The Use Of Digital Signal Processing in Satellite Communication

by

Jonathan Richard Bramwell.

Submitted to the Council for National Academic Awards in Partial Fulfilment for the Degree of Doctor of Philosophy.

March 1988

Sponsoring Establishment:
Plymouth Polytechnic

Department of Communications Engineering.

Collaborating Establishment:
Royal Signals and Radar Establishment, Defford.

Version :1.01

The Use Of Digital Signal Processing in Satellite Communication

by

Jonathan Richard Bramwell.

Abstract

The recent emphasis on Information Technology has increased the need for methods of data communications with a greater interest in the areas of Satellite communications. Data communications over a satellite can be easily achieved by the use of excessive power and bandwidth but efficient management of the satellite resource requires more elegant means of transmission. The optimum modulator and demodulator can be described by mathematical expressions to represent the physical processes that are required to transmit and receive a signal. Digital Signal Processing circuits can be use to implement these mathematical functions and once correctly designed are not susceptible to variations in accuracy and hence can maintain an accurate representation of the mathematical model.

This thesis documents an investigation into the algorithms and techniques that can be used in the digital implementation of a Satellite Data Modem. The technique used for carrier phase recovery and data decoding is a major variation on a method proposed by Viterbi and Viterbi and relies on Phase Estimation instead of the more common carrier regeneration techniques. A computer simulation of this algorithm and its performance is described and the overall performance of the simulation is compared to theoretical analysis and experimental performance of a Multi-Data Rate Satellite Modem covering data rates in the range 16 Ksymbol/sec to 256 Ksymbol/sec in both the BPSK and QPSK data formats.

1. Introduction.

The recent emphasis on Information Technology has increased the need for methods of data communications with a greater interest in the areas of Satellite communications. Data communications over a satellite can be easily achieved by the use of excessive power and bandwidth but efficient management of the satellite resource requires more elegant means of transmission. Many methods have been devised and the principles are outlined in several good communications text book <3,5,6,9,10,13> along with the theoretical solutions for solving the problems that satellite communications bring i.e. Narrow bandwidths and high phase noise, Low signal power, large frequency errors, noise, etc. However, many solutions are theoretically possible but practically beyond, or at the edge of, analogue technology and so difficult to implement. Until recently the modulation and demodulation of all signals relied on the use of analogue circuitry to generate the modulated signal and to demodulate the received signal. The optimum modulator can be described by a mathematical expression to represent the physical processes that are required to modulate and transmit a signal. Similarly the optimum demodulator may be represented by a similar set of mathematical expressions. However, the use of analogue components to implement these mathematical expressions in the modulation and demodulation processes is only an approximation of the mathematical process required and hence a revision is required of the mathematical model. This results in the model now being non-optimum. The major cause of this effect is due to the individual components only approximating to a required function, i.e. multiplier, adder, integrator etc. Often these approximations are perfectly valid over a range of interest but require much development work to implement the function correctly and difficulty in maintaining accuracy. With the increase in

processing speeds now available in Large Scale Integration (LSI) and Very Large Scale Integration (VLSI) digital circuits the use of analogue components is no longer the only viable alternative. The use of digital circuits can implement these mathematical functions precisely with the only limitations being set by the sampling frequency and the required signal to quantization noise at the output, set by the number of bits within the digital representation. It may also enable functions to be implemented that are almost impossible to create using analogue techniques ,i.e. close approximations to Non causal filters. The digital implementation is also, once correctly designed, not susceptible to variations in accuracy and hence can maintain an accurate representation of the mathematical model.

This thesis documents an investigation into the algorithms and techniques that can be used in the digital implementation of a Satellite Data Modem. The research into the design of such a modem was funded and initiated by the Department of Trade and Industry (DTI) and will be taken into production with the collaboration of Telemetrix Research Ltd, of Tewkesbury, Gloucestershire,UK. The specification for the modem was laid down by the DTI and is given in Appendix A. The basic requirement was for a digital modem that could employ Binary Phase Shift Keying (BPSK) or Quadrature Phase Shift Keying (QPSK) with absolute decoding over a range of data rates from 16 Ksymbols/sec to 256 Ksymbols/sec with an extension up to 4.096 Msymbols/sec. These data rates were required to be quickly switchable over the entire range. As a first step in devising the best Digital Signal Processing (DSP) algorithms the mathematical models of the transmitting and receiving functions need to be described for different signalling formats. This will then enable the modulator and demodulator to be implemented in Digital hardware. This

algorithm and hardware examination is divided into sections documenting the modulator and the demodulator, with sub-sections on digital demodulation, filtering, phase decoding and carrier phase recovery. The technique used for carrier phase recovery and data decoding is a major variation on a method proposed by Viterbi and Viterbi <26> but correcting all the errors and problems imposed by the Viterbi's technique. These new proposals are thought to be unique and have been patented <19>. A computer simulation of this algorithm and its performance is described and the overall performance of the simulation is compared to theoretical analysis and the experimental performance. The experimental results were obtained over a satellite link using the Intelsat 5A F11 satellite with the collaboration of British Telecomm International (BTI), Goonhilly Downs,Cornwall.

Suitable methods for modelling the signal to be transmitted and received are described below. Methods for digitally demodulating a signal ,either of a continuous nature (FM video signal), or of a discrete nature (BPSK/QPSK) are proposed.

2. Mathematical Modelling

2.1. Introduction to Mathematical Modelling of a Signal.

A mathematical model is a representation of a physical process. This model must be known at both the transmitter and receiver to enable the signal to be correctly modulated and demodulated. One model that is commonly used throughout communications systems is that of the Complex Envelope Representation.

2.1.1. The Complex Envelope Representation.

Any bandpass waveform can be represented in the form <6>:-

$$v(t) = \mathrm{Re}\{ g(t).e^{j\omega c.t} \} \qquad \text{(Eq2.1.1.a)}$$

where g(t) is the complex envelope of v(t)

$$g(t) = x(t) + j.y(t) \qquad \text{(Eq2.1.1.b)}$$

$$x(t) = A(t)\mathrm{Cos}\ \Phi(t) \qquad \text{(Eq2.1.1.c)}$$

$$y(t) = A(t)\mathrm{Sin}\ \Phi(t) \qquad \text{(Eq2.1.1.d)}$$

$$\Phi(t) = \tan^{-1}(y(t)/x(t)) \qquad \text{(Eq2.1.1.e)}$$

with the $\tan^{-1}(.)$ function extended over four quadrants.

Where $\omega c = 2.\pi.Fc.$, Fc is the carrier frequency of the bandpass signal and A(t) is the magnitude of the signal x(t), y(t), A(t) and $\Phi(t)$ are all real baseband signals, while g(t) is a complex baseband signal.

By substitution ,the bandpass waveform can therefore be fully represented by:-

$$v(t) = A(t)\text{Cos } \Phi(t).\text{Cos } \omega ct - A(t)\text{Sin } \Phi(t).\text{Sin } \omega ct$$

(Eq2.1.1.f)

Often the $x(t) = A(t)$ Cos $\Phi(t)$ is called the In-phase component and $y(t) = A(t).\text{Sin } \Phi(t)$ the Quadrature component. This model is also sometimes called the Quadrature Model.

The power of this model is considerable, not just in the representation of a bandpass signal from a baseband signal but also in the facts that if v(t) is deterministic then x(t),y(t),A(t), and $\Phi(t)$ are also deterministic.It is similarly true if v(t) is a stochastic process, i.e. representing bandpass noise, then x(t),y(t),A(t) and $\Phi(t)$ represent baseband noise and are similarly stochastic. As an example this model will be used to represent a Frequency Modulated (FM) Signal.

2.2. The FM Representation.

A FM signal can be represented by <5>:-

$$r(t) = A.\mathrm{Cos}\{\ \omega ct + K\int_{-\infty}^{\tau} m(t)dt + \theta(t)\ \}$$

(Eq2.2.a)

where m(t) is the modulating signal, K is a constant proportional to the modulation index of the FM signal, and $\theta(t)$ is an initial phase error at the receiver, which for simplicity is firstly consider to be zero.

$$\text{Let } K\int_{-\infty}^{\tau} m(t)dt = \Phi(t) \qquad \text{(Eq2.2.b)}$$

therefore

$$r(t)=A.\mathrm{Cos}\{\ \omega ct + \Phi(t)\ \} \qquad \text{(Eq2.2.c)}$$

by simple trigonometry this becomes:

$$r(t)=A.[\mathrm{Cos}\ \Phi(t).\mathrm{Cos}\ \omega ct - \mathrm{Sin}\ \Phi(t).\mathrm{Sin}\ \omega ct]$$

(Eq2.2.d)

This can be clearly seen to be in the form of the complex envelope representation hence the components $A.\mathrm{Cos}\Phi(t)$ and $A.\mathrm{Sin}\Phi(t)$ are both baseband in nature and with a carrier frequency of $\omega_c/2\pi$. The object of the demodulator is to recover the modulating signal m(t).

If r(t) is able to be multiplied by Cos ct,which assumes a perfect phase reference,then the result is given by :-

$$r_I(t) = A\{\text{Cos } \Phi(t).\text{Cos } \omega ct - \text{Sin } \Phi(t).\text{Sin } \omega ct \}.\text{Cos } \omega ct$$

$$= A.\{ \text{Cos}(\omega ct + \Phi(t)\}.\text{Cos } \omega ct$$

$$= \frac{A.\{ \text{Cos}(2\omega ct + \Phi(t)) + \text{Cos } \Phi(t)\}}{2} \qquad \text{(Eq2.2.e)}$$

Low pass filtering to remove high order frequency components and normalising gives:-

$$r_I(t) = A.\text{Cos } \Phi(t) \qquad \text{(Eq2.2.f)}$$

Similarly if r(t) is multiplied by Sin ωct and filtered, the Quadrature component is obtained:-

$$r_Q(t) = A.\text{Sin } \Phi(t) \qquad \text{(Eq2.2.g)}$$

The practical implementation of this process can be seen in fig 2.2.a.

