTER AND RECEIVER CIRCUIT DIAGRAMS

B - IMPORTANT DSP56001 REGISTERS

C - TRANSMITTER REAL-TIME ASSEMBLER PROGRAM CODE 'TRANSMIT.ASM'

D - RECEIVER REAL-TIME ASSEMBLER PROGRAM CODE 'RECEIVE.ASM'

E - REAL-TIME HOST PASCAL PROGRAM LISTING 'TRANSMIT.PAS'

F - TRANSMITTER SIMULATION PROGRAM LISTINGS

G - RECEIVER SIMULATION PROGRAM LISTINGS

H - FILTER GENERATOR PROGRAM LISTING 'SIMPSON.PAS'

I - M.1020 FREQUENCY CHARACTERISTICS

J - DIGINET

K - PUBLISHED PAPER

APPENDIX A. TRANSMITTER AND RECEIVER CIRCUIT DIAGRAMS

This appendix contains the six circuit diagrams of the transmitter and receiver external hardware. The listed diagrams are:-

- External program and X data memory.
- Clock circuit.
- Transmitter digital-to-analogue converter.
- Butterworth analogue smoothing filter.
- Pseudorandom bit sequence generator, 7 and 22 register length.
- Receiver analogue-to-digital and digital-to-analogue converter.
- Noise generator circuit.

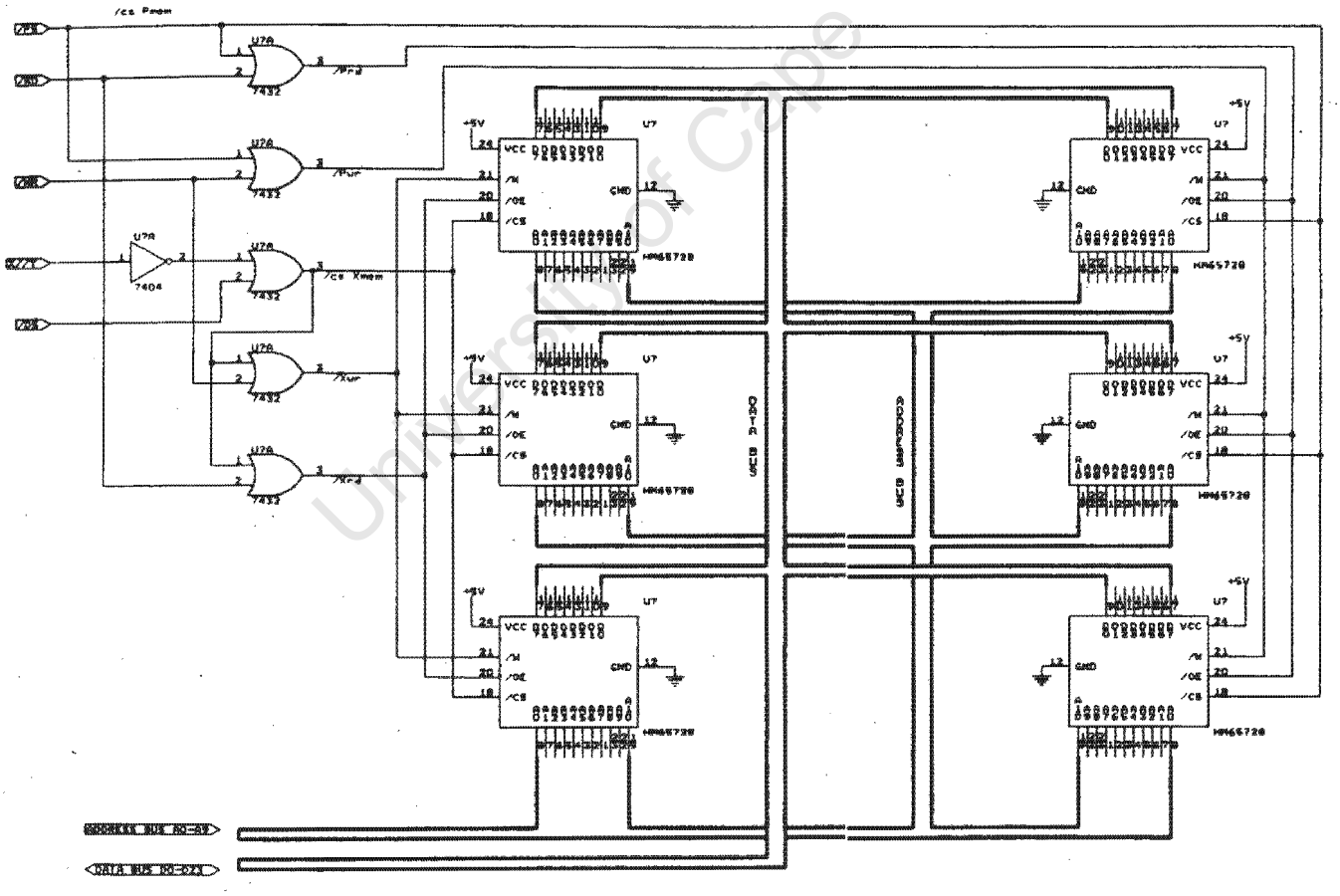


Figure A.1 - External program and data memory.

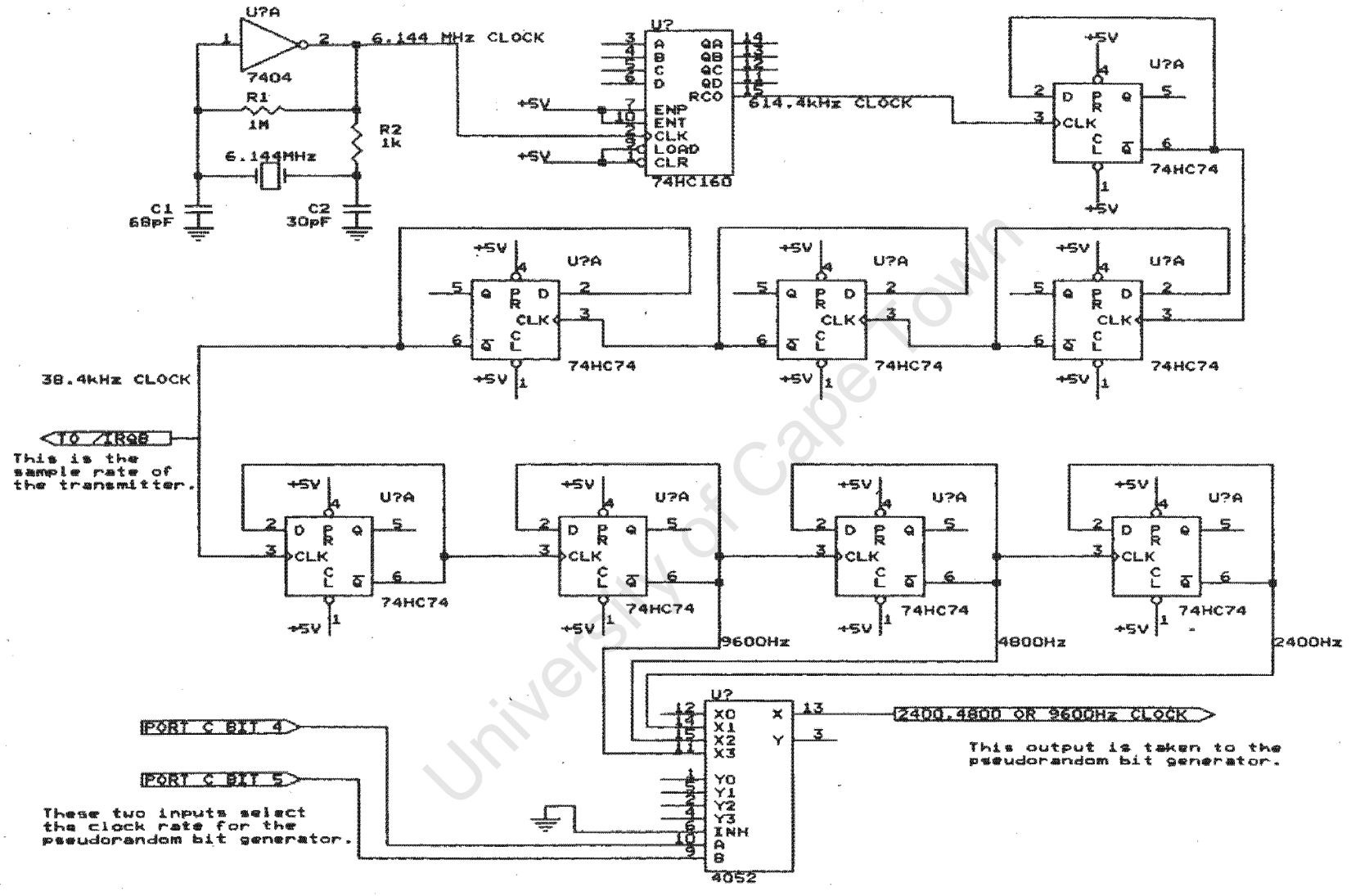


Figure A.2 - Clock circuit diagram.

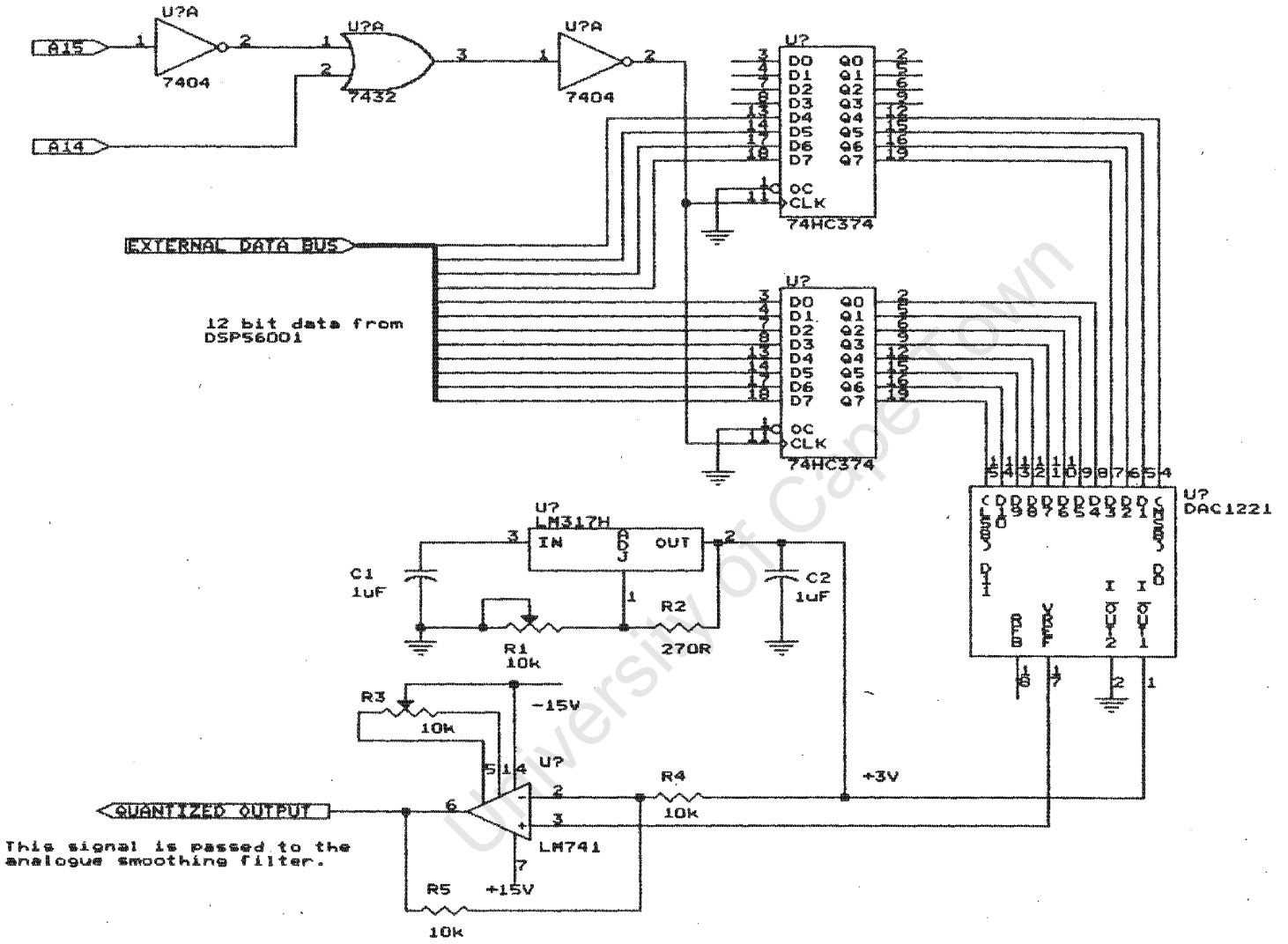


Figure A.3 - Transmitter digital-to-analogue converter.

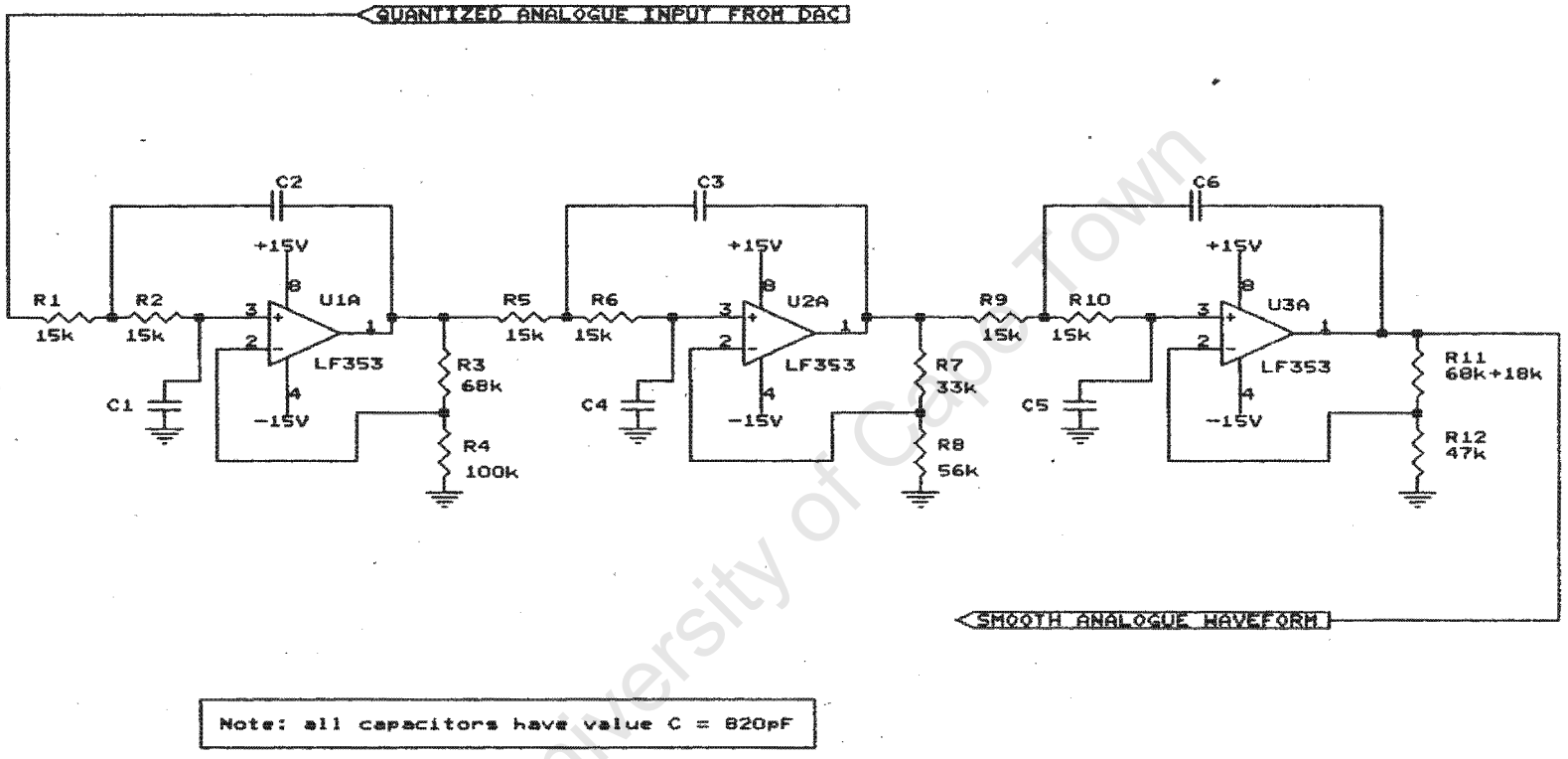


Figure A.4 - Butterworth analogue smoothing filter.