From these two components it is possible to determine $\Phi(t)$:-

Defining

$$\Phi(t) = \Gamma^{-1}\left[\frac{r_Q(t)}{r_I(t)}\right] \qquad \text{(Eq2.2.h)}$$

Where

$$\Gamma^{-1}(.) = \frac{[1+\text{sign}(r_Q(t))]}{2}\left[\Phi'(t)+[1-\text{sign}\{r_I(t)\}](\pi/2)\right] + \frac{[1-\text{sign}(r_Q(t))]}{2}\left[\Phi'(t)-[1-\text{sign}\{r_I(t)\}](\pi/2)\right] \qquad \text{(Eq2.2.i)}$$

and

$$\Phi'(t) = \text{Tan}^{-1}\left[\frac{r_Q(t)}{r_I(t)}\right] \qquad \text{(Eq2.2.j)}$$

Sign(.) is the signature function defined as:-

$$\text{sign}(x) = \begin{cases} 1 & \text{for } x \Rightarrow 0 \\ -1 & \text{for } x < 0 \end{cases}$$

since $\Phi(t) = K \int_{-\infty}^{\tau} m(t)dt$

$$m(t) = \frac{1}{K} \cdot \frac{d\Phi(t)}{dt} \qquad \text{(Eq2.2.k)}$$

and hence it is possible to resolve the value of m(t). It should be noted at this point that it can be shown that even if the phase reference contains an error then the effect in the receiver is removed by the differentiation to produce the same result as Eq 2.2.k above.

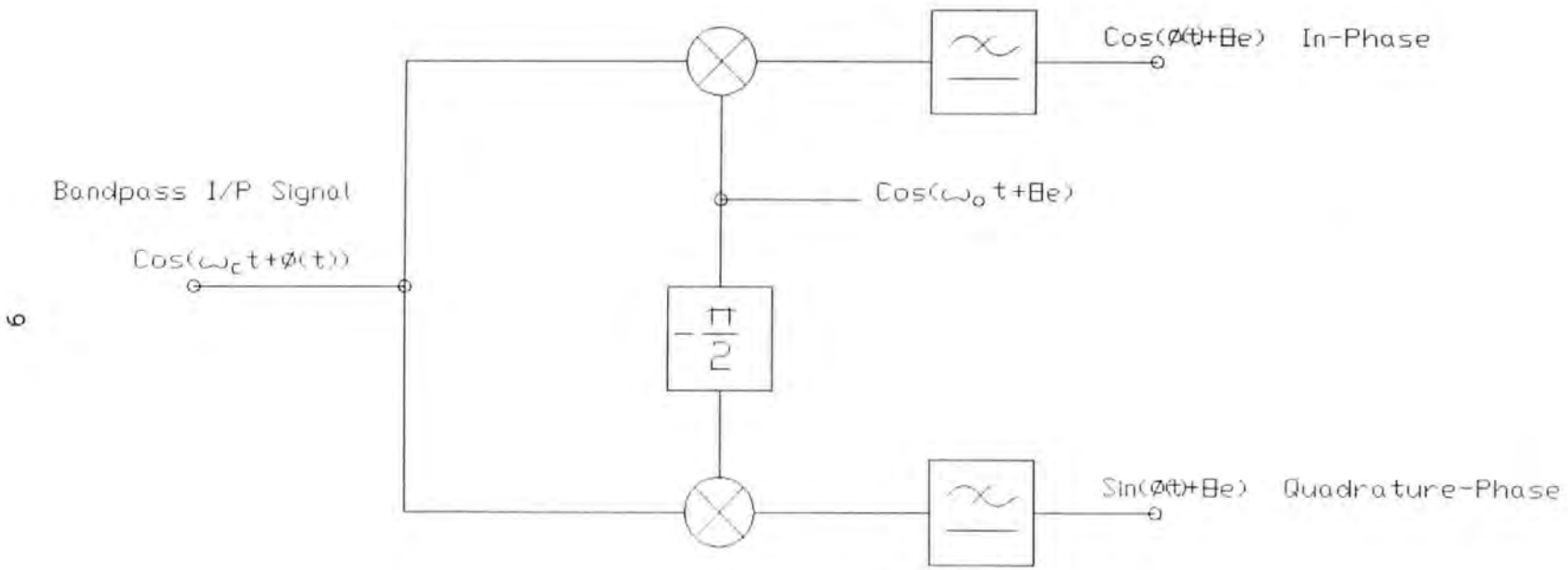


Fig 2.2.a Quadrature Down-Conversion Implementation

2.3. Frequency Spectra.

Considering Eq2.2.d it can be shown that this transforms into a frequency spectra consisting of both positive and negative frequencies around 0 Hz with a real component only as previously defined in Eq2.1.1.a and hence is more clearly described by :-

$$r(t) = A.\left[\frac{e^{j(\omega ct+\Phi(t))} + e^{j(-\omega ct-\Phi(t))}}{2} \right] \qquad \text{(Eq2.3.a)}$$

This can be expressed in diagrammatic form as in fig 2.3.a. When down-converted to baseband by a local oscillator with a perfect phase reference of Cos ωct the resultant before Low pass filtering is given by:-

$$r_I(t) = \frac{A\left[e^{j(2\omega ct+\Phi(t))} + e^{-j(2\omega ct+\Phi(t))} \right]}{4} + \frac{A\left[e^{j\Phi(t)} + e^{-j\Phi(t)} \right]}{4} \qquad \text{(Eq2.3.b)}$$

This is shown in fig 2.3.b. Similarly with down-conversion using a quadrature local oscillator of Sin ct the resultant spectra is given by:-

$$r_Q(t) = \frac{A\left[e^{j(2\omega ct+\Phi(t))} - e^{-j(2\omega ct+\Phi(t))} \right]}{4j} - \frac{A\left[e^{j\Phi(t)} + e^{-j\Phi(t)} \right]}{4j} \qquad \text{(Eq2.3.c)}$$

and can be seen in Fig 2.3.c.

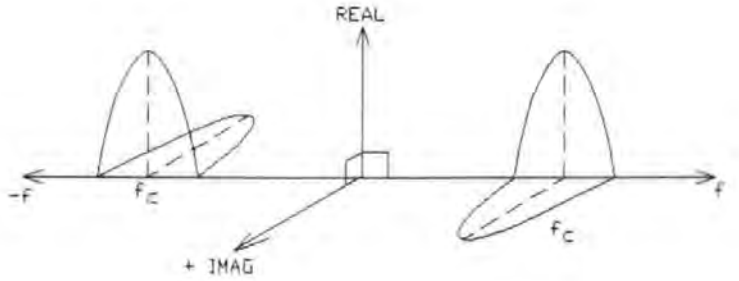
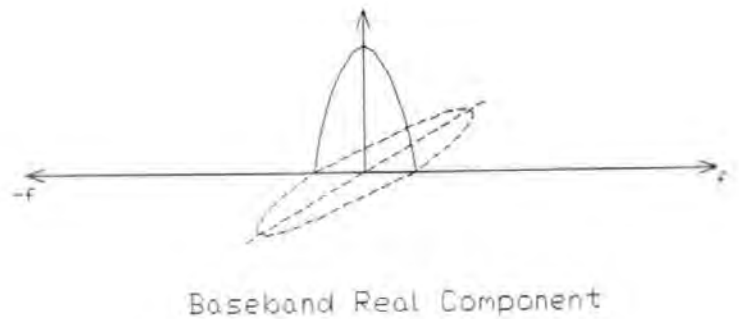


Fig 2.3.a Bandpass Frequency spectra



Baseband Real Component

Fig 2.3.b Frequency spectra for In-phase Channel

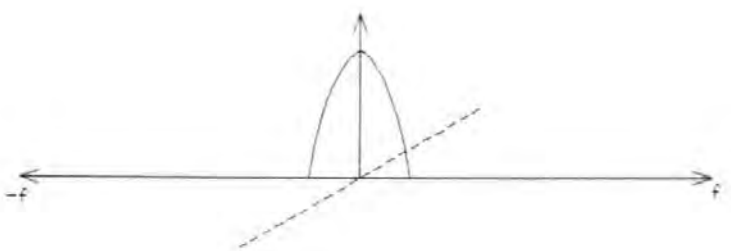


Fig 2.3.c Frequency Spectra For Quadrature Phase Channel

2.4. Processing of Quadrature Baseband Components.

Consider the digital implementation of a demodulator for the FM example above. As a practical example the signal to be demodulated is a 30 MHz bandwidth, FM modulated TV signal transmitted over a KU band satellite link such as Eutelsat F1 or Intelsat V F22. The signal is received from the satellite by a parabolic antenna (dish) of typically 1.2 metres diameter in the frequency range 10.95 to 11.7 GHz. The low noise amplifier and down-converter, amplify and convert the signal down to a 1st IF of approximately 1 GHz. A typical spectrum is shown in fig 2.4.a. Further analogue circuitry down-converts to a 2nd IF of 70 MHz, while also providing frequency selectivity to reduce the spectrum to that of the wanted signal only. This signal is then at a suitable stage to input to a Quadrature down-converter prior to the digital demodulator. The operation of the frequency conversion is standard and will not be elaborated on further.