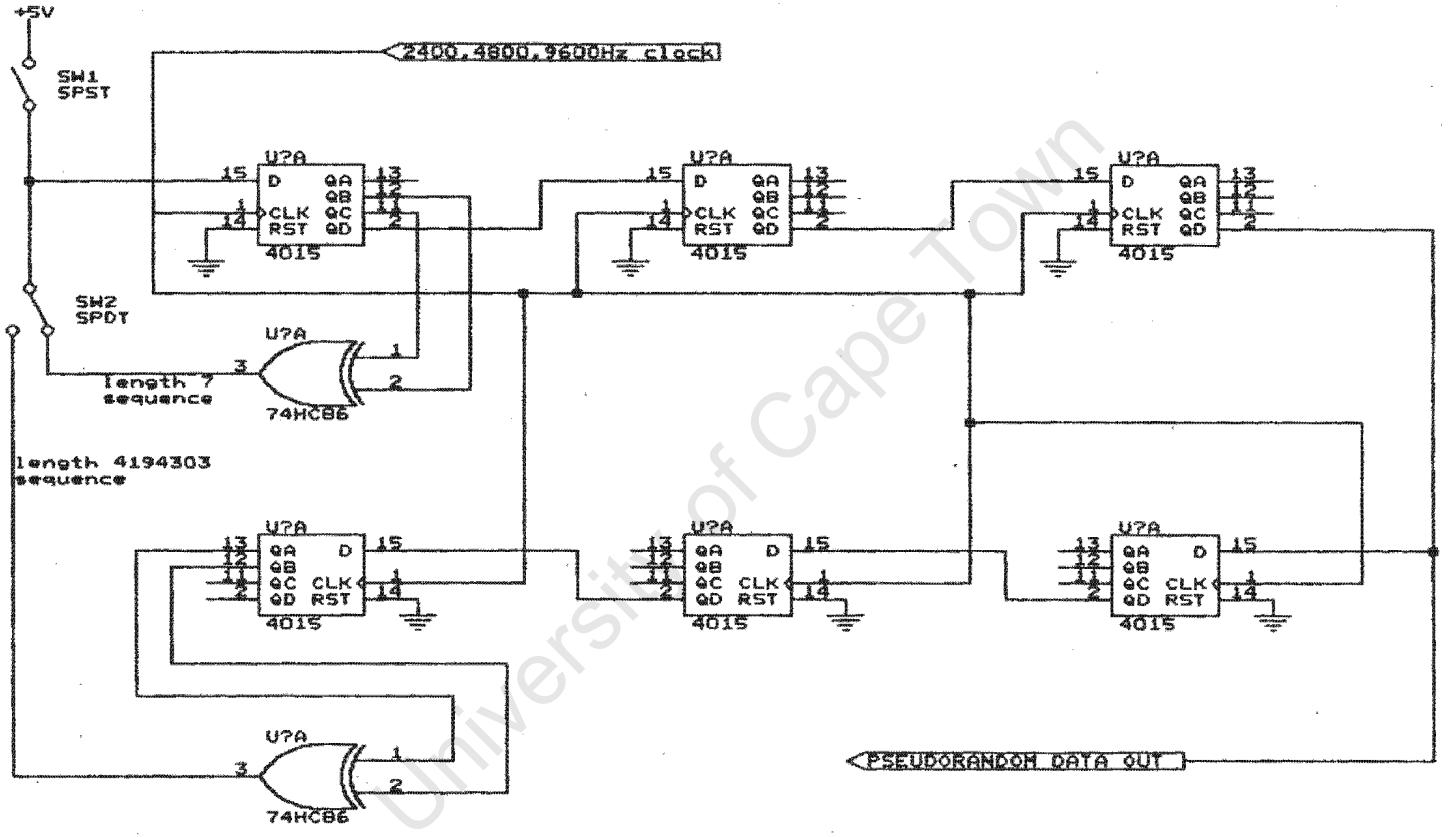


Figure A.5 - Pseudorandom bit sequence generator,
7 and 22 register length.

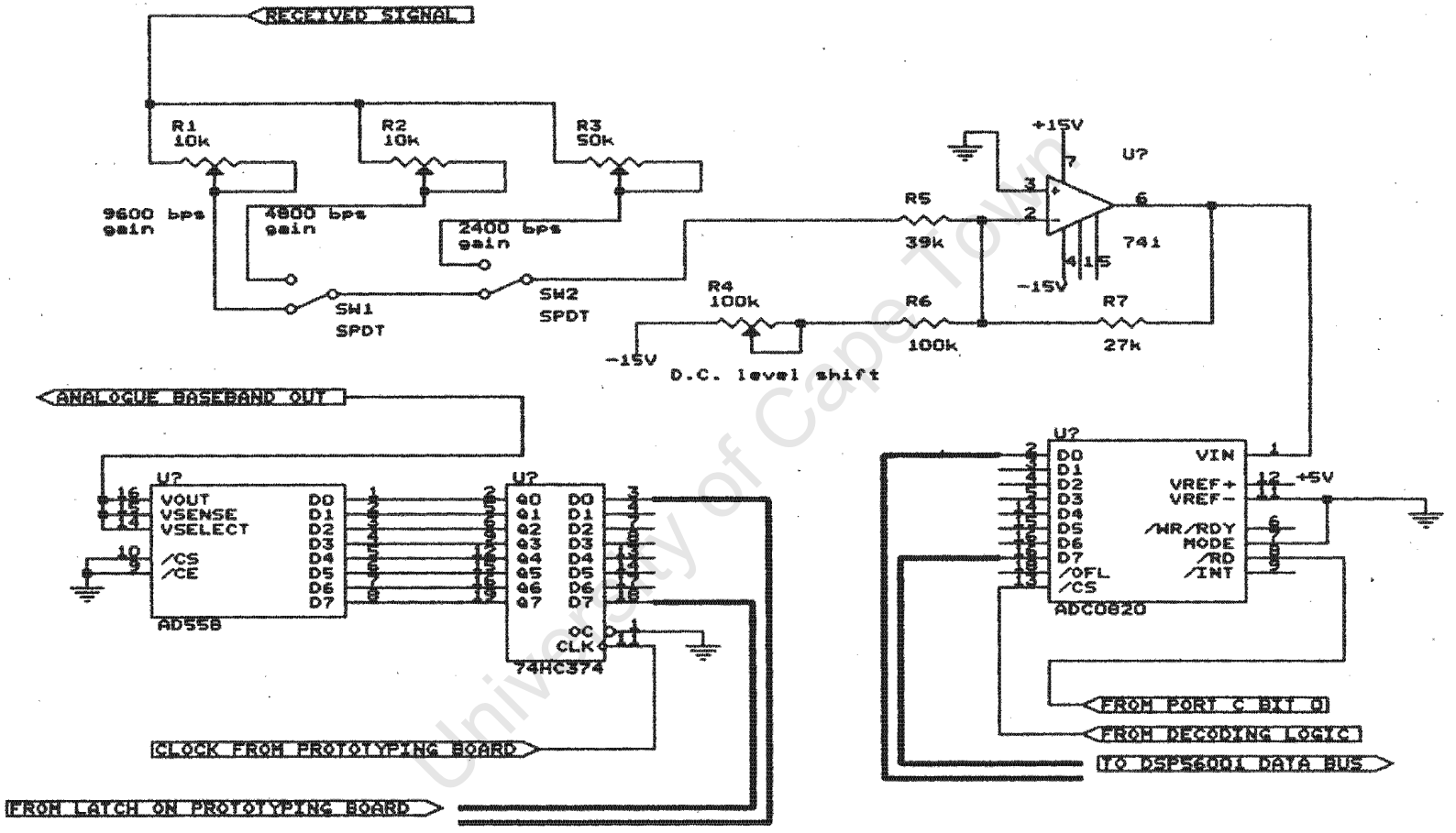


Figure A.6 - Receiver analogue-to-digital converter and output DAC.

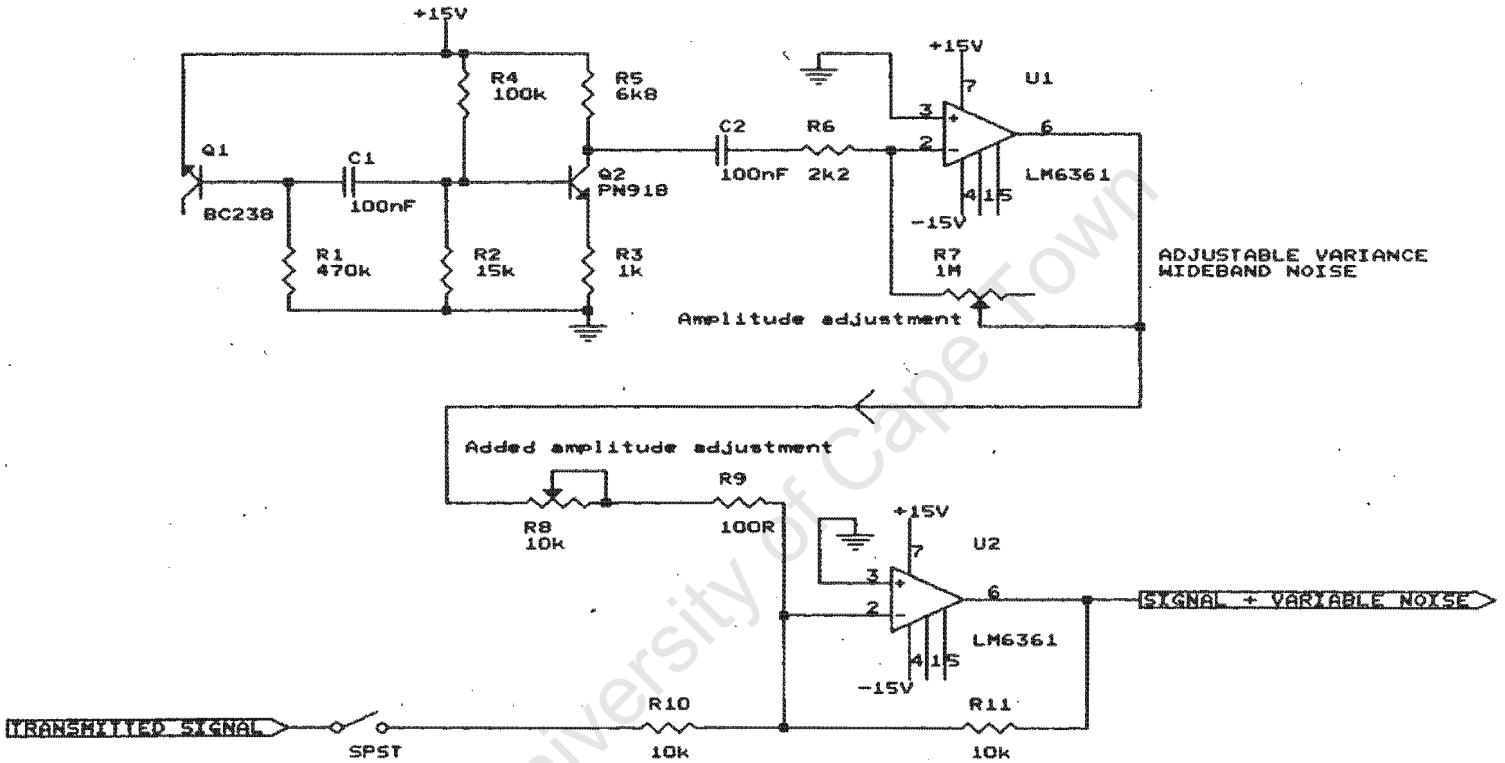


Figure A.7 - Noise generator.

APPENDIX B. IMPORTANT DSP56001 REGISTERS

There are several host interface and Port C registers in the DSP56001 microprocessor that are used frequently in the software for this project. The DSP56001 and the host computer access different sets of registers in the host interface, and these two programming viewpoints are explained here. The Port C is only accessible to the DSP56001. The relevant Host interface and Port C registers are listed below with a short description of their functions. These registers are referenced in the software sections of Chapters 8 and 9.

HOST INTERFACE - DSP56001 VIEWPOINT

The host interface is seen as a memory-mapped peripheral occupying the X memory space X:$FFE8 - X:$FFEB. Access to the host interface registers is achieved by writing to or reading from these registers.

Host Data Register X:$FFEB

This is a read/write location used for DSP/host data transfers. It consists of two groups (one for DSP-host transfer, one for host-DSP) of 3 byte-wide registers, enabling 24-bit data transfer between the DSP56001 and the host computer. Data loaded into X:$FFEB from the DSP appears on the host side of the interface in the 8-bit data registers at host addresses $5, $6 and $7, and vice-versa, assuming that the destination register in either case is clear.

Host Status Register X:$FFE9

This is an 8-bit read-only register reflecting the status of the host interface. The individual bits have the following meaning:-

HRDF - host receive data full flag indicates that data has been transferred to X:$FFEB from the host computer side of the interface.

HTDE - host transmit data empty flag indicates that data has been transferred from X:$FFEB to the host computer side of the interface, and is now empty.

HCP - host command pending flag indicates that the host computer has set the HC bit at address $1, and a host command interrupt is pending.

HF0,HF1 - two host flags for general purpose flagging from the host to the DSP side of the interface.

DMA - not used in this application.

Host Control Register X:$FFE8

This is an 8-bit read/write register to control the interrupts and the flags in the host interface. The individual bits have the following meanings:-

HRIE - host receive interrupt enable is used to enable a DSP56001 interrupt when the receive data register in X:$FFEB is full.

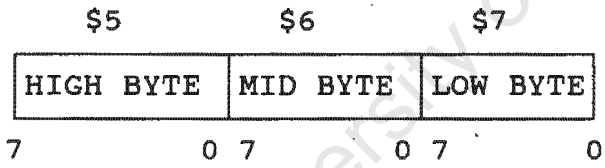
HTIE - host transmit interrupt enable is used to enable a DSP56001 interrupt when the transmit register at X:$FFEB is empty.

HCIE - host command interrupt enable is used to enable a DSP56001 interrupt when the HCP bit in the Host Status Register (X:$FFE9) is set.
HF2,HF3 - host flags used for general purpose DSP-to-host flagging.

HOST INTERFACE - HOST COMPUTER VIEWPOINT

Here the host interface is seen by the host computer as eight byte-wide memory locations. The host interface sees the eight locations at addresses $0 - $7, but when using the PERALEX prototyping card the address $0 is at the base address of the computer expansion port. An example in this project is when the host computer accesses the register at $0 on the host side of the interface, the address $2B0 is used by the host. This gets decoded on the prototyping card to the address $0. Address $2B1 gets decoded as address $1, and so on.

Transmit/Receive Byte Registers $5, $6 and $7

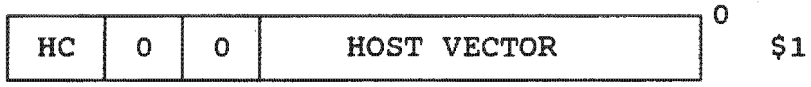


Address $5 contains the high data byte, $6 contains the middle byte and $7 contains the low data byte. When writing and reading, the low byte ($7) must always be accessed last to ensure correct interface operation.

Interrupt Control Register $0

The only bits of interest in this register are HF0 and HF1, the general-purpose flags used for host-to-DSP flagging. The rest of the bits are not used in this project.

Command Vector Register $1



HV bits 0-4 - the host vector address bits are set by the host. They select the 5-bit program address to be jumped to when the host vectors an interrupt to the DSP56001. The actual program address that the DSP56001 vectors to is 2*HV.
HC - the host command bit is set by the host computer when requesting a vectored interrupt to the address specified in HV bits 0-4. When the DSP services the interrupt, this bit is cleared.

Interrupt Status Register $2

RXDF - receive data register full indicates that the Receive Byte Registers (address $5, $6 and $7) have received data from the DSP side of the host interface (when the bit is set).
TXDE - transmit data register empty indicates that the data in the Transmit Byte Registers (address $5, $6 and $7) has been transferred to the X:$FFEB register on the DSP side of the host interface, and the Transmit Byte Registers are empty (when the bit is set).
TRDY - transmitter ready indicates that both the X:$FFEB register on the DSP side of the host interface and the Transmit Byte Registers are empty.
HF2,HF3 - host flags 2 and 3 are used for DSP-host flagging.

PORT C REGISTERS

In this project the Port C pins are programmed for general purpose I/O. Any Port C pin can be programmed to be either input or output. The registers and addresses of relevance are given below.

Port C Control Register X:$FFE1

8	7	6	5	4	3	2	1	0	X:$FFE1

Setting a particular bit in this register configures the corresponding pin for serial interface use, and clearing the bit configures the pin for general purpose I/O. All the pins are configured for general purpose I/O in this project.

Port C Data Direction Register X:$FFE3

8	7	6	5	4	3	2	1	0	X:$FFE3

Clearing a particular bit in this register configures the corresponding pin as an input, and setting the bit configures the pin as an output when the pin is selected for general purpose I/O.

Port C Data Register X:$FFE5

8	7	6	5	4	3	2	1	0	X:$FFE5

This is an 8-bit read/write data register. Writing to this register will write to the pins designated as outputs, and reading from it will read from the pins designated as inputs.

APPENDIX C. TRANSMITTER REAL-TIME ASSEMBLER PROGRAM CODE 'TRANSMIT.ASM'

The following program listing is for the real-time transmitter DSP56001 assembler program.

```
;THIS PROGRAM IS FOR THE TRANSMITTER DSP CHIP. THE
;PROGRAM READS A RANDOM DATA STREAM, CODES IT ACCORDING TO
;PARTIAL RESPONSE SIGNALLING RULES AND CONVOLVES IT WITH
;A LOW-PASS FILTER LOOKUP TABLE UNDER INTERRUPT CONTROL. THE
;I AND Q CHANNELS ARE CALCULATED AND THEN MODULATED ONTO
;QUADRATURE CARRIERS.
;
;r0-lookup table pointer
;r1-IRQA counter
;r2-I channel carrier pointer
;r3-stored pilot tone pointer
;r4-buffer pointer
;r7-tone lookup table pointer-only used once when loading
;                              the pilot tone
;r6-Q channel carrier pointer

     opt  fc,mu,s,w,mex

lkpleng      equ   320          ;Lookup table length.
lkpmod       equ   319          ;Lookup table modulus.
lkpoff       equ   16           ;Lookup table offset.
points       equ   16           ;Points per symbol.
symbols      equ   20           ;Symbols in lookup table.
pastI        equ   $050         ;Temporary store for past I
                                ;duobinary value.
pastQ        equ   $051         ;  "         "    "   "   Q
                                ;duobinary value.
offset       equ   $052         ;Offset of four in encoding
                                ;calculations.
modulus      equ   $053         ;Modulus of 3, and level
                                ;shift for encoding.
scale        equ   $054         ;Scaling factor for pilot
                                ;tone.
store        equ   $055         ;Store for input bits.
correct      equ   $056         ;Mem location to correct msb
                                ;for d/a.
mask         equ   $057         ;Stores mask for 12 bits to
                                ;be sent to d/a.
baud         equ   $058         ;Stores baud rate.
buffoff      equ   $060         ;Buffer offset address.
buffmod      equ   $061         ;Buffer modulus.
bufstrt      equ   $062         ;Buffer start address.
numint       equ   $063         ;Stores mod of number of
                                ;interrupts.
                                ;required to read in bit.
bitmask      equ   $064         ;Mask bits in temp store.
status       equ   $065
numout       equ   $066
```

```
        org     p:$0000
        jump    begin1          ;Skip the bootstrap program
                                ;code.
        org     p:$000a         ;/IRQB interrupt.
        jsr     calc
        org     p:$0024         ;2400 bps interrupt.
        jmp     rate1
        org     p:$0026         ;4800 bps interrupt.
        jmp     rate2
        org     p:$0028         ;9600 bps interrupt.
        jmp     rate3
        org     p:$002a         ;19.2 khz clock 180 deg phase
                                ;change.
        bchg    #8,x:$ffe5      ;This is for the benefit of
                                ;the receiver.

;START OF MAIN PROGRAM.

    org  p:$0100
begin1
       movec #6,omr             ;Enable on-board ROMs.
       nop

;TRANSMIT ZERO LEVEL FOR RECEIVER OFFSET DETECTION.

       move  #>$800,a1          ;Sends half full-scale to the
                                ;DAC chip.
level  move  a1,y:$8000         ;Write data to DAC latch.
       rep   #10
       nop
       move  a1,y:$1000         ;Disable latch.
       rep   #10
       nop
sit    jset  #3,x:$ffe9,sit     ;Sit idle until flagged by
                                ;Host interface.
       nop

       jsr  in                  ;Read in lookup table.
       jmp  init                ;Initialize the baud
                                ;parameters.

;ENABLE INTERRUPTS AND PORT C IO PINS.

enable
      move   #$0,a0
      move   a0,x:$ffe1      ;Configure Port C pins as IO.
      move   #$1f0,a0
      move   a0,x:$ffe3

  ;Bit #3 inputs the data stream.
  ;Bit #4 and #5 are the output code for the clock divider
  ;on the external transmitter board.
  ;Bit #6 provides the receiverwith the 2.4 kHz
  ;symbol calculate interrupt( /IRQB ).
  ;Bit #7 provides the receiver with the carrier reset
  ;signal.
  ;Bit #8 provides the receiver with the 19.2 kHz
  ;sampling interrupt ( /IRQA ).
```

```
        move    #$0c30,a0
        move    a0,x:$ffff          ;Set interrupt priority
                                    ;register.
        bset    #2,x:$ffe8          ;Enable the host interrupt.
        bclr    #8,x:$ffe5
        clr     a
        move    a0,sr               ;Clear status register
                                    ;interrupt mask.

;MAIN PROGRAM LOOP

main    nop                         ;Wait for interrupts.
        nop
        jmp     main

;INTERRUPT SERVICE ROUTINE TO PERFORM CONVOLUTION.

calc    movep     #$0,x:$ffff       ;Disable all interrupts.
        jsr     out                 ;Read out previously
                                    ;modulated signal value.
        jsr     count               ;Check to read in bit.
        clr     a       x:(r0)+n0,x1    ;Load first lookup
                                        ;table value.
        clr     b
        btst    #0,y:baud           ;Check if different mac
                                    ;routine required for 2400
                                    ;bit rate.
        jcs     onecalc

;CONVOLVE THE I AND Q CHANNELS.

        do      #symbols,point

        mac     x1,x0,a         y:(r4)+,x0
        mac     x1,y0,b         x:(r0)+n0,x1      y:(r4)+,y0
point
     ;Register a contains the I value,
     ;register b contains the Q value.

        asr     a             x:(r0)+,x1
                            ;Dummy move to update pointer.
        move    a0,x1
                            ;x0 and y0 are preserved.
        clr     a                 y:(r2)+n2,y1
        mpy     x1,y1,a           x:(r0)-n0,x1
                            ;Modulate onto I carrier.
        btst    #0,y:baud
        jcs     sc    ;If bit rate = 2400, only I channel used.
        asr     b                   ;Scale result.
        move    b0,x1
        clr     b               y:(r6)+n6,y1
        mpy     x1,y1,b       ;Modulate onto Q carrier.
        add     b,a           ;Add modulated I and Q channels.
sc      btst    #2,y:baud
        jcs     tone
        asl     a                   ;Additional scale for 2400
                                    ;and 4800 bps signals.
```

```
        btst    #0,y:baud
        jcc     tone
        asl     a
tone    move    x:(r3)+,b               ;Add the pilot tone
        add     b,a
        do      #12,shift               ;Scale result for DAC.
        asr     a
shift
        move    y:mask,x1                ;Mask bits for DAC.
        and     x1,a
        move    a1,y:correct      ;Correct most significant bit.
        bchg    #11,y:correct     ;This changes value into DAC
                                  ;format.
        move     y:correct,a1
lev     move    a1,y:numout
        jsr     sym                     ;Calculate symbol?
        move    #$0,sr                  ;Clear interrupt mask in
                                        ;status register.
        movep    #$0c30,x:$ffff         ;Re-enable the interrupts.
        rti

;SUBROUTINE TO OUTPUT PREVIOUSLY CALCULATED VALUE.

out     move    y:numout,a1
        move    a1,y:$8000          ;Output to d/a.
        clr     a    r2,y1
        move    #>$124,a1
        move    r2,x1
wait1
        move    a1,y:$1000
        cmp      x1,a         ;If the I carrier is at location
                              ;$124, send carrier reset signal to
                              ;receiver.
        jne      t_out
        bclr    #7,x:$ffe5      ;If necessary,assert the
                                ;carrier sync signal.
t_out   move     #>$130,a1
        cmp      y1,a           ;If the I carrier is at location
                                ;$130, de-assert the carrier
                                ;synchronizing signal.
        jne      w_out
        bset    #7,x:$ffe5
w_out   bchg     #8,x:$ffe5     ;provides the 19.2 kHz clock for
                                ;the receiver.
        rts
```

```
;SUBROUTINE TO COUNT THE 38.4 KHZ INTERRUPTS.
;a bit must be read in every 4th, 8th, or 16th interrupt.

count move   x:(r1)+,a      ;Increment counter (bogus move).
      move   y:numint,a
      move   r1,x1          ;Load value of counter.
      and    x1,a           ;Check if r1 contains multiple
                            ;of 4,8 or 16.
                            ;For 9600 bps, numint contains
                            ;value 3, for 4800 bps, it
                            ;contains value 7, and for 2400
                            ;bps, it contains value 15.
      jseq   read           ;If necessary, read in bit.
      rts

;SUBROUTINE TO READ IN BIT.
;stores bit in location y:$055 with past bits.

read  clr    b
      btst   #3,x:$ffe5      ;Read in new bit.
      jcc    empty
      move   #>$1,b1
empty
      move   y:store,a       ;Load past bits (if any).
      addl   b,a             ;Include new bit.
      move   y:bitmask,x1
      and    x1,a            ;Remove unnecessary bits.
      move   a,y:store       ;Store bits.
      rts

;SUBROUTINE TO CHECK IF NEW SYMBOLS MUST BE CALCULATED.
;new symbol calculated every 16 interrupts.

sym   bset   #6,x:$ffe5  ;De-assert the symbol interrupt
                         ;for the receiver.
      clr    b   r1,x1
      clr    a  #>$4,b1
      cmp    x1,b
      jne    sym2
      bclr   #6,x:$ffe5   ;If necessary, assert symbol
                          ;sampling interrupt.
                          ;This occurs every 16 interrupts,
                          ;with a delay of 4 interrupts
                          ;between transmitter and receiver.
sym2  or     x1,a
      jseq   code           ;If necessary, code new bits.
      rts
```

```
;SUBROUTINE TO ENCODE DATA AND STORE IN BUFFER FOR
FILTERING.

code  move    y:(r4)+n4,x1     ;Bogus move to increment buffer
pointer.
      move    #$1000,r0        ;Reset lookup table pointer.
      move    y:store,a        ;Load 1, 2, or 4 bits into a.
      move    y:modulus,x0
      move    y:offset,y0      ;Offset necessary for PRS
calculations.
      move    y:baud,b0        ;Load baud rate indicator.
      asr     b
      jcs     cont1            ;If bit 0 of y:baud is set,
                               ;indicates 2400.
      asr     a                ;Shift and isolate 1 or 2 upper
                               ;bits.
      asr     b
      jcs     cont1            ;If bit 1 of y:baud is set,
                               ;indicates 4800.
      asr     a

      ;Code the I channel bit(s).

cont1 and     x0,a             ;Mask out unnecessary bits.
      add     y0,a             ;Level shift for encoding
                               ;procedure.
      move    y:pastI,x1
      sub     x1,a             ;Subtract past precoded symbol.
      and     x0,a             ;Take modulus.
      move    a1,y:pastI       ;Store as new precoded symbol.
      add     x1,a             ;Algebraicly add past symbol,
                               ;and level shift to remove d.c.
                               ;component.
      sub     x0,a           y:baud,b0
      move    a1,y:(r4)+       ;Store in buffer.
      asr     b
      jcs     cont2            ;Continue if not 2400 bps.

      ;Code the Q channel bits(s).

      move    y:store,b
      and     x0,b             ;Mask out unnnecessary bits.
      add     y0,b             ;Procede exactly as above.
      move    y:pastQ,y1
      sub     y1,b
      and     x0,b
      move    b1,y:pastQ
      add     y1,b
      sub     x0,b
      move    b1,y:(r4)+
      move    b1,y0          ;Load newly calculated symbols
                             ;into a0 and b0.
cont2 move    a1,x0
      rts

      org     p:$1000          ;USES EXTERNAL PROGRAM MEMORY
```

```
;MAC ROUTINE FOR 2400 BPS

onecalc
      move    x0,y0
      do      #symbols,point2
      mac     x1,y0,a          x:(r0)+n0,x1  y:(r4)+,y0
point2
      move    y0,x0
      jmp     point

;SUBROUTINE TO READ IN THE LOOKUP TABLE AND BAUD RATE.
;THIS IS PLACED IN EXTERNAL PROGRAM MEMORY AS IT IS
;ONLY ACCESSED AT THE BEGINNING OF PROGRAM EXECUTION.

in    move    #$1000,r0           ;Start address of lookup
table.
      move    #lkpmod,m0
      move    #lkpoff,n0

rate  jclr    #0,x:$ffe9,rate     ;Read in bit rate from Host.
      move    x:$ffeb,x0
      move    x0,y:baud

      do      #lkpleng,endin      ;Loop reads in lookup table.
bitin jclr    #0,x:$ffe9,bitin
      move    x:$ffeb,x0
      move    x0,x:(r0)+          ;Store tap weights in
                                  ;external X memory.
endin bset    #8,x:$ffe5          ;Initialise the 19.2 kHz
                                  ;clock for receiver.
      nop
      rts

;SUBROUTINES TO SET THE PARAMETERS FOR 2400, 4800 AND 9600
;BPS OPERATION. THESE ROUTINES ARE CALLED FROM THE
SUBROUTINE
;'INIT'.

sethi                             ;Set parameters for 9600 bps.
      bset    #4,x:$ffe5          ;Output code for the clock
                                  ;divider.
      bset    #5,x:$ffe5
      move    #39,a0              ;Change the encoded symbol
                                  ;buffer length.
      move    a0,y:buffmod
      move    #36,a0
      move    a0,y:buffoff
      move    #4,a0
      move    a0,y:bufstrt
      move    a0,y:offset
      move    #3,a0
      move    a0,y:modulus
      move    a0,y:numint
      move    #>$f,a0
      move    a0,y:bitmask
      nop
      rts
```

```
setmid                                  ;set parameters for 4800 bps
        bclr    #4,x:$ffe5
        bset    #5,x:$ffe5
        move    #39,a0
        move    a0,y:buffmod
        move    #36,a0
        move    a0,y:buffoff
        move    #4,a0
        move    a0,y:bufstrt
        move    #2,a0
        move    a0,y:offset
        move    #1,a0
        move    a0,y:modulus
        move    #7,a0
        move    a0,y:numint
        move    #>$3,a0
        move    a0,y:bitmask
        rts

setlo                                   ;set parameters for 2400
bps
        bset    #4,x:$ffe5
        bclr    #5,x:$ffe5
        move    #19,a0
        move    a0,y:buffmod
        move    #18,a0
        move    a0,y:buffoff
        move    #2,a0
        move    a0,y:bufstrt
        move    #2,a0
        move    a0,y:offset
        move    #1,a0
        move    a0,y:modulus
        move    #$f,a0
        move    a0,y:numint
        move    #>$1,a0
        move    a0,y:bitmask
        rts


;INTERRUPT ROUTINES FOR BIT RATE CHANGE

rate1                           ;2400 bps rate change.
        move    #$0300,sr
        move    #>$1,a0
        move    a0,y:baud
        jmp     init

rate2                           ;4800 bps rate change.
        move    #$0300,sr
        move    #>$2,a0
        move    a0,y:baud
        jmp     init
```

```
rate3                           ;9600 bps rate change.
        move    #$0300,sr
        move    #>$4,a0
        move    a0,y:baud
        jmp     init

;SUBROUTINE TO INITIALIZE THE OPERATING MODE

init    move    #$0300,sr       ;Mask out the interrupts.
        move    #$0,a0
        move    a0,x:$ffff      ;Disable the interrupts.
        move    y:baud,b0       ;Load bit rate.
        asr     b
        asr     b
        asr     b
        jscs    sethi           ;If bit #2 is set then
                                ;rate is 9600 bps.
        move    y:baud,b0
        asr     b
        asr     b
        jscs    setmid          ;If bit #1 is set then
                                ;rate is 4800 bps.
        move    y:baud,b0
        asr     b
        jscs    setlo           ;If bit #0 is set then
                                ;rate is 2400 bps.
        move    #$00,r4         ;Initialize buffer pointers.
        move    y:buffmod,m4
        move    y:buffoff,n4
        move    #0,y0
        do      #64,clear       ;Clear the circular buffer.
        move    y0,y:(r4)+
clear

        move    #0,x0
        move    x0,y:pastI          ;Temp store for past encoded
                                    ;I number.
        move    x0,y:pastQ          ;Temp store for past encoded
                                    ;Q number.
        move    x0,y:store          ;Store for bits as they are
                                    ;input.
        move    x0,y:numout
        move    #$fff,a0            ;12 bit mask for DAC.
        move    a0,y:mask
        clr     a


        move    #$100,r7        ;Pilot tone pointer.
        move    #4,n7           ;Read every 4th point from
                                ;sine lookup.
        move    #$ff,m7
        move    #$100,r2        ;I carrier lookup table pointer.
        move    #12,n2          ;Read every 12th point.
        move    #$ff,m2
        move    #$140,r6        ;Q carrier offset by 90 deg.
        move    #12,n6
        move    #$ff,m6
        move    #$1000,r0       ;Set low-pass filter pointer.
```

```
        move    y:bufstrt,r4    ;Intialise the buffer start
                                ;address.
        move    #$0,r1          ;Counter to read in bits,
        move    #$f,m1          ;mod 16.
        clr     a
        move    a,x0
        move    a,y0

;LOAD AND SCALE PILOT TONE

pilot
        move    #$0,r3
        move    #$3f,m3
        do      #64,shiftt
        btst    #23,y:(r7)
        move    y:(r7)+n7,b
        jscc    neg
        jscs    pos
        move    b,x:(r3)+
shiftt
        move    #$01,r3
        jmp     enable              ;Go to start of program.


;The following two routines negate the b2 register
;and scale the resulting pilot tone value. This gets
;the pilot tone into the correct format for the d/a chip.

neg     move    #$00,b2
        do      #6,scale2
        asr     b
scale2
        rts

pos     move    #$ff,b2
        do      #6,scale3
        asr     b
scale3
        rts

;THE END OF THE PROGRAM.

        end
```