After frequency conversion to baseband, as described above, the signals must be low pass filtered. In this implementation the filters are implemented by the use of analogue components. The bandpass signal was 30 Mhz but at baseband this is reduced to 15 Mhz. Before digital processing can take place, the choice of sampling rate must take into account the bandwidth of the baseband analogue signal. Sampling theorem states that any continuous signal will be fully represented by a sequence of samples provided that the sampling frequency is at least twice the highest frequency present <6>. If this is not the case, then aliasing occurs and the resulting spectra is corrupted as shown in Fig 2.4.b. This implies that the received quadrature signals must be sampled at a rate of at least 30 Mhz. However the frequency ,at which the signal has

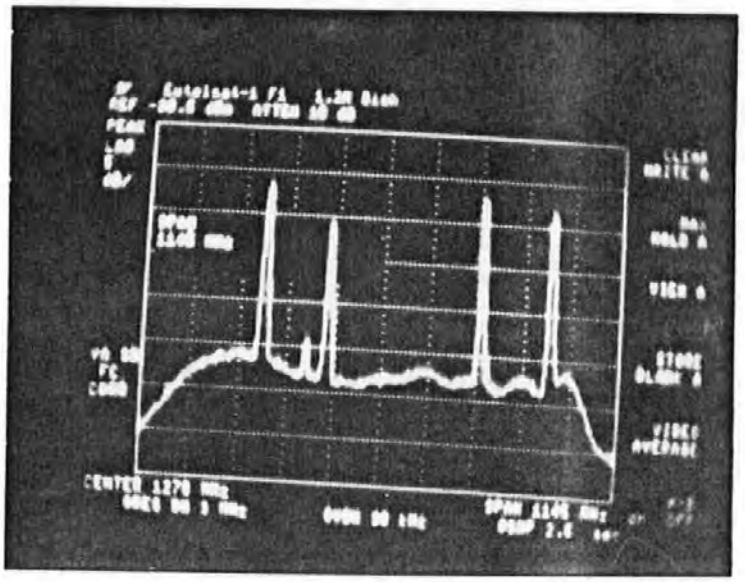


Fig 2.4.a Typical Received Spectrum for EutelSat F1

significant spectral components above -20 dB, extends to 17 MHz and therefore aliasing will occur unless sampled at least at 35 MHz. As this is beyond the range of the digital components to operate, the baseband signals are filtered to 12.5 Mhz 3dB bandwidth with a 15 Mhz stopband,i.e. more than 30 dB attenuation at 15 MHz, using a 7th Order Elliptic filter as shown in fig 2.4.c. Therefore sampling can occur at 30 Mhz. This causes some degradation in saturated colour performance by attenuating the wide deviation signals that these colours produce, although this degradation is tolerable in practice.One effect of this attenuation is that the signal can now drop below the FM threshold, in these regions, when operating close to SNR threshold. This results in 'theshold events' or 'noise spikes' in the received picture . The demodulator must recover the modulating video signal and to achieve this two separate processes are required,as shown in the block diagram fig 2.4.d. The first process is to obtain the phase of the vector $\Phi(t)$. This is achieved by decoding the sampled Phase and Quadrature channels into a Phase angle by the use a Programmable Read Only Memory (PROM). The incoming baseband channels are each sampled at 30 MHz by multiplexing two A/D converters sampling at 15MHz to produce two 6 bit sample streams. The total number of combinations for both channels, 12 bits, is $2^{12} = 4096$ which can easily be accommodated in a single PROM. This PROM would therefore be programmed with an algorithm to convert a 12 bit address to a 8 bit output which is proportional to the phase of the vector. The algorithm used is therefore that of the extended Tan^{-1} function ($\Gamma^{-1}(.)$), as previously described in Eq 2.2.h. The output is now a representation of the phase vector represented by the two quadrature channels. In the quadrature model analysis it was shown that to fully demodulate the received signal into the original modulating video signal,differentiation must take place. From the output phasor map, fig 2.4.e, it can