APPENDIX D. RECEIVER REAL-TIME ASSEMBLER PROGRAM CODE 'RECEIVE.ASM'

The following listing is for the receiver real-time DSP56001 assembler program.

```
;THIS PROGRAM IS THE .ASM FILE OF THE RECEIVER.
;THE RECEIVER IS SUPPLIED WITH 2 INTERRUPTS.
;/IRQA CAUSES SAMPLING OF THE RECEIVED WAVEFORM AT 19.2KHZ.
;THE SAMPLE VALUES ARE DEMODULATED AND STORED IN A CIRCULAR
;BUFFER WHERE THEY CAN BE CONVOVED WITH THE LOW-PASS FILTER
;VALUES. /IRQB CAUSES THE DEMODULATED VALUES IN THE CIRCULAR
;BUFFER TO BE CONVOLVED WITH THE LOW-PASS FILTER LOOKUP
;TABLE, GIVING THE SYMBOL VALUE. THE PREVIOUSLY CALCULATED
;SYMBOL VALUE IS OUTPUT. THE VALUE IS ALSO DECODED TO
;RECOVER THE BIT STREAM.

;r0-pointer for bit delay buffer for error checking
;r1-lookup table pointer
;r2-Q channel buffer pointer for convolution
;r3-Q channel buffer pointer for sample and read routine
;r4-I channel carrier pointer
;r5-I channel buffer pointer for sample and read routine
;r6-I channel buffer pointer for convolution
;r7-Q channel carrier pointer

     opt  fc,mu,s,w,mex

offset   equ     $0fe    ;stores the d.c offset value of the
                         ;ADC, which supplies a signal
                         ;between 0 and 5V. This must be
                         ;subtracted from the received
                         ;signal to allow proper
                         ;demodulation.
a1save   equ     $0fd    ;Stores for the accumulator
                         ;registers etc.
a0save   equ     $0fc    ;These are necessary as the /IRQA
                         ;interrupt routine
y1save   equ     $0fb    ;can occur during the /IRQB
                         ;convolution routine.
y0save   equ     $0fa
x1save   equ     $0f9
x0save   equ     $0f8
bitrate  equ     $0f7    ;Stores a bitrate indicator for the
                         ;3 data rates.
oldvalI  equ     $0f6    ;Stores the previously calculated I
                         ;symbol value.
oldvalQ  equ     $0f5    ;Stores the previously calculated Q
                         ;symbol value.
bitsout  equ     $0f4    ;Stores the 8-bit sequence to be
                         ;output.
newbits  equ     $0f3    ;Temp store for the new 8-bit
                         ;sequence.

lkpleng  equ     160     ;Lookup table length.
lkpmod   equ     159     ;Modulo for circular calculations.
```

```
buffmod  equ     159        ;Input buffer same length as lookup
                            ;table.

     org  p:$0000           ;Skip the bootstrap program code.
     jmp  begin1

        org     p:$0008      ;IRQA for 19.2 kHz clock ( causes
                             ;sampling ).
        jsr     readin
        org     p:$000a      ;IRQB for symbol calculation,
                             ;data output.
        jsr     calc
        org     p:$0024    ;Sets the parameters for 2400 bps.
        jsr     low
        org     p:$0026    ;Sets the parameters for 4800 bps.
        jsr     mid
        org     p:$0028    ;Sets the parameters for 9600 bps.
        jsr     high
        org     p:$002a      ;Increases the data-compare
                             ;buffer length.
        jsr     shftu
        org     p:$002c        ;Decreases buffer length.
        jsr     shftd

;INITIALISE THE VARIABLES AND START THE PROGRAM.

        org     p:$0080
begin1  move    #$6,omr        ;Enable on-board ROMS.
        clr     a
        move    a0,x:$ffff   ;Disable all interrupts.
        move    a0,x:$ffe1

    ;Configure Port C as general purpose IO.
    ;Bit #0 is an output to activate A/D read.
    ;Bit #1 is the output to transmit the decoded
    ;bit stream.
    ;Bit #2 is an input, polled to determine if the
    ;carriers need resetting.
    ;Bit #4 is used to output a delayed version
    ;of the bit stream from the transmitter.
    ;Bit #6 is used as an error indicator clock, and is
    ;pulsed when the transmitted and decoded
    ;bits are different.

        move    a0,x:bitrate ;default to 2400 bps.
        move    a0,x:bitsout ;Clear output bits.
        move    a0,x:newbits
        move    #$100,r4       ;I carrier.
        move    #24,n4         ;Offset corresponding to 19.2
                               ;kHz sampling.
        move    #$ff,m4
        move    #$140,r7       ;Q carrier.
        move    #24,n7
        move    #$ff,m7
        move    #$1100,r0      ;External memory space for
                               ;delay buffer.
```

```
        move    #$a8,m0
        move    #$53,a0         ;Configure pin 0,1,4,6 as
                                ;output.
        move    a0,x:$ffe3
        movep   #$c37,x:$ffff

  ;Enable interrupts. Host interrupt is given priority 3,
  ;/IRQA is given priority 3, and /IRQB is given priority 2.

        bset    #2,x:$ffe8      ;Unmask host interrupt.
        andi    #0,mr
        jsr     table           ;Read in lookup table, clear
                                ;circular buffer.

;READ IN THE DC OFFSET FROM THE A/D. THIS IS DONE WHILE THE
;TRANSMITTER SENDS A ZERO LEVEL OUTPUT.

        move    y:$2000,a       ;Enable the A/D chip select.
        bclr    #0,x:$ffe5      ;Assert the read signal.
        do      #40,wait0       ;Wait for conversion.
        nop
wait0
        move    y:$2000,a       ;Read in the actual a/d value.
        move    #>$ff,x0        ;Mask the necessary 8 bits.
        and     x0,a
        move    a1,x:offset     ;Store the offset.
        bset    #0,x:$ffe5      ;De-assert the read signal.
        move    a,y:$1000       ;Disable A/D chip select.
        jmp     loop

loop    nop                     ;loop for interrupts
        nop
e_loop  jmp     loop

;/IRQA INTERRUPT ROUTINE TO SAMPLE RECEIVED SIGNAL AT
;19.2KHZ, MULTIPLY I AND Q VALUES BY THE RESPECTIVE
;CARRIERS, AND OUTPUT THE PREVIOUSLY DECODED BIT STREAM.

readin  movep   #$c30,x:$ffff   ;Disable the IRQA interrupt.
        move    y:$2000,a2      ;Enable the a/d chip select.
        bclr    #0,x:$ffe5      ;Assert the read signal.

        move    a1,x:a1save     ;Save registers.
        move    a0,x:a0save
        move    x0,x:x0save
        move    x1,x:x1save
        move    y0,x:y0save
        move    y1,x:y1save
        do      #15,wait        ;Additional wait for
                                ;conversion.
        nop
wait
        move    y:$2000,a       ;Read in actual ADC value
                                ;into a1.
        bset    #0,x:$ffe5      ;De-assert the read signal
                                for ADC.
        move    #>$ff,x1
```

```
        and     x1,a                ;Mask lower 8 bits of
                                    ;register a1.
        move    y1,y:$1000          ;Disable a/d chip select.
        move    x:offset,x0         ;Subtract dc offset for
                                    ;demodulation.
        sub     x0,a    y:(r4)+n4,y1
        move    a,x1    y:(r7)+n7,y0
        mpy     x1,y1,a             ;Multiply by I carrier.
        move    a,y:(r5)+           ;Load new sample into I
                                    ;buffer.
        mpy     x1,y0,a
        move    a,x:(r3)+           ;Load sample into Q buffer.
        btst    #2,x:$ffe5          ;Check for carrier reset sync.
        jcs     cont
        move    #$118,r4            ;If necessary, reset I and Q
                                    ;carriers.
        move    #$158,r7
cont    nop

;THIS SECTION OF THE ROUTINE OUTPUTS THE DATA STREAM VIA
;PORT C BIT #1. IT ALSO COMPARES THE TRANSMITTED BIT STREAM
;AT THE TRANSMITTER WITH THE DECODED BIT STREAM AT THE
;RECEIVER. THE SYSTEM INTRODUCES A LARGE DELAY BETWEEN THE
;INPUT DATA AND THE OUTPUT DATA, SO COMPARISON IS DONE BY
;READING THE TRANSMITTED BIT STREAM AND STORING IT IN A
;CIRCULAR BUFFER, WHICH GIVES IT THE REQUIRED DELAY.

        move    x:bitsout,a0        ;Move the stored bit stream
                                    ;into a0.
        asr     a
        jcs     one                 ;The bit is shifted into the
                                    ;carry.
        bclr    #1,x:$ffe5          ;If the carry is set then
                                    ;output a 1 ,
        move    #0,x1               ;else output a zero.
        jmp     outbit              ;Skip if carry is set.
one     bset    #1,x:$ffe5
        move    #>$1,x1
        nop                         ;The nop's ensure equal
                                    ;timing for
        nop                         ;outputting a 1 or a 0.
outbit  move    a0,x:bitsout        ;Save shifted data stream.
        clr     a    #0,x0
        btst    #5,x:$ffe5          ;Test the data bit from the
                                    ;transmitter.
        jcc     delco               ;Jump if = 0.
        move    #>$1,a0
delco   move    a0,x:(r0)+          ;Store the transmitted bit
                                    ;in buffer.
        move    x:(r0),a            ;Move last stored element
                                    ;into register a for
                                    ;ouputting and comparison.
        cmp     x0,a
        jeq     rest0
        bset    #4,x:$ffe5          ;Output delayed bit as 1
        jmp     rest1
rest0   bclr    #4,x:$ffe5          ;or output delayed bit as 0.
```

```
rest1   cmp     x1,a                ;Compare the decoded bit
                                    ;stream (x1) to the
                                    ;transmitted bit stream (a).
        jeq     rest2
        bset    #6,x:$ffe5              ;If different, strobe bit
                                        ;6 in Port C.
rest2   move    x:a1save,a
        move    x:a0save,a0
        move    x:x0save,x0             ;restore saved registers
        move    x:x1save,x1
        move    x:y0save,y0
        move    x:y1save,y1
        bclr    #6,x:$ffe5              ;Clear bit 6.
        movep   #$c37,x:$ffff           ;re-enable interrupts
        rti

;IRQB INTERRUPT ROUTINE TO PERFORM CONVOLUTION WITH RECEIVER
;LOOKUP TABLE. THIS ROUTINE ALSO THRESHOLDS THE DEMODULATED
;SIGNAL AT THE SYMBOL SAMPLING INSTANTS AND DECODES THE
;RESULTING LEVEL.

calc    movep   #$c00,x:$ffff   ;Disable the IRQB interrupt.
        move    x:newbits,a0    ;Load new decoded bit stream
                                ;for output,
        move    a0,x:bitsout    ;transfer to output storage.
        move    x:oldvalI,a     ;Output I channel symbols to
                                ;DAC.
        move    #>$80,x0        ;Add dc offset required for
                                ;DAC.
        add     x0,a
        move    a,y:$8000       ;Clock the DAC latch.
        rep     #$30
        nop
        move    #$1000,r1       ;Disable the DAC latch.
        clr     b
        clr     a       x:(r1)+,x0  ;Load first filter value of
                                    ;the lookup table.
        move    r3,r2           ;Since r5 and r3 needed for
                                ;reading in samples,
        move    r5,r6           ;r6 and r2 must be used for the
                                ;convolution.
        move    x:-(r2),x1      ;Align to first value of
                                ;buffers.
        move    y:-(r6),y0
        move    y:(r6)+,y0      ;Load value from buffer
        movep   #$c07,x:$ffff   ;Re-enable /IRQA.

        ;CONVOLUTION LOOP

        do      #160,point
        mac     y0,x0,a          x:(r2)+,x1   y:(r6)+,y0
                                ;I channel
        mac     x1,x0,b          x:(r1)+,x0
                                ;Q channel

point   do      #3,shft
        asl     b               ;Scale for output.
        asl     a
```

```
shft
      move    a,x:oldvalI      ;Store filtered values.
      move    b,x:oldvalQ

;THIS SECTION THRESHOLDS THE FILTERED VALUE AND STORES THE
;CORRESPONDING BITS TO BE OUTPUT IN X:NEWBITS.
;THE CHIP OUTPUTS A BIT FOR EVERY 19.2KHZ INTERRUPT, BUT THE
;BITS IN X:NEWBITS ARE REPEATED TO GIVE THE CORRECT BIT
;RATE, IE. FOR 2400 BPS, ALL 8 BITS IN X:NEWBITS WILL BE THE
;SAME; FOR 4800 BPS, THE 4 BITS IN 2 GROUPS WILL BE THE
;SAME; FOR 9600 BPS, THE 2 BITS IN 4 GROUPS WILL BE THE
;SAME.

      move    #0,x0                   ;Default bits out are 0.
      move    x:bitrate,b0            ;Load bitrate.
      asr     b       x:oldvalI,a     ;Load I channel value, and
                                      ;move lower bit of bitrate
                                      ;into carry.
l_out jcs     h_out                   ;If bit 0 in bitrate is
                                      ;set then 9600 bps.
      asr     b
      jcs     m_out                   ;If bit 1 is set then 4800
                                      ;bps.
      abs     a       #>$31,y0        ;$31 is the threshold for
                                      ;3-level 2400.
      cmp     y0,a
      jgt     cont2                   ;Outer levels indicate 0
                                      ;was sent.
      move    #>$ff,x0                ;For lowest bit rate, all
                                      ;1's or 0's.
      jmp     cont2                   ;Skip 4800 and 9600 bps
                                      ;thresholding.

      ;4800 BPS ROUTINE - calculate for the I channel first.

m_out abs     a       #>$2a,y0        ;Threshold for 4800 bps is
                                      ;$2a.
      cmp     y0,a
      jgt     moutQ
      move    #>$0f,x0                ;Outer levels indicates 1.

      ;If necessary, calculate for the Q channel.

moutQ move    x:oldvalQ,a             ;Msb 4 bits for Q channel,
                                      ;lsb bits for I.
      abs     a
      cmp     y0,a
      jgt     cont2
      move    #>$f0,a1                ;Inner level indicates 0.
      or      x0,a                    ;Combine I and Q channel
                                      ;decoded bits.
      move    a,x0
      jmp     cont2
```

```
        ;9600 BPS ROUTINE - calculate for the I channel first.

h_out   move    #>$52,y0
        cmp     y0,a        #>$31,y1
        jlt     Ihi1
        move    #>$03,x0            ;Highest level (6) gives
                                    ;10, store 0011.
        jmp     Qcalc
Ihi1    cmp     y1,a        #>$10,y0
        jlt     Ihi2
        move    #>$0c,x0            ;Level 5 gives 01, store 1100.
        jmp     Qcalc
Ihi2    cmp     y0,a        #$fffff0,y1
        jlt     Ihi3
        move    #0,x0               ;Level 4 gives 00, store 0000.
        jmp     Qcalc
Ihi3    cmp     y1,a        #$ffffcf,y0
        jlt     Ihi4
        move    #>$0f,x0            ;Level 3 gives 11, store 1111.
        jmp     Qcalc
Ihi4    cmp     y0,a        #$ffffae,y1
        jlt     Ihi5
        move    #>$03,x0            ;Level 2 gives 10, store 0011.
        jmp     Qcalc
Ihi5    cmp     y1,a
        jlt     Ihi6
        move    #>$0c,x0            ;Level 1 gives 01, store 1100.
        jmp     Qcalc
Ihi6    move    #0,x0

        ;Now calculate for the Q channel.

Qcalc
        move    x:oldvalQ,a
        move    #>$52,y0
        cmp     y0,a        #>$31,y1
        jlt     Qhi1
        move    #>$30,x1            ;Highest level (6) gives 10,
                                    ;store 0011.
        jmp     IQcalc
Qhi1    cmp     y1,a        #>$10,y0
        jlt     Qhi2
        move    #>$c0,x1            ;Level 5 gives 01, store 1100.
        jmp     IQcalc
Qhi2    cmp     y0,a        #$fffff0,y1
        jlt     Qhi3
        move    #0,x1               ;Level 4 gives 00, store 0000.
        jmp     IQcalc
Qhi3    cmp     y1,a        #$ffffcf,y0
        jlt     Qhi4
        move    #>$f0,x1            ;Level 3 gives 11, store 1111.
        jmp     IQcalc
Qhi4    cmp     y0,a        #$ffffae,y1
        jlt     Qhi5
        move    #>$30,x1            ;Level 2 gives 10, store 0011.
        jmp     IQcalc
```

```
Qhi5  cmp     y1,a
      jlt     Qhi6
      move    #>$c0,x1          ;Level 1 gives 01, store 1100.
      jmp     IQcalc
Qhi6  move    #0,x1

IQcalc                          ;Combine the decoded I and Q bits.
      move    x1,a
      or      x0,a
      move    a,x0
cont2
      move    x0,x:newbits        ;Store the new decoded bits.

      movep   #$c37,x:$ffff       ;Re-enable /IRQB.
      rti

;SUBROUTINE TO READ IN LOOKUP TABLE

table move    #$1000,r1             ;Starting address of
                                    ;lookup table.
      move    #159,m1               ;Uses external memory.

      do      #160,endin        ;Loop to read in lookup table.
bitin jclr    #0,x:$ffe9,bitin      ;Wait for valid data,
                                    ;then read from host
                                    ;interface.
      move    x:$ffeb,x0
      move    x0,x:(r1)+
endin move    #$1000,r1

;CLEAR THE CIRCULAR BUFFER

      move    #$0,r6            ;Demodulated I channel buffer
                                ;start address.
      move    #$0,r5
      move    #$0,r3            ;Demodulated Q channel buffer
                                ;start address.
      move    #buffmod,m6
      move    #buffmod,m5
      move    #buffmod,m3
      move    #buffmod,m2
      move    #0,y0
      do      #lkpleng,clear    ;Clear the circular buffers.
      move    y0,y:(r6)+
      move    y0,x:(r3)+
clear rts

;INTERRUPT ROUTINES TO STORE BITRATE PARAMETERS.

low   move    #0,a0             ;2400 bits per sec.
      move    a0,x:bitrate
      rti
mid   move    #>$2,a0           ;4800 bits per sec.
      move    a0,x:bitrate
      rti
high  move    #>$3,a0           ;9600 bits per sec.
      move    a0,x:bitrate
      rti
```

```
;INTERRUPT ROUTINES TO CHANGE THE LENGTH OF THE DATA-
;COMPARING BUFFER.

shftu                       ;Increase the length of the buffer.
      move    m0,a
      move    #>$1,x0
      add     x0,a
      move    a,m0
      rti

shftd move    m0,a          ;Decrease the length of the buffer.
      move    #>$1,x0
      sub     x0,a
      move    a,m0
      rti

;END OF PROGRAM.

      end→
```