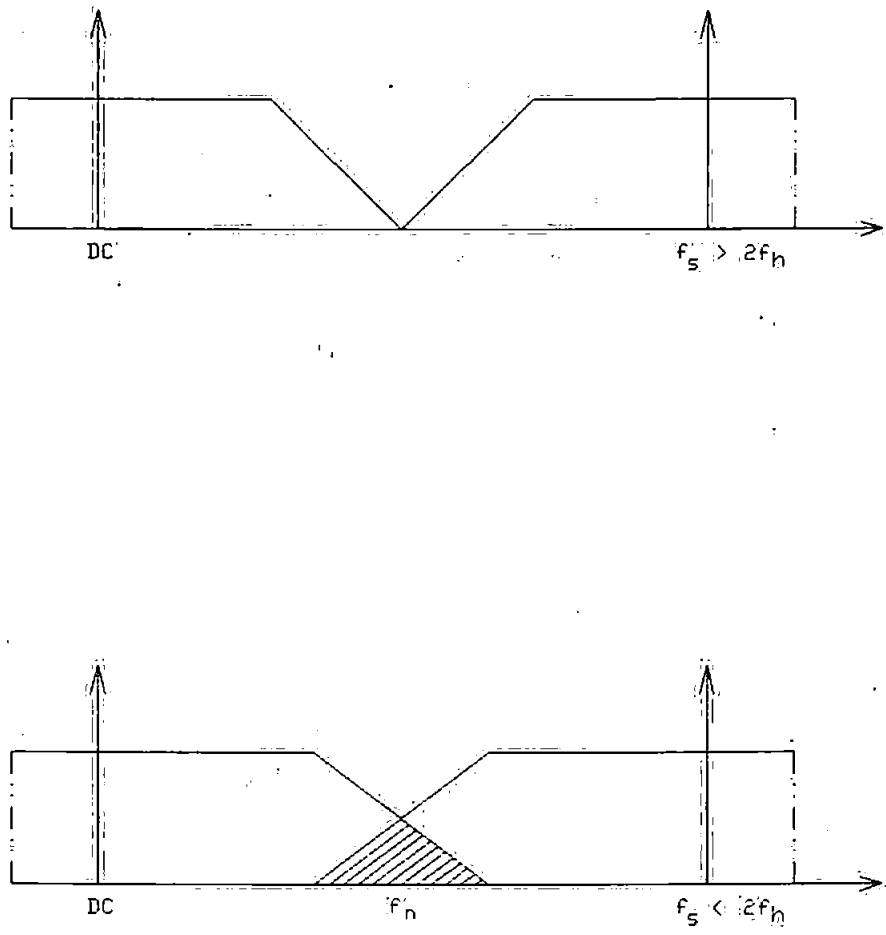


Fig 2.4.b Aliasing Frequency Spectra due to sampling frequency too low

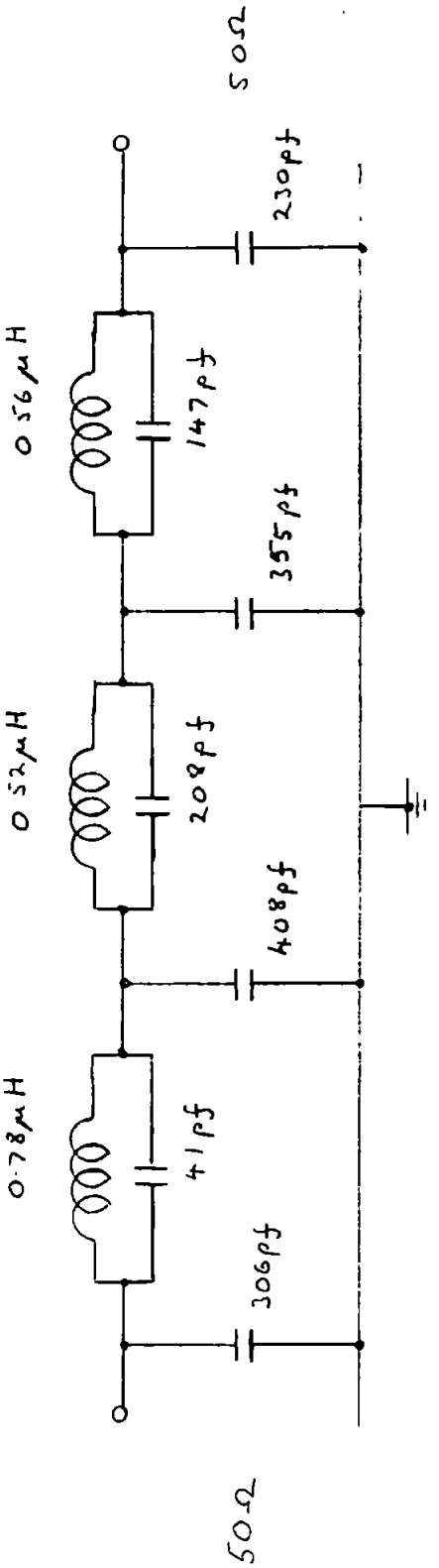


Fig 2.4.c 7 th Order Elliptic Filter

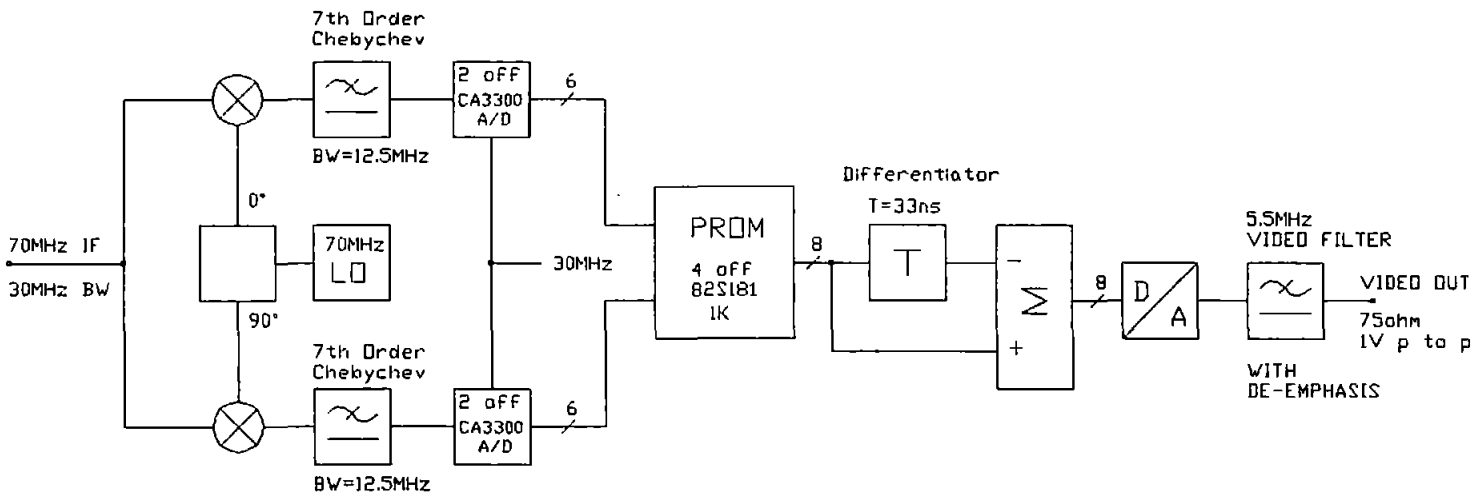


Fig 2.4.d Demodulation of an FM signal using Quadrature Down Conversion

be seen that the vector is represented as being a positive increasing value to a maximum of +180 degrees in an anti -clockwise direction and a negative increasing value to a maximum of -180 degrees in the clockwise direction. The difference value is obtained between two successive sample vector values, and in each case the 'shortest route' must be taken. This is fully explained in a later section but a simple interpretation can be obtained by considering sampling theory. If a rotating phasor of period τ, represented by two quadrature components is sampled according to sampling theory, this sampling must take place at a frequency of $2/\tau$. Hence if the first sample occurs at 0 degrees then the next sample occurs at the 180 degree point. If the period is less the τ then the phasor sample difference will be less than 180 degrees. Since this is the case and we have defined the maximum phasor difference to be 180 degrees by the sampling theorem then the phasor can only have followed a path around the unit circle which is less than 180 degrees. This is termed the "shortest route". The calculation can be performed by signed magnitude subtraction modulo 360 degrees fig 2.4.f. This form of shortest route subtraction can be more easily expressed in a 2's Complement form of subtraction modulo 360. Following the differentiator the output is fed to a digital to analogue converter. This reproduces the sampled analogue signal which because of its sample nature has frequency spectra existing around multiples of the sample rate frequency. These must be removed by low pass filtering to fully implement the equivalent of the analogue FM demodulator. The analogue signal is then de-empasised and buffered to drive the 75Ω load of the video monitor. Hence by the use of the above processes the FM signal has been demodulated digitally. This approach has been fully explored experimentally as a first stage to this project and the results and conclusions can be found described in detail in reference <21>.

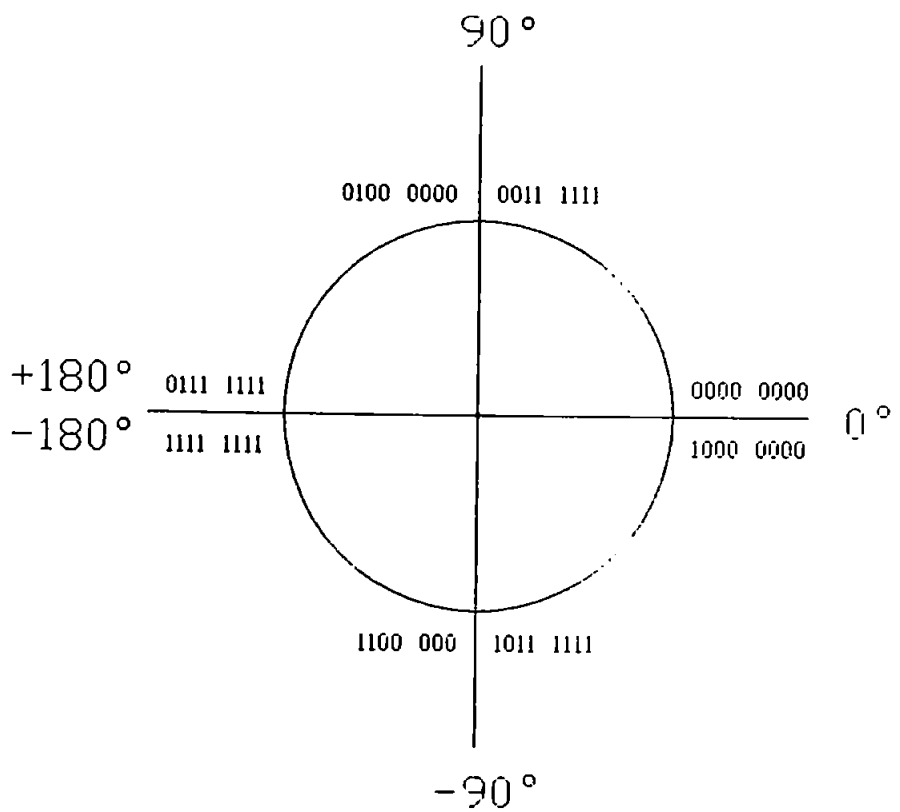


Fig 2.4.e Output Phase Map for Extended $\tan^{-1}$ Function ($\Gamma^{-1}(.)$) With Sign/ Magnitude Representation

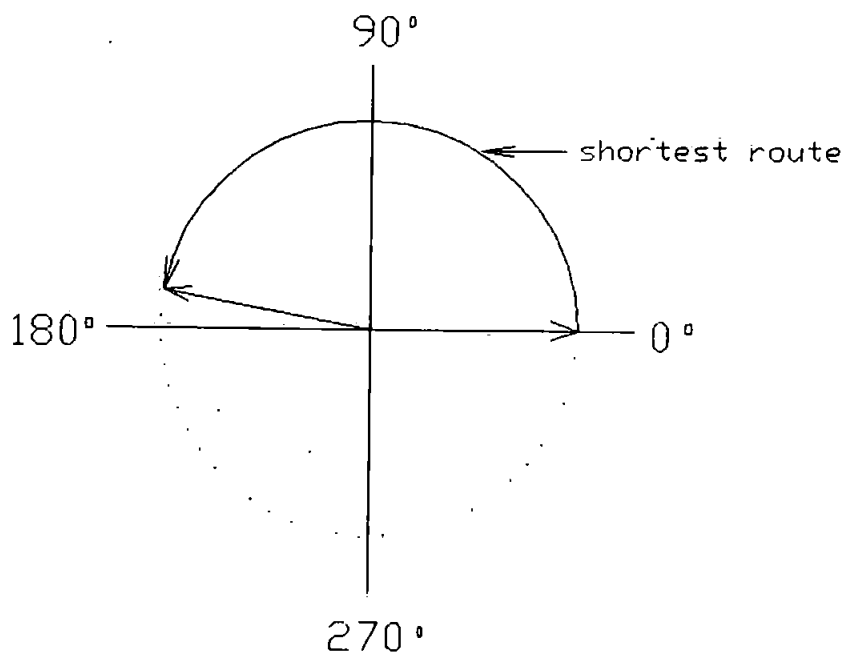


Fig 2.4.f Example of a 'Shortest Route' Calculation around the Unit Circle

3. A Discrete Time Version of the Digital Demodulator for Data Transmission.

In the previous section the digital demodulator was used to implement the discrete reception of a continuously changing time signal i.e analogue, without any loss of definition. A configuration will now be considered where a binary modulated signal is demodulated. From equation 2.1.1.f, it should be noted that the phase angle $\Phi(t)$ is a continuous variable with time, as is the magnitude A(t). Even if this is not the case, and the phase and amplitude only change at discrete instants in time, the complex envelope model can still be used to model the bandpass waveform. If the In-phase and Quadrature components represent the 2 dimensional axis of Eucilidean space, a vector space diagram can be constructed. A simple constellation whereby the amplitude is constant at unity and the phase can take on any one of four equispaced values of 45,135,225,315 degrees can be represented on the space diagram as in Fig 3.a. This is a representation for Quadrature Phase Shift Keying. If the amplitude takes on any one of two values as well as the four phase value then a form of Quadrature Amplitude Modulation is described, as shown in Fig 3.b. Both these signals can be easily decoded by the digital demodulator already described from the ratios of the phase and quadrature channel values by changing the decoder function incorporated into the decode PROM. Many forms of signal constellation can therefore be demodulated by the use of the same digital demodulator technique by programming appropriate decode PROM's. However for a discrete sampled system the function of symbol synchronization and carrier phase lock have also to be carried out. These subjects will be dealt with in a later section but first some basic principles of Binary Phase Shift Keying (BPSK) are considered in connection with its spectrum, spectral shaping by means of filtering and the effects of noise at the receiver.

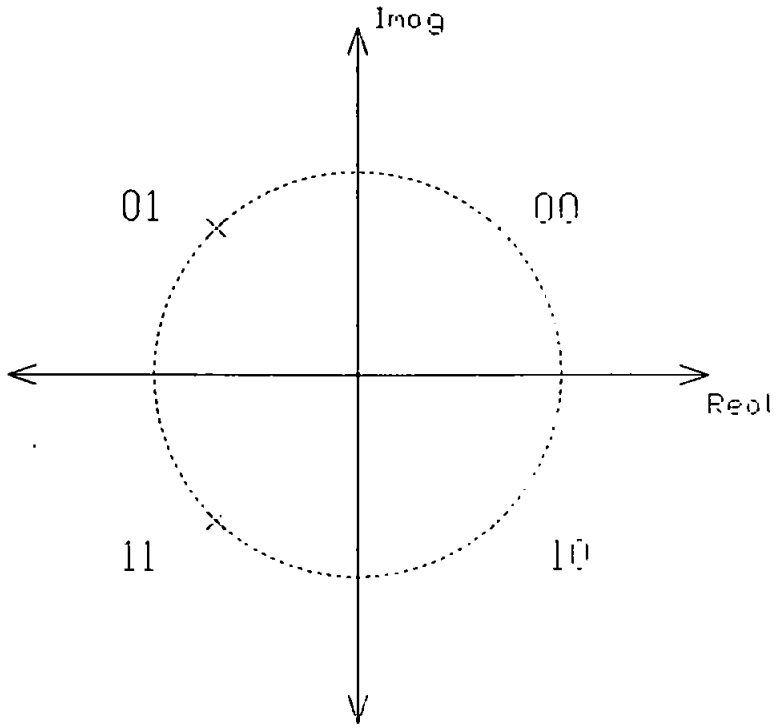


Fig 3.a Space Diagram Constellation for QPSK

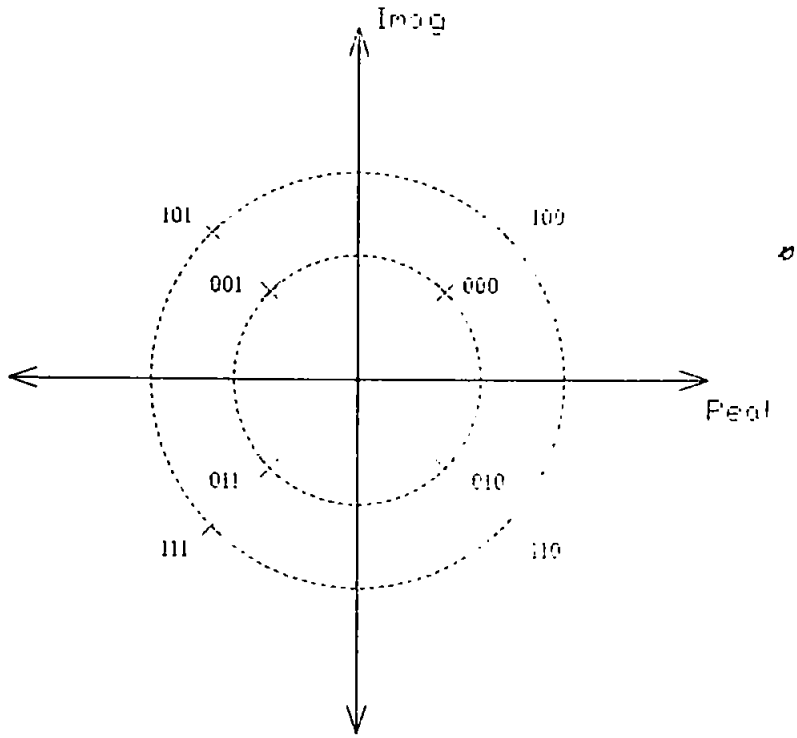


Fig 3.b Space Diagram Constellation for Quadrature Amplitude Modulation

3.1. The Maximum Likelihood Demodulator.

For a completely known signal in Additive White Gaussian Noise it is possible to implement a demodulator which minimizes the probability of making an decision error between the observed received signal r(t), and the transmitted signal s(t). Consider an M'ary signalling system with transmitted signal waveform $s_m(t)$ given by:-

$$s_m(t) = Re[\, u_m(t).\, e^{(j2.\pi.f.t)}] \qquad m= 1,2,..M \qquad \text{(Eq3.1.a)}$$

where $u_m(t)$ represent the low-pass equivalent signal waveforms. The M signals have signal energies individually defined by <13>:-

$$E_m = \int_0^{\tau} s(t)^2\, dt \qquad \text{(Eq3.1.b)}$$

$$= \frac{1.}{2} \int_0^{\tau} |u_m(t)|^2\, dt \qquad \text{(Eq3.1.c)}$$

The transmission of this signal through a channel results in the received signal being attenuated by a factor 'α' ,delayed by 'T_0' and corrupted by Additive White Gaussian Noise (AWGN), represented by n(t) which is the complex representation of frequency shifted bandpass noise. This results in the received signal r(t) being represented by:-

$$r(t) = Re[\, \{\alpha.u_m(t-T_0)e^{(-j2\pi fT_0)} + n(t)\}e^{(j2\pi ft)}] \qquad \text{(Eq3.1.d)}$$

the factor $e^{(-j2\pi fT_0)}$ represents a constant phase delay between the transmitted signal and the received signal. Assuming that the demodulator is synchronized to the

transmitter the dependance of the received signal on 't' can be eliminated. This results in the the received signal being expressed in the equivalent lowpass form:-

$$r(t) = Re[\ \alpha.u_m(t)e^{-j\theta}+n(t)] \qquad \text{(Eq3.1.e)}$$

where $\theta = 2\pi fT_0$ radians. The demodulator must therefore observe the signal for a time t such that $0<t<T$ and decide from this observation which of the M signal has been transmitted. It is desirable for the probability of an error in this decision to be a minimum. This occurs if the demodulator selects the signal with the largest posterior probability i.e. it computes the probability of signal 'm' being transmitted given the signal r(t) is received, during the observation interval. This can be expressed as:-

$$p(m|r(t),0<t<T)\ m=1,2,..M \qquad \text{(Eq3.1.f)}$$

It can be shown (Proakis <13>) that for a set of M signals, of equal likely a-priori probability equal to 1/M, with AWGN the demodulator must compute the M decision variables:-

$$D_n = Re[\ e^{(j\theta)} \int_0^T r(t).u_n^*(t)dt\ ] - \alpha.E_{n=1,2..M} \qquad \text{(Eq3.1.g)}$$

where $u_n^*(t)$ = is complex conjugate of the variable $u_n(t)$ and decide in favour of the largest D_n, and hence signal m. If however, the signalling waveforms are of equal energy then the $\alpha.E$ term is a constant and can be neglected in the computation and hence becomes:-

$$D_n = Re[\ e^{(j\theta)} \int_0^T r(t).u_n^*(t)dt\ ] \qquad \text{(Eq3.1.h)}$$

For a Quadrature Amplitude Modulated signal, u(t) is given by:-

$$u(t) = \sum_{n=0}^{n=\infty} In.\ g(t-nT) \qquad \text{(Eq3.1.i)}$$

where $I_n = I_{nr} + jI_{ni}$,and I_{nr} and I_{ni} are two information carrying symbols which can take on any one of k states. If I_{nr} and I_{ni} can only take on the value +/- 1 , then this is equivalent to Quadrature Phase Shift Keying (QPSK). If the I_{ni} symbol is set to zero then Binary Phase Shift Keying results. The function g(t) represents the low-pass pulse shaping function and is one factor that restricts the bandwidth of the transmitted signal. The transmitted signal s(t) therefore becomes:-

$$s_m(t) = Re[\ g(t).\ e^{(j2\pi ft + \theta_d + L)}] \qquad \text{(Eq3.1.j)}$$

$\theta_d = Tan^{-1}(I_{ni}/I_{nr})$ adjusted for a full 360 degrees.

$\theta_d = \Gamma^{-1}(I_{ni}/I_{nr})$

for I_{nr}, I_{ni} = +/- 1 for QPSK, and I_{nr} =+/- 1, I_{ni} = 0 for BPSK. L is a phase offset on the resultant constellation, an example of which can be seen in fig 3.1.a. Assuming g(t) is real then :-

$$s_m(t) = g(t).\{Cos\ (\ 2\pi ft + \theta_d + L\ )\} \qquad \text{(Eq3.1.k)}$$

by simple trigonometry :-

$$s_m(t) = g(t)\{A_p.Cos\ 2\pi ft + A_q.Sin\ 2\pi ft\} \qquad \text{(Eq3.1.l)}$$

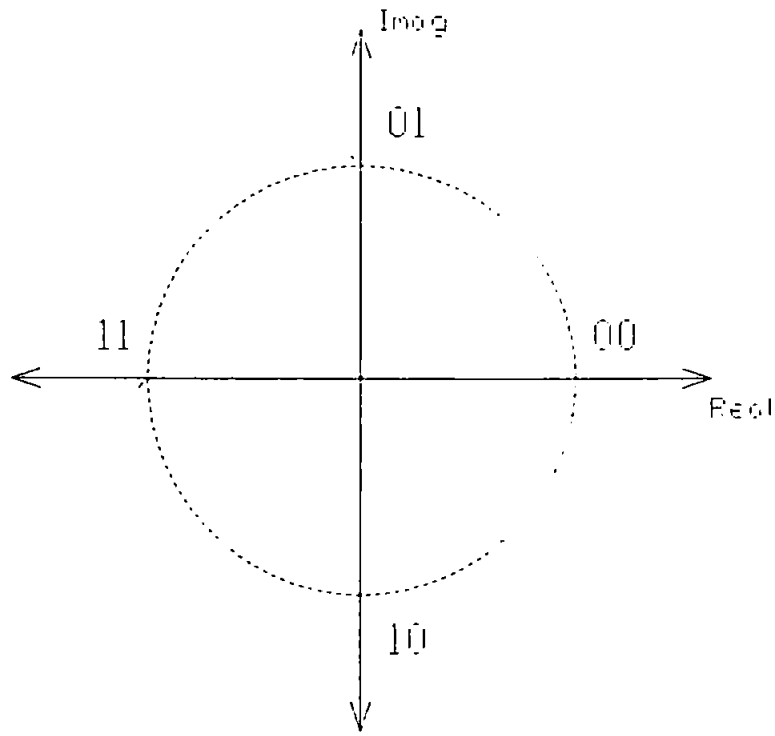


Fig 3.1.a Space Diagram for QPSK with Phase Offset L=45 deg

where

$$A_p = \mathrm{Cos}(\theta_d + L)$$
$$A_q = \mathrm{Sin}(\theta_d + L)$$

hence the transmitted signal can be constructed from the addition of two quadrature carriers individually modulated by the shaped information stream I_{ni} and I_{nr}.