APPENDIX E. REAL-TIME HOST PASCAL PROGRAM LISTING 'TRANSMIT.ASM'

The following program listing is for the real-time Pascal host program.

```
program transmit;

(*-------------------------------------------------------------*)
(* This is the host control program for the transmitter   *)
(* and receiver DSP boards. The program downloads low-    *)
(* pass filter lookup tables to the two microprocessor    *)
(* chips, controls the bit rate (2400, 4800 or 9600 bps ),*)
(* the phase of the 19.2 kHz clock, and the length of the *)
(* delay buffer at the receiver.                          *)
(* This program must be used with TRANSMIT.ASM and        *)
(* RECEIVE.ASM for real-time operation.                   *)
(*-------------------------------------------------------------*)

uses
 dos, crt, dsplib2;

type
  store1 = array[1..512] of longint;
           (* array for lookup table *)

var j,k:byte;
    infile:text;
    i:integer;
    baud,num24:longint;
    rate_select:real;
    tbaseaddress,rbaseaddress,timeout:word;
    Tx_points,Rx_points:store1;
    yesno:char;
    quit,load:boolean;

Procedure Level_transmit;

(*************************************************************)
(* Provides a 1 second delay for the receiver to read the *)
(* zero level transmitted by the transmitter.             *)
(*************************************************************)

var testchar,ch:char;

begin
  writeln('Transmitting zero level');
  delay(1000);
end;
```

```
Procedure Read_from_file;

(*********************************************************)
(* This procedure reads the low-pass filter lookup table  *)
(* into turbo.                                            *)
(*********************************************************)

var
  a,i:integer;
  temp:real;
  scale:longint;

begin
  writeln('Reading files into turbo..');
  assign(infile,'35216_01.dec');
  reset(infile);
                        (* Read in 352 point root raised *)
                        (* cosine filter response.       *)
  i:=1;
  for a := 1 to 352 do
  begin
      readln(infile,temp);
      Tx_points[a] := round(temp*524280);
      if (a mod 2) = 1 then       (* Read every 2nd point *)
      begin                                (* for receiver. *)
        Rx_points[i] := round(temp*200000);
        i:=i+1;
      end;
    end;
  close(infile);
end;

Procedure Lookup_to_transmitter;

(*****************************************************)
(* This procedure downloads the lookup table to the  *)
(* transmitter chip and sets the bit rate.           *)
(*****************************************************)

begin
  writeln('enter bit rate ( 9600, 4800, 2400 ): ');
  read(rate_select);
  baud := 4;
  if rate_select = 4800 then baud := 2;
                              (* Set option for bit rate. *)
  if rate_select = 2400 then baud := 1;
  DSP_writeflag(tbaseaddress,timeout);
                          (* Wait until DSP chip is ready. *)
  DSP_writelongint(tbaseaddress,baud);
                         (* Download option to transmitter. *)
  writeln('downloading lookup table to transmitter DSP..');
    for i := 1 to 352 do
    begin
      DSP_writeflag(tbaseaddress,timeout);
      DSP_writelongint(tbaseaddress,Tx_points[i]);
    end;
end;
```

```
Procedure lookup_to_receiver;

(************************************************)
(* Downloads the lookup table to the receiver *)
(************************************************)

begin
  writeln('downloading lookup table to receiver DSP..');
    for i := 1 to 176 do
    begin
      DSP_writeflag(rbaseaddress,timeout);
      DSP_writelongint(rbaseaddress,Rx_points[i]);
    end;
end;

Procedure  bit_rate_change;

(*********************************************************)
(* This procedure changes the bit rate of the system.  *)
(* This is done by interrupting the receiver and       *)
(* transmitter microprocessors, which reset certain    *)
(* parameters. This procedure also allows changing of  *)
(* the length of the data delay buffer that is used    *)
(* to compare the transmitted and decoded data         *)
(* streams, and changing the phase of the 19.2 kHz     *)
(* clock signal at the transmitter.                    *)
(*********************************************************)

var
   hostvector:word;

          (* 'hostvector' is the program location *)
          (* the DSP chip jumps to for interrupt  *)
          (* execution.                           *)

   ch:char;
   a:integer;
   count:integer;

begin
   ch := readkey;
   case ch of
            '1':begin                          (* 2400 bps *)
                  hostvector := $24;
                  DSP_vectorint(tbaseaddress,hostvector);
                  DSP_vectorint(rbaseaddress,hostvector);
                end;
             'm':begin                         (* 4800 bps *)
                  hostvector := $26;
                  DSP_vectorint(tbaseaddress,hostvector);
                  DSP_vectorint(rbaseaddress,hostvector);
                end;
             'h':begin                         (* 9600 bps *)
                  hostvector := $28;
                  DSP_vectorint(tbaseaddress,hostvector);
                  DSP_vectorint(rbaseaddress,hostvector);
                end;
```

```
                'u':begin
                          (* Increases the length of the   *)
                          (* delay buffer in the receiver. *)
                      hostvector:=$2a;
                      DSP_vectorint(rbaseaddress,hostvector);
                    end;
                'd':begin
                           (* Decreases the length of the   *)
                           (* delay buffer in the receiver. *)
                      hostvector:=$2c;
                      DSP_vectorint(rbaseaddress,hostvector);
                    end;
                'p':begin
                (* Changes the phase of the 19.2khz signal. *)
                      hostvector := $2a;
                      DSP_vectorint(tbaseaddress,hostvector);
                    end;
                'q':quit := true;
        end;
end;

(*****************)
(* MAIN PROGRAM *)
(*****************)

begin
  tBaseAddress:=$2b0;       (* Transmitter base address *)
  rbaseaddress:=$2b8;         (* Receiver base address *)
  port[tbaseaddress]:=port[tbaseaddress] or  $8;
                  (* Set bit for zero level transmission *)
  port[rbaseaddress]:=port[rbaseaddress] and $f7;
  DSP_Go(tBaseAddress);
                     (* Start transmission of zero level *)
  quit := false;
  load := false;
  writeln('starting');          (* Start operation *)
  timeout:=100;
  level_transmit;
  DSP_go(rbaseaddress);                    (* Start receiver *)
  read_from_file;          (* Read lookup table into array *)
  lookup_to_receiver;        (* Download table to receiver *)
  delay(1000);    (* Delay for receiver to read zero level *)
  port[tbaseaddress]:=port[tbaseaddress] and $f7;
                            (* Start normal transmission *)
  lookup_to_transmitter;
                         (* Download table to transmitter *)
  writeln('press ''h'' to change to 9600 bps,
                ''m'' for 4800 bps, and ''l'' for 2400 bps');
  writeln('       (''q'' to quit, ''p'' to change the phase
                                   of the 19.2 kHz clock )');
   repeat
     bit_rate_change;
   until quit;
end.→
```

APPENDIX F. TRANSMITTER SIMULATION PROGRAM LISTINGS 'TXIQ.PAS' AND 'TXIQ.ASM'

This appendix contains listings of two simulation programs. The programs (in order) are:-

'TXIQ.PAS' - transmitter Pascal program.
'TXIQ.ASM' - transmitter assembler program.

```
program txiq;

(*-------------------------------------------------------*)
(* This program downloads a lookup table to the          *)
(* transmitter DSP56001 and sends a random 9600 bps data  *)
(* stream to the chip. The chip encodes and modulates the *)
(* stream to give a 49-QPRS output. The DSP chip is       *)
(* interrupt-driven, reading in data bits every 4         *)
(* interrupts and outputting a value every interrupt. This*)
(* program must be used with TXIQ.ASM. This is a          *)
(* simulation program, and the result of the convolution  *)
(* and modulation is stored in a file called RESULT.DEC.  *)
(*-------------------------------------------------------*)

uses
 dos, crt, GDriver, GKernel, printer, dsplib2;

type
  store1 = array[1..512] of longint;
                              (* Lookup table *)
  store2 = array[1..2048] of longint;
                              (* Input data stream *)
  store3 = array[1..4096] of real;
                              (* Result of convolution *)

var j,k:byte;
    infile:text;
    i:integer;
    num24:longint;
    baseaddress,timeout:word;
    lookup_table:store1;
    bit_stream:store2;
    signal:store3;
```

```
Procedure Read_from_file;

(*********************************************)
(* This procedure reads the lookup table and *)
(* the data stream into turbo.               *)
(*********************************************)

var
  a:integer;
  temp:real;      (* Temporary store before scaling *)
  scale:longint;

begin
  writeln('Reading files into turbo..');
  assign(infile,'32016_02.dec');
    (* Use a 320 point root raised-cosine low-pass filter*)
  reset(infile);
  for a := 1 to 320 do
    begin
      readln(infile,temp);
      lookup_table[a] := round(temp*40000 );
    end;                (* Read in the lookup table values *)
  close(infile);
  assign(infile,'data.io');
                (* This file contains random 1's and 0's. *)
  reset(infile);
  for a := 1 to 1024 do readln(infile,bit_stream[a]);
  close(infile);
end;


Procedure writePoints;

(*************************************)
(* This procedure downloads the lookup *)
(* table to the DSP chip.              *)
(*************************************)

begin
  writeln('downloading lookup table to DSP..');
    for i := 1 to 320 do
    begin
      DSP_writeflag(baseaddress,timeout);
      DSP_writelongint(baseaddress,lookup_table[i]);
    end;
end;
```

```
Procedure TxRxpoints;

(***********************************************)
(* This procedure interrupts the DSP and       *)
(* reads and writes data to/from the chip      *)
(***********************************************)

var
  a,l:integer;
  hostvector,digit:byte;

begin
  writeln('sending/receiving data to/from DSP..');
  hostvector := $24;
              (* Host interrupt vector location *)
    for a := 1 to 4096 do
    begin
        l := round(int(1 + (a-1)/4));
             (* Increments every 4 interrupts *)
        digit := bit_stream[l];
             (* An interrupt occurs 4 times for each bit. *)
        DSP_writelongint(baseaddress,digit);
        DSP_vectorint(baseaddress,hostvector);
        DSP_readflag(baseaddress,timeout);
        DSP_readlongint(baseaddress,num24);
        signal[a] := num24/50000;
             (* Read output from transmitter, scale down *)
    end;
end;

Procedure Write_to_file;

(*************************************)
(* Stores result of convolution and *)
(* modulation in RESULT.DEC.        *)
(*************************************)

var
  a:integer;

begin
  writeln('storing result in RESULT.DEC..');
  assign(infile,'result.dec');
  rewrite(infile);
  for a := 1 to 4096 do writeln(infile,signal[a]);
  close(infile);
end;


begin
  BaseAddress:=$2b0;
  DSP_Go(BaseAddress);          (* Start DSP chip operation *)
  timeout:=100;
  read_from_file;    (* Read lookup table, bits from files *)
  writepoints;             (* Download lookup table to chip *)
  TxRxPoints;  (* Transmit bits, store modulated waveform *)
  write_to_file;                    (* Write result to file *)
end .→
```

```
;SIMULATION PROGRAM TO CONVOLVE INPUT DATA STREAM WITH
;LOOKUP TABLE UNDER HOST INTERRUPT CONTROL. THE MODULATED I
;AND Q CHANNELS ARE CALCULATED, AND THERESULT IS WRITTEN
;BACK TO THE HOST.
;
;r0-lookup table pointer
;r1-IRQA counter
;r2-I channel carrier pointer
;r3-tone lookup table pointer
;r4-buffer pointer
;r6-Q channel carrier pointer

     opt  fc,mu,s,w,mex

buffoff      equ   60          ;buffer offset address
location
buffmod      equ   63          ;buffer modulus
lkpleng      equ   512         ;lookup table length
lkpmod       equ   511         ;lookup table modulus(0-512)
lkpoff       equ   16          ;lookup table offset(0-16)
points       equ   16          ;points per symbol
symbols      equ   32         ;symbols in lookup table
pastI        equ   $050        ;temporary store for past I
                               ;duobinary value.
pastQ        equ   $051        ;temporary store for past Q
                               ;duobinary value.
four         equ   $052        ;offset of four in encoding
                               ;calculations.
modulus      equ   $053        ;modulus of 3, and level
                               ;shift for encoding.
scale        equ   $054        ;scaling factor for tone
store        equ   $055        ;store for input bits


     org  p:$0000
     jmp  begin1
     org     p:$0024          ;host vector interrupt location.
     jsr     calc

     org  p:$0100
begin1
      movec  #6,omr            ;Enable roms.
      move   #0,x0
      move   x0,y:pastI        ;Temp store for past encoded
                               ;I number.
      move   x0,y:pastQ        ;Temp store for past encoded
                               ;Q number.
      move   x0,y:store        ;Store for bits as they are
                               ;input.
```

```
        move    #4,a0
        move    a0,y:four
        move    #3,a0
        move    a0,y:modulus
        move    #4096,a0
        move    a0,y:scale
        clr     a
        move    #$100,r3            ;Pilot tone pointer.
        move    #4,n3               ;Read every 4th point.
        move    #$ff,m3
        move    #$100,r2            ;I carrier lookup table
                                    ;address.
        move    #12,n2              ;Read every 12th point for
                                    ;carrier.
        move    #$ff,m2
        move    #$140,r6            ;Q carrier offset by 90 deg.
        move    #12,n6
        move    #$ff,m6

;CLEAR THE CIRCULAR BUFFER
;this buffer contains both the I and Q values

        move    #$00,r4
        move    #buffmod,m4
        move    #buffoff,n4
        move    #0,y0
        do      #64,clear
        move    y0,y:(r4)+
clear

;READ IN THE LOOKUP TABLE

        move    #$1000,r0               ;Starting address of
                                        ;lookup table.
        move    #lkpmod,m0
        move    #lkpoff,n0


        do      #lkpleng,endin  ;Loop to read in lookup table.
bitin   btst    #0,x:$ffe9      ;Wait for valid data.
        jcc     bitin
        move    x:$ffeb,x0
        move    x0,x:(r0)+      ;Store in external X memory.
endin

;SET POINTERS

        move    #$1000,r0
        move    #$04,r4     ;Intialise the buffer start address.
        move    #$0,r1
        move    #$f,m1
        clr     a
        move    a,x0
        move    a,y0
```

```
;ENABLE INTERRUPTS

        move    #$0c00,a0
        move    a0,x:$ffff      ;Set interrupt priority register.
        move    #0,a0
        move    a0,sr           ;Set interrupt mask.
        bset    #2,x:$ffe8      ;Enable the host interrupt.

;MAIN PROGRAM LOOP

main    nop
        nop
        jmp main

;INTERRUPT ROUTINE TO PERFORM CONVOLUTION

calc    jsr     count                   ;Read in bit?
        clr     a       x:(r0)+n0,x1    ;Load lookup table value.
        clr     b

        do      #symbols,point

        mac     x1,x0,a         y:(r4)+,x0
        mac     x1,y0,b         x:(r0)+n0,x1    y:(r4)+,y0
point
        move    x:(r0)-n0,x1            ;Bogus moves to set
                                        ;lookup pointer.
        move    x:(r0)+,x1              ;
        asr     a
        move    a0,x1                   ;x0 and y0 are preserved
        clr     a    y:(r2)+n2,y1
        mpy     x1,y1,a                 ;Modulate I channel.
        asr     b
        asr     b
        move    b0,x1
        clr     b    y:(r6)+n6,y1
        mpy     x1,y1,b         ;Modulate Q channel.
        add     a,b             ;Add modulated I and Q channels.
                                ;This results in a 49-QPRS signal.
;ADD THE PILOT TONE

        move    y:scale,x1
        move    y:(r3)+n3,y1
        clr     a
        mpy     y1,x1,a         ;Scale down.
        add     a,b             ;Add tone to 49-QPRS signal.

out     btst    #1,x:$ffe9
        jcc     out
        move    a1,x:$ffeb
        jsr     sym             ;Calculate symbol?
        rti
```

```
;SUBROUTINE TO COUNT INTERRUPTS
;a bit must be read in every 4th interrupt

count move   x:(r1)+,a      ;Increment counter (bogus move).
      move   r1,x1          ;load value of counter.
      move   #3,a1
      and    x1,a           ;Check if 2 LSB's are clear.
(multiple of 4)
      jseq   read           ;If so, read in bit.
      clr    a
      rts

;SUBROUTINE TO READ IN BIT
;stores bit in location y:$055 with past bits

read  btst   #0,x:$ffe9
      jcc    read
      move   x:$ffeb,b       ;Read in new bit from host.
      move   y:store,a       ;Load past bits (if any).
      addl   b,a             ;Combine bits.
      move   #$f,b1
      move   b1,x1
      and    x1,a            ;Remove unnecessary bits,
      move   a,y:store       ;and store.
      rts

;SUBROUTINE TO CHECK IF NEW SYMBOLS MUST BE CALCULATED
;new symbol calculated every 16 interrupts

sym   move   r1,x1           ;If r1 = 0, then code
      clr    a               ;new bits.
      or     x1,a
      jseq   code
      rts

;SUBROUTINE TO ENCODE DATA AND STORE IN BUFFER

code  move   y:(r4)+n4,x1    ;Bogus move to increment
                             ;pointer.
      move   #$1000,r0       ;Reset lookup table pointer.
      move   y:store,a       ;Load 4 bits into a.
      move   y:modulus,x0
      asr    a
      asr    a               ;Shift and isolate 2 upper bits.
      and    x0,a
      move   y:store,b
      and    x0,b            ;Isolate 2 lower bits
      move   y:four,y0
      add    y0,a     y:pastI,x1   ;Add offset for
                                   ;calculations.
      add    y0,b     y:pastQ,y1
      sub    x1,a             ;Subtract previous symbol.
      sub    y1,b
      and    x0,a             ;Take modulo-4.
      and    x0,b     a1,y:pastI

      add    x1,a     b1,y:pastQ
      add    y1,b
```

```
sub     x0,a        ;Subtract offset of 3.
sub     x0,b        a1,y:(r4)+  ;Store coded values in
                                ;buffer.
move    a1,x0
move    b1,y:(r4)+
move    b1,y0
rts

end
```

APPENDIX G. RECEIVER SIMULATION PROGRAM LISTINGS 'RXIQ.PAS' AND 'RXIQ.ASM'

This appendix contains two program listings. They are:-

'RXIQ.PAS' - receiver Pascal simulation program.

'RXIQ.ASM' - receiver assembler simulation program.

```
program rxiq;

(*-------------------------------------------------------*)
(* This program downloads the filter lookup table to the  *)
(* receiver DSP56001. It then writes the 49-QPRS signal   *)
(* generated by the transmitter to the receiver chip. The *)
(* chip demodulates and LPF's the signal to give a 7 level*)
(* output. The sampling rate of the receiver is 19.2 kHz, *)
(* half that of the transmitter. The lookup table is      *)
(* therefore half the length of that used in the          *)
(* transmitter, and the receiver will sample only every   *)
(* 2nd point from the stored transmitted waveform. This   *)
(* program must be used with RXIQ.ASM. The result of the  *)
(* demodulation is stored in DEMOD.DEC.                   *)
(*-------------------------------------------------------*)

uses
 dos, crt, GDriver, printer, GKernel, dsplib2;

type
  store1 = array[1..512] of longint;
                                (* Lookup table *)
  store2 = array[1..2048] of longint;
                                (* Input data stream *)
  store3 = array[1..2048] of real;
                              (* Result of convolution *)

var j,k:byte;
    infile,infile1,infile2:text;
    i:integer;
    num24:longint;
    baseaddress,timeout:word;
    lookup_table:store1;
    Tx_output:store2;
    result:store3;
    offset:longint;
```

```
Procedure Read_from_file;

(*********************************************************)
(* This procedure reads the low-pass filter lookup       *)
(* table into an array, as well as the 49-QPRS  signal   *)
(* generated by the transmitter.                         *)
(*********************************************************)

var
  a:integer;
  scale,temp:real;
  max:longint;

begin
  writeln('Reading files into turbo..');
  assign(infile,'32016_02.dec');
              (* 320 point root raised-cosine filter taps *)
  reset(infile);
  for a := 1 to 320 do
    begin
      readln(infile,temp);
                  (* Read every 2nd value for 160 points *)
      lookup_table[a] := round(temp*2000000 );
      readln(infile,temp);
    end;
  close(infile);
  assign(infile,'result.dec');
        (* Now load modulated waveform from transmitter *)
  reset(infile);
  offset:=0;
  max:=0;
  for a := 1 to 2048 do
    begin
      readln(infile,scale);       (* Read every 2nd point *)
      Tx_output[a] := round(scale*400000);

      if abs(Tx_output[a])>max then
                    max := abs(Tx_output[a]);
            (* Store the maximum value in the file *)

      if Tx_output[a]<offset then offset := Tx_output[a];
      readln(infile,scale);
    end;
  close(infile);

(* The following few lines convert the transmitted a.c. *)
(* signal to a signal with a d.c. offset and maximum    *)
(* value $ff. This is done to simulate the action of an *)
(* 8-bit a/d chip at the input of the receiver.         *)

  offset:=round(-1*offset*128 div max);
  for a := 1 to 2048 do
    begin
     Tx_output[a] := round(Tx_output[a]*128 div max)+offset;
    end;
end;
```

```
Procedure writePoints;

(****************************************)
(* This procedure downloads the lookup *)
(* table to the receiver DSP           *)
(****************************************)

begin
  write('downloading lookup table to DSP..');
    for i := 1 to 160 do
    begin
      DSP_writeflag(baseaddress,timeout);
      DSP_writelongint(baseaddress,lookup_table[i]);
    end;
end;


Procedure TxRxpoints;

(*************************************************************)
(* This procedure sends the transmitted signal values to *)
(* the receiver and reads back the demodulated and low-  *)
(* pass filtered result.                                 *)
(*************************************************************)

var
  a,k,l,m:integer;
  hostvector:byte;

begin
  writeln('sending/receiving data to/from DSP..');
   a:=1;
    for i := 1 to 2048 do
    begin
        num24 := Tx_output[i];
        DSP_writeflag(baseaddress,timeout);
        DSP_writelongint(baseaddress,num24);
                        (* write points to receiver *)
        if (i mod 8)= 0 then  (* After every 8 samples, a *)
                              (* symbol value must be     *)
                              (* calculated by the DSP.   *)
        begin
          DSP_readflag(baseaddress,timeout);
          DSP_readlongint(baseaddress,num24);
          result[a] := num24;
          a:=a+1;     (* Read the calculated symbol value *)
        end;
    end;
end;
```

```
procedure Write_to_file;

(***********************************************)
(* Stores result of demodulation in DEMOD.DEC *)
(***********************************************)

var
  a:integer;

begin
  writeln('storing result in DEMOD.DEC..');
  assign(infile1,'demod.dec');
  rewrite(infile1);
  for a := 1 to 2048 do
    begin
      writeln(infile1,result[a]);
    end;
  close(infile1);
end;

(****************)
(* MAIN PROGRAM *)
(****************)

begin
  BaseAddress:=$2b0;
  DSP_Go(BaseAddress);              (* Start chip operation *)
  timeout:=100;
   read_from_file;
      (* Read lookup table, transmitted signal from files *)
   writepoints;                  (* Download lookup table *)
   DSP_writeflag(baseaddress,timeout);
   DSP_writelongint(baseaddress,offset);
                            (* 'Transmit' the d.c. offset *)
   TxRxPoints;             (* Transmit and receive points *)
   write_to_file;           (* Store result *)
end .→
```

```
;SIMULATION PROGRAM TO DEMODULATE THE TRANSMITTED SIGNAL.
;THE DEMODULATED I CHANNEL IS CALCULATED.
;THIS PROGRAM MUST BE USED WITH 'RXIQ.PAS'.
;
;r0- low-pass filter lookup table pointer.
;r1- sampled data input buffer pointer.
;r2- buffer pointer for mac routine calculation.
;r3- I carrier pointer.

     opt  fc,mu,s,w,mex

buffmod      equ   159     ;Input buffer modulus.
lkpleng      equ   160     ;Lookup table length.
lkpmod       equ   159     ;Lookup table modulus.
points       equ   8       ;Points per symbol.
symbols      equ   20      ;Symbols in lookup table.
offset       equ   $00     ;Memory location to store the
                           ;simulated dc offset of the ADC
                           ;chip.
     org  p:$0000
     jmp  begin1

     org  p:$0100
begin1

      movec  #6,omr          ;Enable on-board roms.
      move   #$100,r3        ;I carrier lookup table address.
      move   #24,n3          ;Twice the offset as the
                             ;transmitter.
      move   #$ff,m3

;CLEAR THE CIRCULAR BUFFER

      move   #$0,r1          ;Start address of the buffer.
      move   #buffmod,m1
      move   #buffmod,m2
      move   #0,x0
      do     #lkpleng,clear
      move   x0,x:(r1)+      ;Clearing loop.
clear

;READ IN LOOKUP TABLE

      move   #$1000,r0    ;Starting address of lookup table.
      move   #lkpmod,m0

      do     #lkpleng,endin   ;Loop to read in lookup table.
bitin jclr   #0,x:$ffe9,bitin ;Wait for valid data.
      move   x:$ffeb,x0
      move   x0,x:(r0)+       ;Store value in X memory.
endin
      move   #$00,r1          ;Intialise the buffer start
                              ;address.
      move   #$1000,r0
```

```
;READ IN D.C OFFSET

dcin  jclr   #0,x:$ffe9,dcin
      move   x:$ffeb,x0
      move   x0,y:offset          ;Store value of offset in
                                  ;on-board RAM.

;MAIN PROGRAM LOOP FOLLOWS


;ROUTINE TO READ IN DATA AND DEMODULATE


data  do     #8,readin            ;Read in 8 values, then
                                  ;calculate a symbol.
in    jclr   #0,x:$ffe9,in
      move   x:$ffeb,a            ;Read in the value.
      move   y:offset,y0
      sub    y0,a                 ;Subtract the dc offset.
      move   a,x0
      move   y:(r3)+n3,y0
      mpy    x0,y0,a              ;Multiply by I carrier.
      move   a1,x:(r1)+           ;Store demodulated value
                                  ;in the buffer.
readin
      move   #$1000,r0

;ROUTINE TO PERFORM CONVOLUTION WITH RECEIVER LOOKUP TABLE

calc  move   r1,r2          ;r2 used for the convolution loop.
      move   x:(r0)+,x0     ;Load first lookup table value.
      clr    a
      move   x:(r2)+,y0     ;Load first sample value.

      do     #lkpleng,point  ;Convolution loop.
      mac    y0,x0,a          x:(r2)+,y0
      move   x:(r0)+,x0
point nop

out   jclr   #1,x:$ffe9,out  ;Write calculated symbol value
                             ;to Host.
      move   a1,x:$ffeb
      jmp    data            ;Return to beginning and
                             ;repeat.

      end
```