Maximum Likelihood Demodulation of this transmitted signal in the presence of noise results in the computation of the decision variable Dm, by substitution of equation 3.1.j. into 3.1.h produces:-

$$D_n = e^{(j\theta)} \int_0^T r(t).g^*(t)e^{(-j\theta_d + L)}\, dt \qquad \text{(Eq3.1.m)}$$

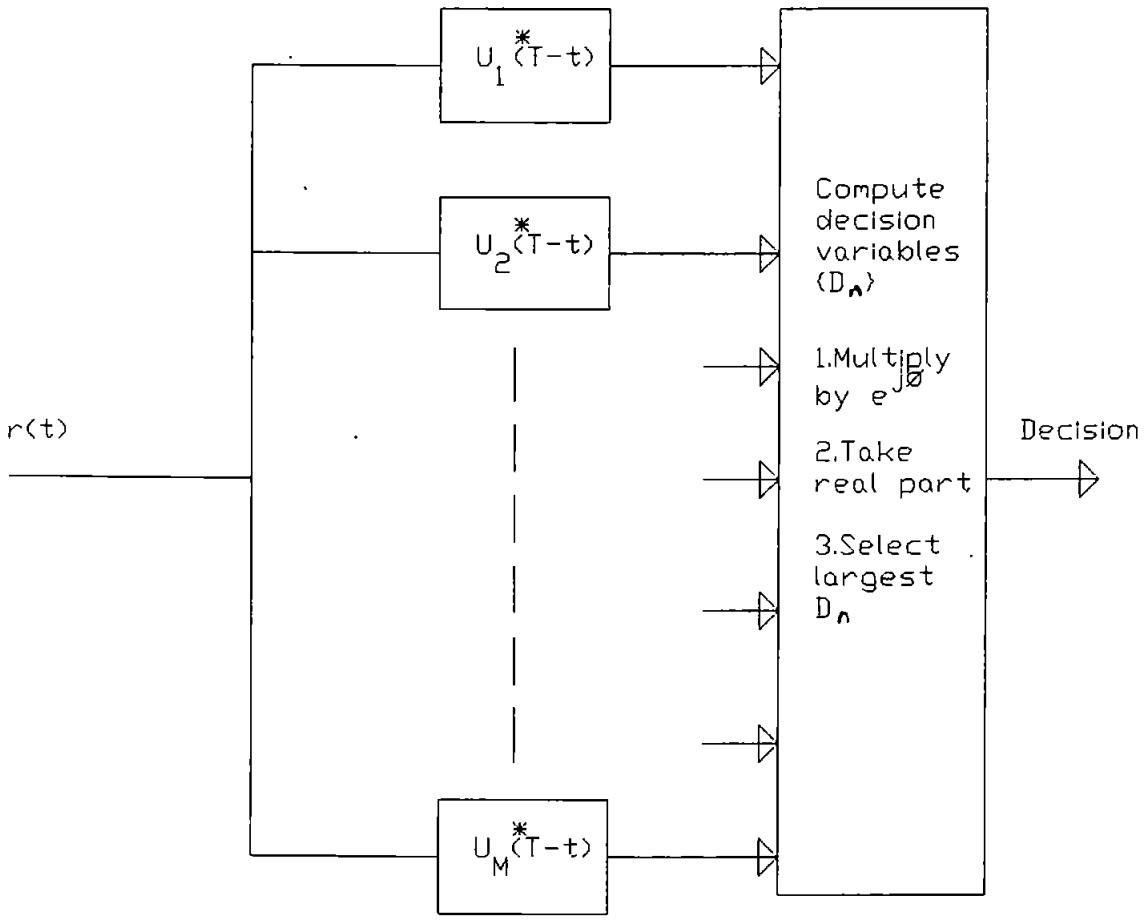
The exponent under the integral is independent of the integration variable and hence can be factored out of the decision variable.

$$D = e^{(j\theta)} \int_0^T r(t).\, g^*(t)\, dt \qquad \text{(Eq 3.1.n)}$$

This can be implemented as a single matched filter or cross correlator. The resultant vector D can be phase processed to obtain a received phasor θ_{rd} which is then compared against the set of transmitted phasors produced by $\{s_m(t)\}$.

The optimum demodulator must therefore compute this complex decision variable D from the received signal r(t) during the interval $0<t<T$, convert from a complex to polar form and compare the resultant phase against a set of possible transmitted phasor. This can be achieved by a system as shown in fig 3.1.b. The main objective of the research has

been to implement the above processes with increased accuracy in both the transmitter and receiver, compared to previous methods while carefully shaping the transmit spectrum by the correct selection of the function g(t).



Matched Filter Demodulator

Fig 3.1.b Optimum Demodulator Implementation

3.2. Binary Phase Shift Keying.

If we consider a binary random waveform in a non return to zero format (NRZ) such that a logical 0 is represented by a -1, and a logical 1 by a +1, the space diagram can be displayed as two points 180 degrees apart Fig 3.2.a. For an analysis of this waveform we can consider it as a series of random impulses convolved with a rectangular impulse response of period + and - T/2 (where T is the symbol period) and amplitude of value 1. As the impulse sequence is random (or pseudo-random) it's frequency spectrum is ostensibly uniformly flat over the bandwidth of interest and of normalized amplitude equal to 1, hence the convolution of the two time responses, now represented by the multiplication of the two frequency responses, produces a power spectrum identical to that of a rectangular impulse response. From the Fourier transform this is found to be that of the Sinc function as shown in Fig 3.2.b

$$| V(f) | = \frac{T\ (\mathrm{Sin}\ \pi fT)}{(\pi fT)} \qquad \text{(Eq3.2.a)}$$

by observation of fig 3.2.b it can be seen that the spectra of the BPSK signal has major components >-50dB out as far as 100 times the symbol rate (single sided spectrum) and 99% of the power is contained within 20 times the symbol rate (double sided spectrum) <6>. Hence this means that very large bandwidths would be required to transmit the whole BPSK spectrum without loss or distortion and hence makes the transmission of a NRZ format undesirable. To overcome this restriction the spectrum of the NRZ symbol stream is shaped by filtering. This introduces new problems. Filtering of the signal with little care produces Inter-Symbol Interference produced by the convolved impulse

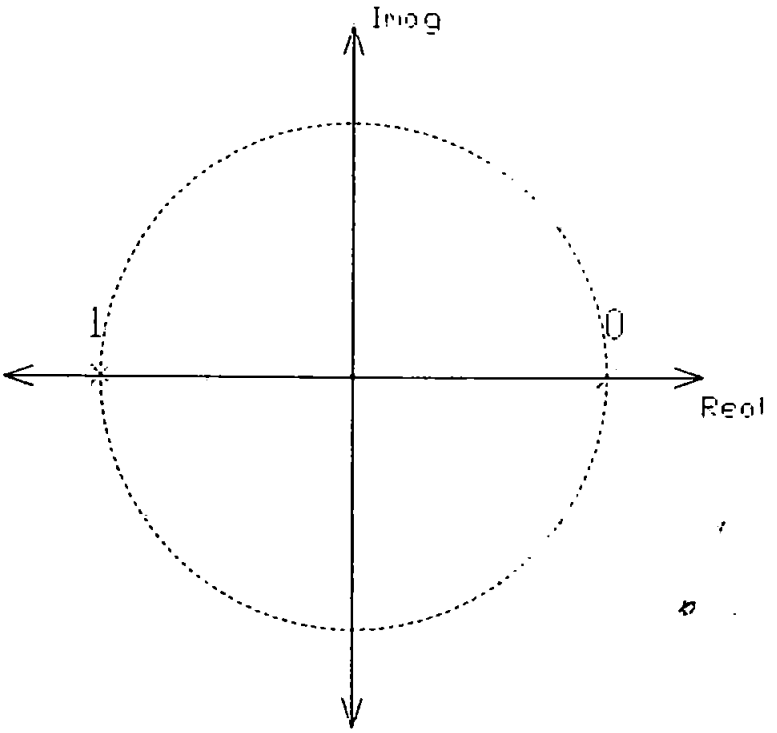


Fig 3.2.a Space Diagram for a BPSK signal

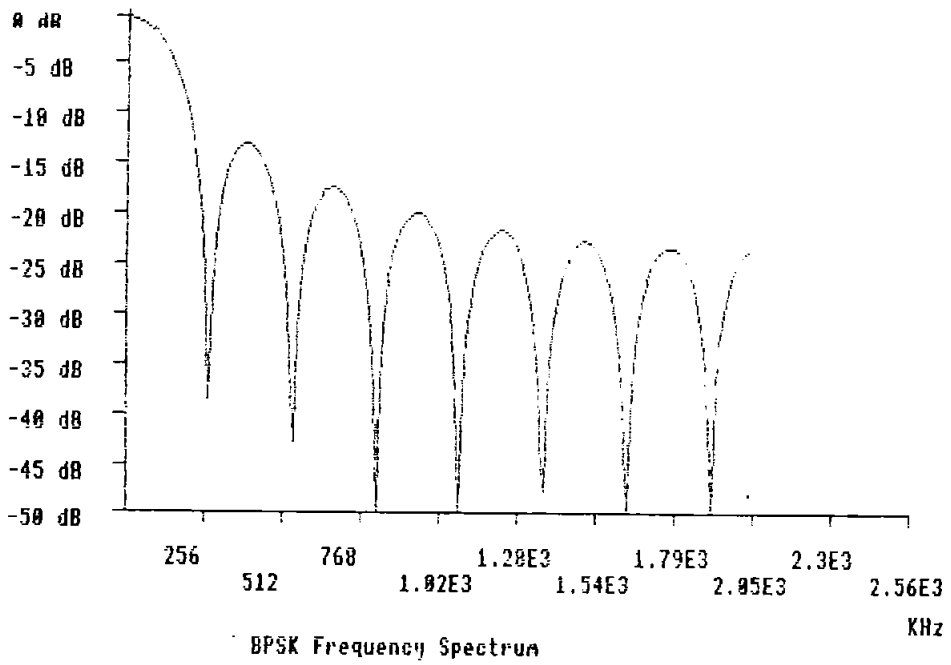


Fig 3.2.b Frequency Response for an Unfiltered BPSK Signal

responses of the NRZ pulses and the filter. This is manifest in the form of impulse response tails adding to the main lobe at the sampling instant resulting in a closure of the Data Eye an example of which is shown in Fig 3.2.c.