APPENDIX H. FILTER GENERATOR PROGRAM LISTING 'SIMPSON.PAS'

This appendix contains a program listing of 'Simpson.pas'. This program generates raised-cosine and square-root raised-cosine lookup tables.

```
program Simpson;

(*************************************************************)
(* This program generates raised-cosine and root raised-     *)
(* cosine filter impulse responses. The first part           *)
(* computes the raised-cosine impulse response from an       *)
(* explicit formula. The filter width, in symbols, the       *)
(* points per symbol, and the value of α are all variable.   *)
(* The 2nd part uses Simpson's rule to do the inverse        *)
(* Fourier transform on a root raised-cosine filter          *)
(* frequency response. The resulting time impulse response   *)
(* is to be real-valued, so only the cosine term in the      *)
(* transform is used. Simpson's rule states that for a       *)
(* continuous function f on [a,b] partitioned into n         *)
(* divisions where n is even, ie values of f at divisions    *)
(* are a=x0,x1....xn=b, then                                 *)
(*    ⌠b                                                     *)
(*    │  f(x)dx ≈ (b-a)/3n [f(x0)+4f(x1)+2f(x2)+4f(x3)...    *)
(*    ⌡a                                  +4f(xn-1)+f(xn)]. *)
(*                                                           *)
(*************************************************************)

uses
  crt,dos;

type
  point = array[0..1024] of real;
                                  (* Stores the lookup table *)
var
  n,m,i,j:integer;
  zero_one:word;          (* Alternates between 0 and 1. *)
  two_four:word;
           (* Uses zero_one to alternate between 2 and 4. *)
  numpoints:longint;
             (* Number of points in the frequency domain. *)
  Firstval,Lastval:real;
                   (* Endpoints of the frequency function. *)
  alpha:real;
          (* α of the raised cosine filter freq. response. *)
  sum:real;
        (* Integral of frequency values used in transform *)
  t,H,Y,freq:real;
(* t=time, Y=frequency function, H=width of
                     frequency divisions, freq=frequency.  *)
  choice:integer;
                     (* Choose option for filter response. *)
```

```
  symbols:integer;
                    (* No. of symbols in the lookup table. *)
  lookupsize:integer;
                     (* No. of points in the lookup table. *)
  points_per_sym:integer;
  outfile:text;
  taps:point;         (* Store the raised-cosine response. *)
  points:point;
               (* Stores the root raised-cosine response. *)
  filestring:string;        (* Stores the filename. *)


Procedure Generate_Raised_Cosine;

(*****************************************************)
(* This procedure generates a normal raised-cosine   *)
(* filter response of variable length and resolution. *)
(*****************************************************)

var step:real;
    tempconst:real;

begin
  write('enter symbol width of filter:');
  readln(symbols);
  write('enter points per symbol:');
  readln(points_per_sym);
  write('enter excess bandwidth coefficient α:');
  readln(alpha);
  write('Save as:');
  readln(filestring);
  Writeln;
  writeln;
  write('      Generating Raised-Cosine Impulse response');
  step:=pi/points_per_sym;  (* Time increment step size. *)
  numpoints:=points_per_sym * symbols div 2;
       (* Number of points to calculate. *)
  taps[0]:=1;                              (* At t=0, f(t)=1. *)
  for i:=1 to numpoints do

    taps[i]:=( sin(i*step)/(i*step) )
          * ( cos(alpha*i*step)/(1-sqr(2*alpha*i*step/pi)));

                      (* Calculate the impulse values   *)
                      (* for half the table only, since *)
                      (* table is symmetrical.          *)

  assign(outfile,filestring);
  rewrite(outfile);
  for i:=numpoints downto 0 do
                     (* Store the response for -t. *)
  writeln(outfile,taps[i]);
  for i:=1 to numpoints-1 do
                     (* Store the response for +t. *)
  writeln(outfile,taps[i]);
  close(outfile);
end;
```

```
Procedure Calculate_impulse_response_point;

(*********************************************************)
(* This procedure calculates the impulse response value  *)
(* at one time point. This requires an integration over  *)
(* the entire frequency interval.                        *)
(*********************************************************)

begin
  zero_one := 0; (* Initialize to 0. *)
  sum := 0;
  For n := 1 to numpoints-1 do
                    (* Exclude the first and last points *)
                    (* since they both equal zero.       *)
    begin
      zero_one := (zero_one + 1) mod 2;
            (* Alternates between 0 and 1. *)
      two_four := zero_one*2 + 2;
            (* Alternates between 2 and 4. *)

      if abs(freq) <= (1-alpha) then
        Y := sqrt(0.5) * cos(freq * t)

    (* This is true for the flat centre region *)
    (* of the frequency transfer function.     *)

      else if ((1-alpha) < abs(freq)) and ((1+alpha) >
abs(freq)) then
        Y := sqrt(0.25 * (1-sin(pi * (abs(freq)-
             1)/2/alpha))) * cos(freq * t)

    (* This is true for the cosine-shaped region. *)

      else Y := 0;

      sum := sum + Y*two_four;
         (* Add result to running total. *)
      Freq := Freq + H;
         (* Increment to next frequency value. *)
    end;
    sum := sum * H/3;
         (* Scale correctly as per Simpson's rule. *)
end;
```

```
Procedure Generate_Root_Raised_Cosine;

(*****************************************************)
(* This procedure calculates the root raised-cosine *)
(* impulse response for various length filters.     *)
(*****************************************************)


  begin
    write('Enter symbol width of filter:');
    readln(symbols);
    symbols:=symbols div 2;
    write('Enter number of points per symbol:');
    readln(points_per_sym);
    write('Enter excess bandwidth coefficient α:');
    readln(alpha);
    write('Save as:');
    readln(filestring);
    Writeln;
    writeln;
    write('      Generating Root Raised-Cosine Impulse
response');
    lookupsize:=symbols * points_per_sym;
    assign(outfile,filestring);
        (* Store lookup table in RRCF.PAS. *)
    rewrite(outfile);
    Firstval := -(1 + alpha);
        (* Extremities of the filter *)
    Lastval := 1 + alpha;
        (* frequency response. *)
    Numpoints := 320;
        (* Number of points in frequency domain. *)
        (* Start with calculation for t=0. *)
    H := (lastval-firstval)/(numpoints);
        (* Width of freq. divisions. *)
    For m := 0 to lookupsize do
        (* Calculate 160 response values. *)
    begin
      GoToXY(5,10);
      writeln('Calculating point number: ',m);
      t := m/points_per_sym * pi;
      Freq :=Firstval + H;
         (* First freq. value to use in calculation. *)
      Calculate_impulse_response_point;
         (* Calculate the time response value. *)
      points[lookupsize-m] := sum; (* Store result. *)
    end;
    for m :=0 to lookupsize do
    writeln(outfile,points[m]);
    for m :=lookupsize-1 downto 1 do
    writeln(outfile,points[m]);
    close(outfile);
  end;
```

```
(**************************)
(* START OF MAIN PROGRAM *)
(**************************)

BEGIN
  clrscr;
  write('Generate root raised-cosine (1) or raised-cosine
(2)?:');
  readln(choice);
  if choice=1 then
    Generate_Root_Raised_cosine
  else
    Generate_Raised_cosine;
END.→
```

APPENDIX I. M.1020 FREQUENCY CHARACTERISTICS

The CCITT Recommendation M.1020 specifies the characteristics of special quality leased circuits with bandwidth conditioning. The limits for the overall loss relative to that at 800Hz are shown in Figure I.1. Figure I.2 shows the limits for group delay relative to the minimum measured group delay in the 500 - 2800Hz band.

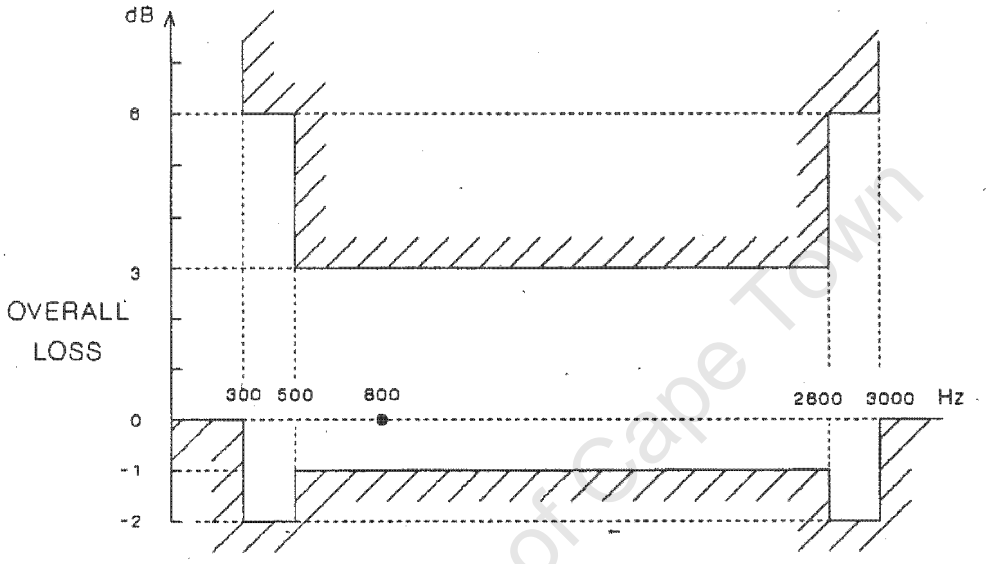


Figure I.1. - The limits for the overall loss relative to that at 800Hz.

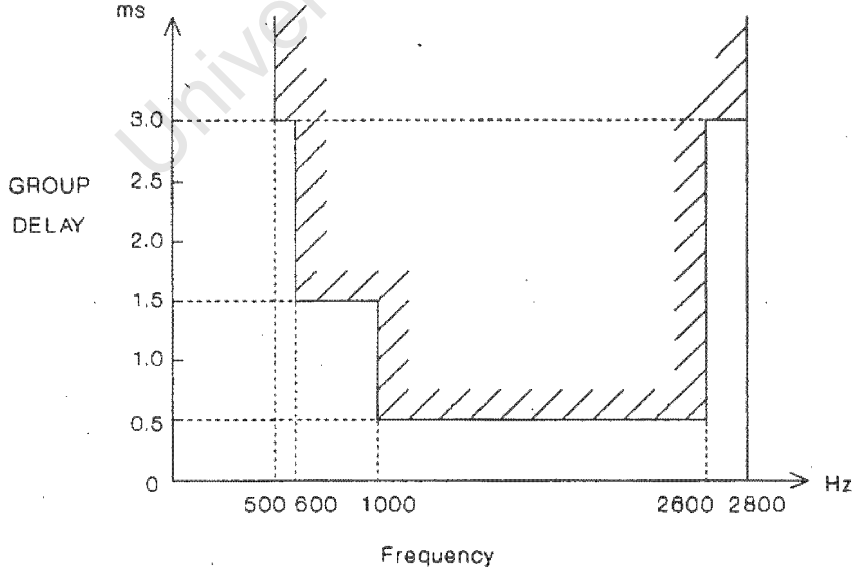


Figure I.2. - The limits for group delay relative to the minimum measured group delay in the 500 - 2800Hz band.

APPENDIX J. DIGINET

INTRODUCTION

DIGINET is a relatively new data link service established by the South African Posts and Telecommunications (SAPT), offering dedicated leased-line facilities to subscribers ['Planning for Diginet', STC Transmission Division]. It allows point-to-point and point-to-multipoint links over digital transmission systems, and is capable of carrying data up to 64 kilobits/sec (kbps). This is achieved by using a system of fibre optics, digital microwave radio and digital coaxial cable as carriers. Originally, DIGINET was targeted at the major business centres, but now analogue long-line links to rural locations are available. A schematic of DIGINET is shown in Figure J1.

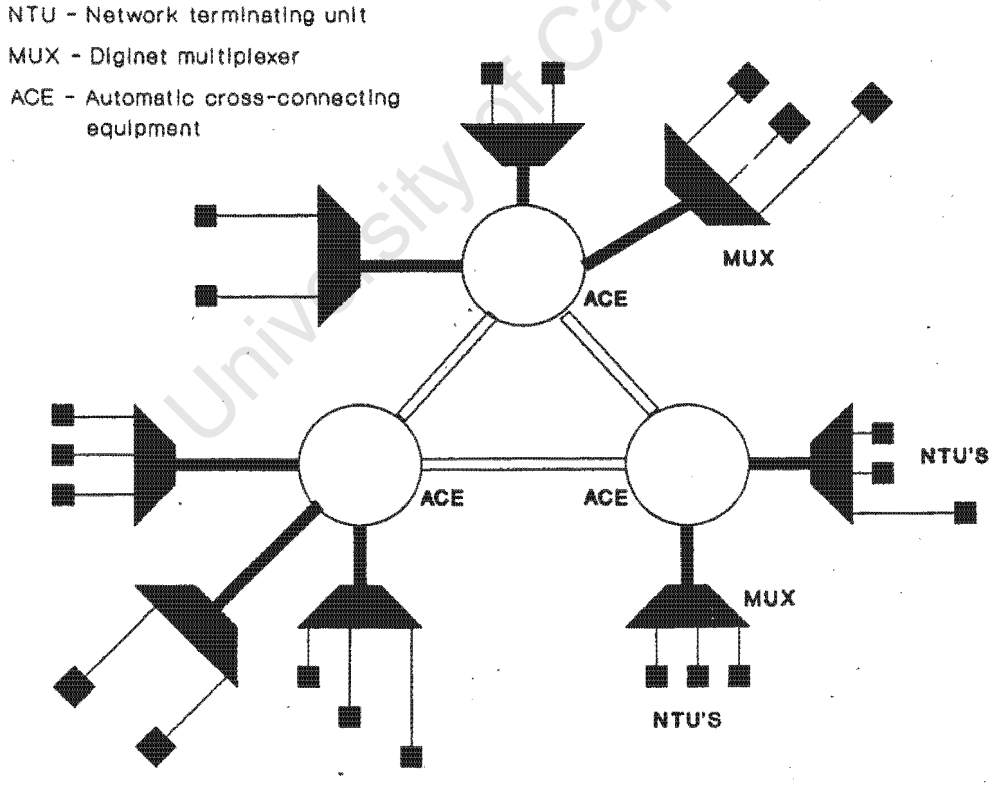


Figure J.1 - The DIGINET network.

THE LONG LINE RURAL LINK

This is the application that is relevent to this project. Figure J2 shows how a rural NTU is connected to DIGINET. Note that modems are situated at the ends of the analogue line. The analogue line is a CCITT recommendation M.1020 4-wire link.