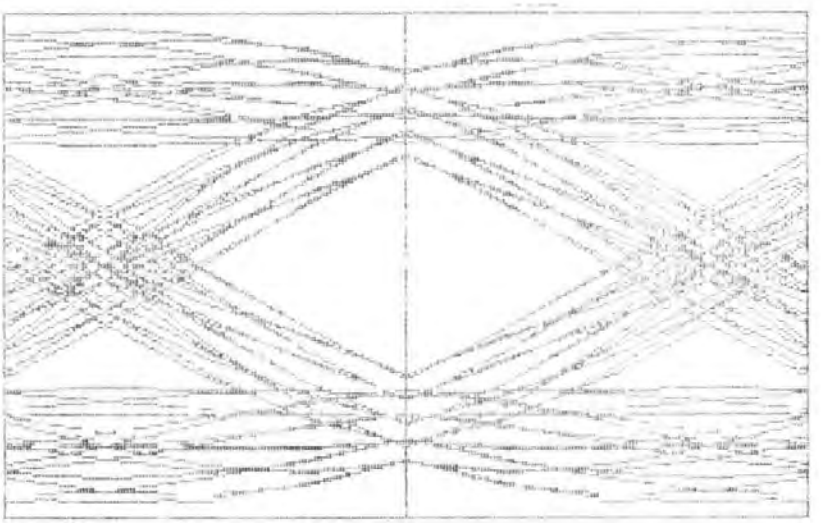


Fig 3.2.c Example of a Data Eye Diagram with Inter-Symbol Interference

3.3. Nyquist Filter Criteria

Nyquist <1> proposed that to overcome this problem the spectrum of a filter must conform to a specific function related to the symbol rate, 2fo:-

$$H(f) = \begin{cases} 1 + Y(f) & f < fo \\ Y(f) & fo < f < 2fo \\ 0 & f > 2fo \end{cases} \qquad \text{(Eq3.3.a)}$$

Where Y(f) is a real function that is even symmetric about f=0 i.e

$$Y(-f) = Y(f) \quad f < 2fo \qquad \text{(Eq3.3.b)}$$

and odd symmetric about f=fo such that:

$$Y(-f+fo) = -Y(f+fo) \quad f < fo \qquad \text{(eq3.3.c)}$$

there is no ISI if the symbol rate is 2fo. A simple example of this symmetry can be seen in fig 3.3.a, but a commonly used form of the Nyquist Filter is that of the Raised Cosine Filter.

3.4. Raised Cosine Roll-off Filters

The family of Raised Cosine Roll-off filters is defined by:-

$$X(f) = \begin{cases} 1 & 0 < f < (1-\beta)/2T \\ 1/2\left(1-\sin(\pi(f-1/2T)/\beta)\right) & (1-\beta)/2T < f < (1+\beta)/2T \\ 0 & f > (1+\beta)/2T \end{cases} \qquad \text{(Eq3.4.a)}$$

with the time response x(t) being given by:-

$$x(t) = \frac{\sin \pi t/T}{\pi t/T} \frac{\cos \beta \pi t/T}{1-(2\beta t/T)^2} \qquad \text{(Eq3.4.b)}$$

The roll-off factor β has a range from 0 to 1 and is often called the percentage roll-off of the filter i.e. 0% and 100%. In all cases the zero crossings occur at the symbol rate period. The Raised Cosine filters have a symmetrical time response around t=0 and hence have a constant phase response, i.e a zero imaginary component. It can be seen from fig 3.4.a that the 0% filter is equivalent to that of a rectangular frequency response and represents the minimum bandwidth that can be used to transmit a signal of symbol rate 2fo symbols/sec without degradation. This produces a sinc function time response which although it complies with the Nyquist filter criteria can produce implementation problems and therefore produces worse ISI. If the symbol timing at the receiver is not exact the data eye is sampled away from the point of zero ISI. Due to the large overshoots that the sinc function produces the ISI is increased and hence the recovered signal is degraded . As the percentage roll-off is increased the impulse response tails reduce more rapidly and hence the amount of ISI can be reduced by the use of a higher percentage roll-off impulse response. Hence symbol timing must be more accurate for low percentage roll-off factors. An analogue approach to the frequency response is hard to practically realize since the filters are usually minimum phase filters with causal impulse responses of very long duration and difficult to phase equalize. Hence by the appropriate choice of roll-off factor we can trade ISI, complexity of symbol timing recovery, and filter realization problems, for increases in transmit bandwidth.

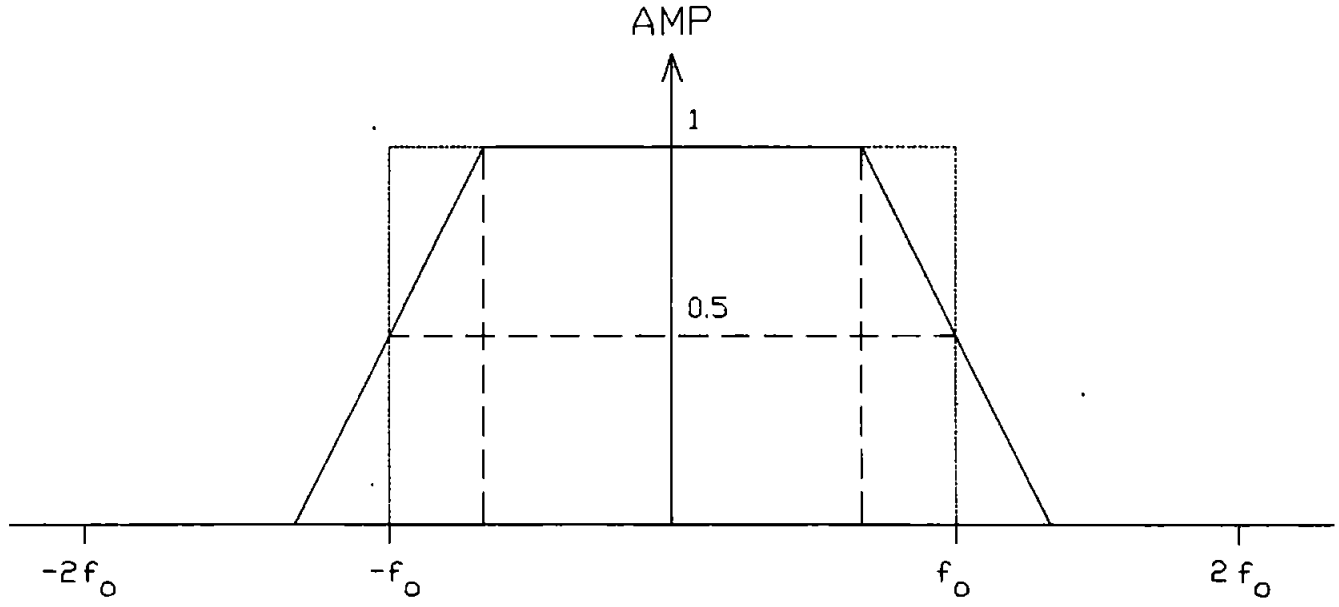


Fig 3.3.a Simple Example of a Nyquist Filter Frequency Spectrum

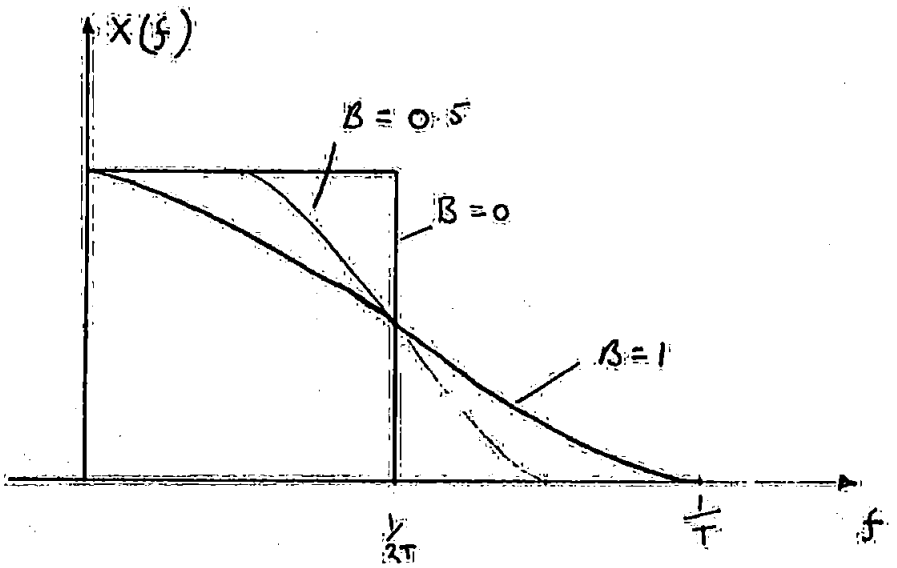


Fig 3.4.a Family of Raised Cosine Filter Frequency Responses

3.5. Matched Receive Filters

In the case of digitally derived signals where the transmitted waveform is known and easily defined it has long been known <6> that it is possible to filter the received signal in such a way that, in the presence of additive white gaussian noise, the output signal can be maximized against the noise.These Matched Filters require the filter impulse response to be the 'mirror image' of the transmitted time signal.If we consider a Raised Cosine filter impulse response its mirror image is also a raised cosine filter impulse response due to its inherent symmetry and hence its zero phase response. Hence the matched filter for a Raised Cosine filter is in itself a Raised Cosine filter. However the spectral response of the channel is now that of a (Raised Cosine Filter)^2 which no longer exhibits zero crossings at the sampling instant. We therefore require a set of filters that exhibit an overall response of a Raised cosine filter while being matched at the receiver. Lucky, Salz and Weldon <7> have shown that the optimum solution is that the transmitter and receiver filters are designed such that their overall response is:

$$H_R(f).H_1(f).H_2(f)= (H_N(f))^{1/2} \qquad \text{(Eq3.5.a)}$$

Where subscript N indicates Nyquist filter, R receiver filter and 1 & 2 any other filters in receiver processing. This filter response is repeated in the transmitter. Therefore to have a Root Raised cosine Filter in both the transmitter and the receiver preserves the overall Nyquist frequency response and the Matched filter criteria.

3.6. Design and Development of Multi-Rate Data Modem

3.6.1. Introduction to Design Requirements.

It has previously been shown that a digital implementation of a phase demodulator can be used to detect a variable phase signal in a continuous format with the output from the phase decode PROM being an exact representation of the received vector relative to the receiver local oscillator. It has been postulated that this could be extended to a discrete binary signal format by the use of a symbol timing recovery function. It has also been shown that one optimum form of filtering in a binary signalling channel for both transmitter and receiver are that of Root Raised cosine filters which can simultaneously comply with both the Nyquist filter and Matched filter criteria. This results in the design for the data modem to become that in fig 3.6.1.a. Due to the digital nature of the proposed system, it is envisaged that by a simple re-configuration of the modem, data rates ranging from 16Ksymbols/sec to 256ksymbols/sec can be achieved. To obtain minimum degradation through the system, all filters must be highly accurate in both the time and frequency domains. Due to the symmetry of the root raised cosine filter impulse response, the theoretical filter is non causal and difficult to realize using standard LC networks, or active filters.