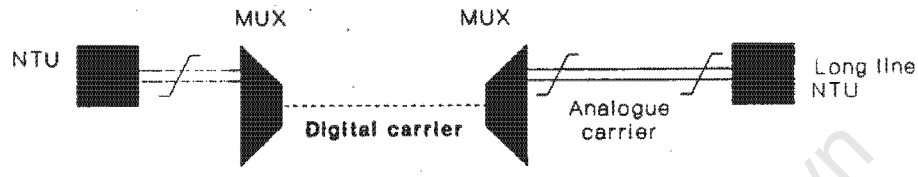


Figure J.2 - The rural link to DIGINET.

TYPICAL DIGINET APPLICATIONS

There are two major applications:- the point-to-point circuit and the point-to-multipoint circuit. Point-to-point operation is the simplest and most popular, and allows transmission at the higher speeds. Multipoint circuits cater for transmission of data at up to 9600 bps, between multiple NTU's.

APPENDIX K. PUBLISHED PAPER - 'THE DEVELOPMENT OF NOVEL MODEM STRUCTURES TO ENHANCE THE THROUGHPUT OF DIAL UP TELEPHONE LINES'

Paper published in COMSIG 1990, Proceedings, Johannesburg.

THE DEVELOPMENT OF NOVEL MODEM STRUCTURES TO ENHANCE THE THROUGHPUT OF DIAL UP TELEPHONE LINES

RUSSEL HORWITZ, TIM D. COURTENAY

and ROBIN M. BRAUN, Member IEEE.

University of Cape Town

ABSTRACT

A popular approach to designing high-speed modems for use on voice-grade telephone channels is to use a high order modulation scheme combined with a digital architecture. This paper describes the development of a modem structure which uses the DSP56001 signal processor to implement 49 quadrature partial response signalling (49 QPRS). The choice of 49 QPRS is justified by its high spectral efficiency and reasonable error rate performance. This system is being developed for two applications, which are described in the paper.

INTRODUCTION

The speed of data transfer over conventional telephone lines is limited by various factors. The predominant limitations are:

1. Channel group delay and magnitude response.
2. Noise introduced by the channel.
3. Noise introduced by the hardware.
4. Hardware complexity and cost.

The modems discussed in this paper are designed for transmission over CCITT standard M.1020 voice grade telephone lines [1]. Two modem designs are described which use a digital implementation of partial response signalling to achieve high data rates at relatively low cost and complexity, while keeping noise introduced by the hardware to a minimum. Section 1 describes the 49 QPRS modulation scheme. This is compared to 16 QAM, which is commonly used for high-speed, bandwidth efficient modem applications. In Section 2, the digital implementation of QPRS is described. The transmitted eye diagram is shown. Sections 3 and 4 describe the features unique to each of the two designs. Time waveforms and power spectra are shown to illustrate their operation.

1. 49 QUADRATURE PARTIAL RESPONSE SIGNALLING

Partial response signalling (PRS) is a bandwidth efficient modulation technique which allows transmission at the Nyquist rate [2]. A controlled amount of intersymbol interference is introduced, causing a spectral shaping which allows a very high roll-off Nyquist filter to be implemented with relative ease. The modems described in this paper use class 1 PRS and employ the following coding rules:

precoding: $b_k = (x_k - b_{k-1}) \bmod 4$

encoding: $y_k = b_k + b_{k-1}$

decoding: $x'_k = y_k \bmod 4$

where x_k is the original data in the form of a 2 bit word (4 level)

b_k is the precoded sequence (4 level)

y_k is the transmitted sequence (7 level)

x'_k is the decoded data (4 level).

The low-pass filtered waveform of the 7-level sequence y_k (for one channel) is shown in Figure 1. The signal value at the eye locations is that of the transmitted y_k sequence. This scheme is used in conjunction with quadrature modulation to produce the 49 QPRS signal. The bandwidth efficiency of this signal is 4 bits/sec/Hz. Figure 2 shows the constellation diagram.

Figure 3 shows performance curves for various modulation schemes. Notice that 49 QPRS has only 2 dB worse error performance than 16 QAM, although 16 QAM requires up to double the bandwidth for the same bit rate, depending on the spectral shaping used. 256 QAM, however, has similar spectral efficiency as 49 QPRS but requires a 10 dB increase in Carrier to Noise ratio for the same error rate.

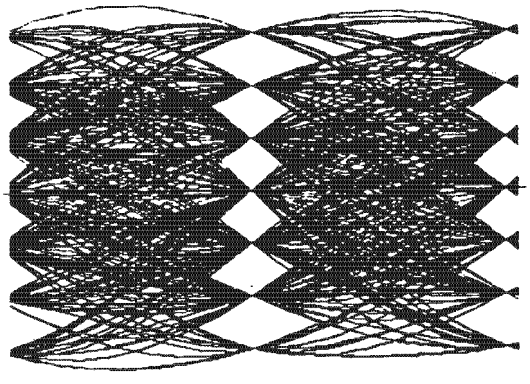


FIGURE 1 - PRS 7-level baseband eye diagram (1 channel shown).

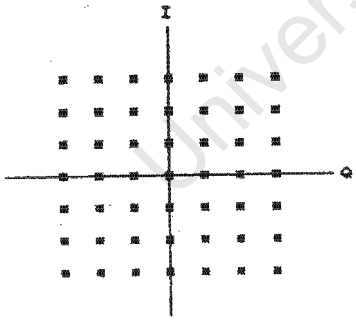


FIGURE 2 - Constellation diagram for 49 QPRS.

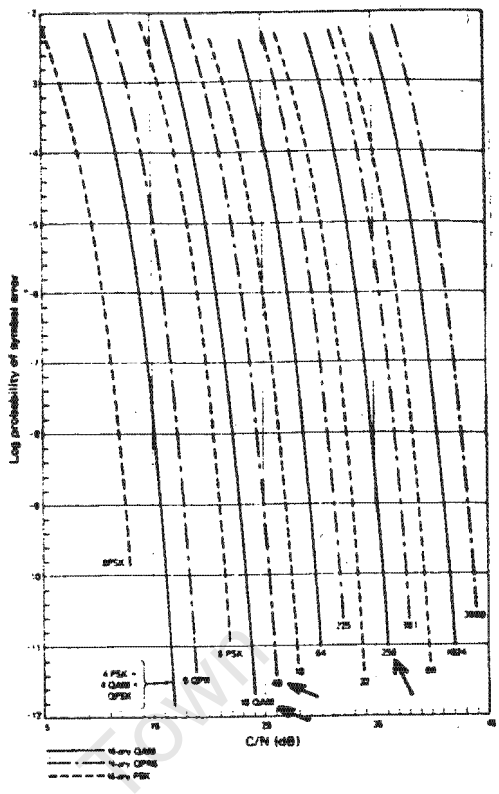


FIGURE 3 - Probability of error performance curves of M-ary PSK, M-ary QAM and N-ary QPR modulation systems [3].

2 DIGITAL IMPLEMENTATION

The Motorola DSP56001 digital signal processor is used to implement all encoding, low-pass filtering, modulation, demodulation, decoding and the addition of pilot tones. Figure 4 shows the sequence of events relating to the transmitter and the receiver. The important advantages of this processor are [4]:

1. A parallel architecture and a no-overhead, hardware DO loop allow FIR filters to be implemented in the theoretical minimum number of execution cycles (one per tap weight).
2. Modulo addressing registers allow efficient implementation of circular buffers.
3. The high speed at which the processor operates (10.25 mips).
4. A built in sine-wave lookup table which is useful for implementing modulation

The above approach allows a complex scheme to be implemented with relative ease. As a result, the time and frequency response correlate very well with theoretical predictions.

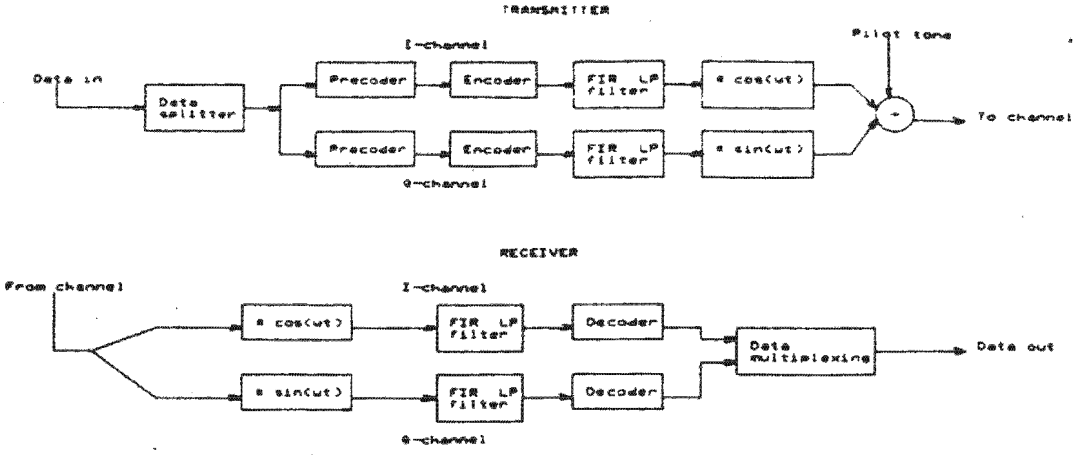


FIGURE 4 - Modem functions implemented by the DSP56001 microprocessor.

To effect a filter pair that complies with both matched filter and Nyquist criteria, square-root raised cosine FIR filters are used at the transmitter and receiver [5]. The frequency response of this filter is given by :

$$X(w) = \left(T/2 \left(1 + \cos \frac{w}{2W} \right) \right) \quad \text{for } w \ 2W \ (1)$$

$$= 0 \quad \text{elsewhere}$$

The FIR filter tap weights, which form the impulse response, were obtained using numerical integration. Sufficient oversampling ensures that spectral replicas can be easily rejected and that the aperture effect [6] does not degrade the signal.

3. APPLICATION A

The first modem design aims to upgrade V.22bis, which is a 2400 bit/sec, full-duplex, frequency division multiplexed modulation scheme, to 4800 bits/sec full duplex operation [2]. As in V.22bis, the carrier frequencies for the transmit and receive channels are at 1200 and 2400 Hz. Pilot tones are transmitted at 400 and 600 Hz with the 2400 and 1200 Hz channels respectively. The system assumes a usable bandwidth of 400 to 3000 Hz.

The frequency of the carriers, symbol timing and pilot tones are all directly related. This allows the carriers and symbol timing waveforms to be unambiguously derived from the pilot tones, which are recovered with phase-locked loops at the receiver. Although the 600 Hz pilot tone is on the band edge of the 1200 Hz channel, it does not interfere with the data as it inserted in such a way that its zero crossings coincide with the eyes. The 400 Hz tone is widely separated in frequency from the 2400 Hz channel and is easily filtered out at the receiver. The measured spectra of the two channels are given in Figure 5.

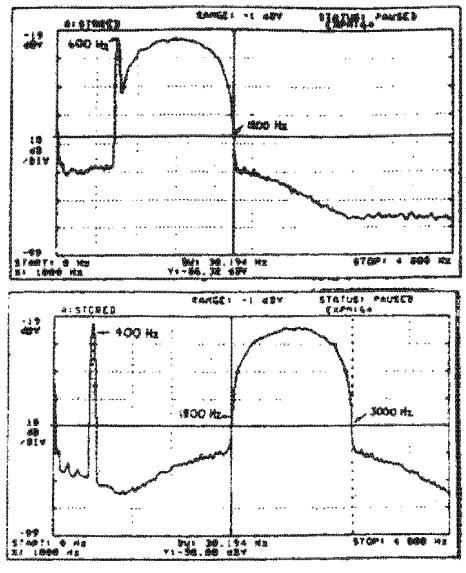


FIGURE 5 a) - 1200 Hz channel with 600 Hz pilot tone.
b) - 2400 Hz channel with 400 Hz pilot tone.
(4 % excess root raised cosine bandwidth)

4. APPLICATION B

The second modem design intends to replace a V.29-type modem, which has variable data rates of 2400, 4800 and 9600 bits/sec and operates over conditioned leased lines, with one intended to function over M.1020 voice-grade lines. Note that there is no bit rate improvement, but the reduced spectral width now conforms to M.1020 constraints. The variable data rates are achieved as follows:

Data Rate	Modulation scheme
9600	49 QPRS
4800	9 QPRS
2400	3 PRS (In-phase channel only)

In the 4800 bits per second case, a one bit word is used. The coding equations remain the same except that the mod 4 operation is changed to mod 2. A further reduction in the bit rate to 2400 bits per second is achieved by using the in-phase channel only. The time waveforms for these cases are given in Figure 6. Figure 7 shows the signal spectrum for the case of 9600 bits/sec. The carrier frequency is 1800 Hz while the pilot tone is inserted at 600 Hz. The pilot tone is employed for the same reason as in application A.

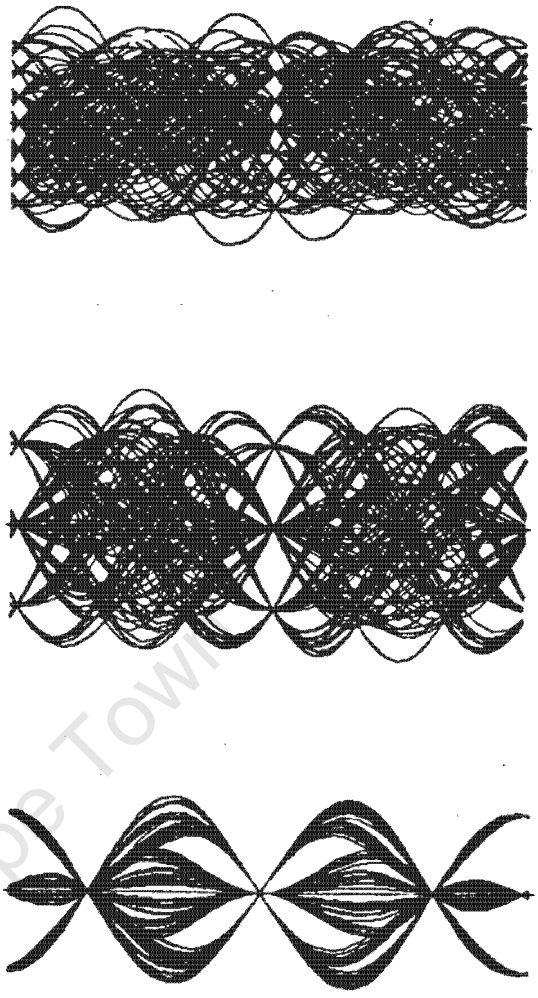


FIGURE 6 - Time waveforms of the three modulation schemes:
a) - 9600 bps
b) - 4800 bps
c) - 2400 bps.

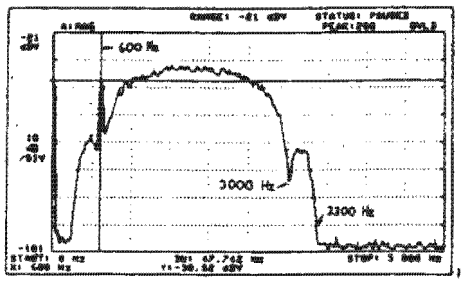


FIGURE 7 - Spectrum of the 49 QPRS waveform (9600 bps). (30 % excess root raised cosine bandwidth)

CONCLUSION

A modem design exhibiting features such as high bandwidth efficiency, low complexity and digital reliability has been described. Two specific applications for which this design is particularly well suited have been described. Future work will include comprehensive signal-to-noise versus bit error rate testing.

REFERENCES

1. Lender A., "The Duobinary Technique for High-speed Data Transmission", IEEE Trans. Commun. Electron., vol. 82, pp. 214-218, May 1963.

2. CCITT Recommendation M.1020, Department of Posts and Telecommunications.

3. Feher K., "Advanced Digital Communications, Systems and Signal Processing Techniques", Prentice-Hall Inc., New Jersey, 1987, pg. 328.

4. Motorola DSP56001 Users Guide.

5. Stremler F. G., "Introduction to Communication Systems, 2nd Edition", Addison-Wesley, Reading, Massachusetts, 1982, pg. 367-370.

6. Taub H. and Schilling D.L., "Principles of Communication Systems", McGraw-Hill, New York,1971.