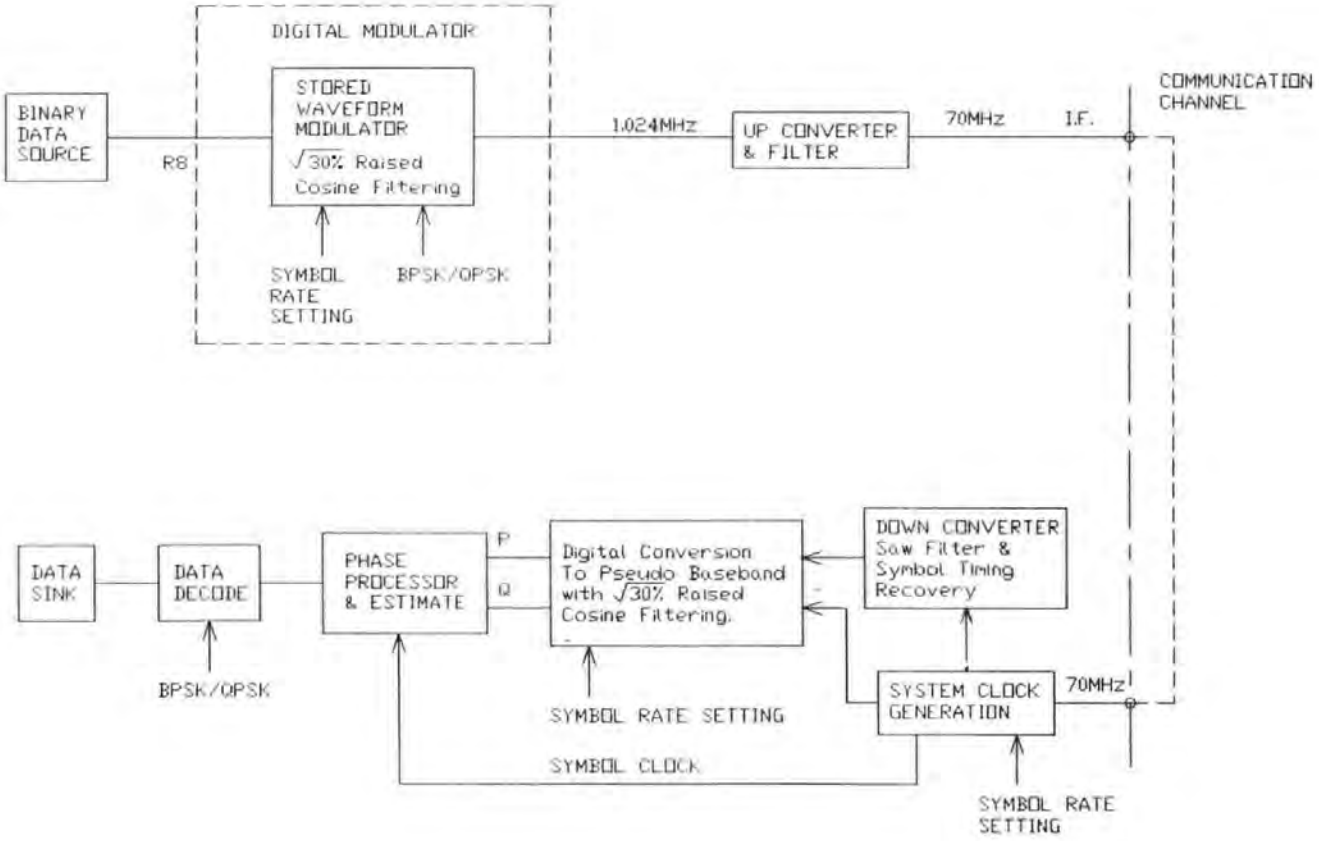


Fig 3.6.1.a Overall System Design for Multi-Data Rate Modem

3.7. Multi-Rate Modem Modulator.

In the discussion so far attention has concentrated on the receiver aspect although it has been proposed that a Root Raised Cosine filter should be used at the transmitter for efficient spectral shaping. The theory and implementation behind the modulator will now be explored.

3.7.1. The Modulator; A Brief Overview.

Consider a previously scrambled,hence pseudo-random data stream being presented to a modulator where by it will be converted into a QPSK signal format and fed to frequency conversion circuitry on a 1.024MHz First IF carrier. The conventional approach has been described by Feher <9>,Couch <6>,Proakis <13> and many other authors with spectral shaping either taking place in the baseband channels, or at the carrier frequency, using analogue filters. The basic block diagram can be seen in fig 3.7.1.a. The incoming data sequence is multiplexed into two data sequence at half the incoming data rate. These two parallel data sequences of NRZ pulses are then filtered for efficient spectral shaping before modulating onto a IF carrier in quadrature.

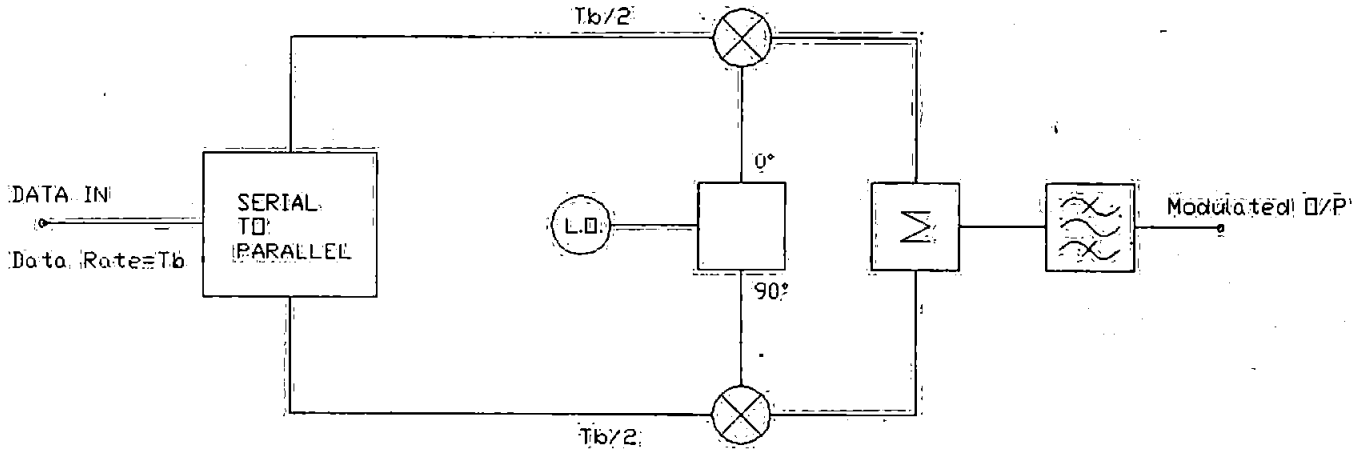


Fig 3.7.1.a Typical QPSK Modulation Method

3.7.2. Background to Digital Filters.

Consider a simple system as shown in fig 3.7.2.a. The input waveform x(t) with spectrum X(f) is filtered by the filter with transfer function H(f) to produce an output waveform y(t) of spectrum Y(f) <8>. This can be expressed as :

$$Y(f) = H(f).X(f) \qquad \text{(Eq3.7.2.a)}$$

If we consider the same system in the time domain we obtain:-

$$y(t) = x(t)*h(t) \qquad \text{(Eq3.7.2.b)}$$

where * represents the convolution of two time responses and h(t) the impulse response of the filter H(f).i.e.

$$y(t) = \int_{-\infty}^{\infty} h(T).x(t-T)dT \qquad \text{(Eq3.7.2.c)}$$

Using the sampling theorem and by sampling at least twice the highest frequency the discrete time representation becomes:-

$$y(kT) = \sum_{n=-\infty}^{n=\infty} h(nT).x((k-n)T) \qquad \text{(Eq3.7.2.d)}$$

hence if this mathematical function can be performed on the sampled data sequence the output sample data is effectively filtered. This function can therefore be modelled in digital hardware and repeated to any degree of accuracy required.

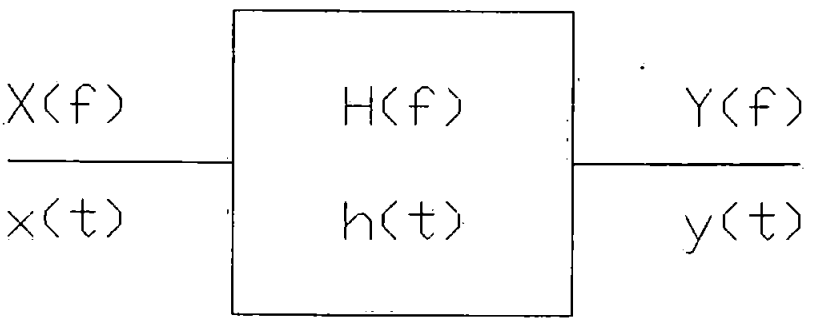


Fig 3.7.2.a Simple Linear Time Invariant Filter System

3.7.3. Digital Transmit Filters.

In the transmitter the incoming data pulses are of the NRZ format and hence only ascribe to two values +1 and -1.The convolution process is therefore limited to:

$$y(kT)= \sum_{n=-\infty}^{\infty} h(nT).x((k-n)T) \qquad \text{(Eq3.7.3.a)}$$

where x(iT)= +1 or -1,k=0 to ∞ and can be approximated by Eq3.7.3.b for a finite length of h(nT).

$$y(kT)= \sum_{n=0}^{N} h(nT).x((k-n)T) \qquad \text{(Eq3.7.3.b)}$$

hence only a limited number of coefficient multiplications and additions need to take place. Consider the data sequence not as a series of NRZ pulses but as a series of pseudo-random impulses with a flat normalized spectral response.The output spectrum from any filter is therefore defined by the filter spectrum only and hence its impulse response. The digital filter now consist of the convolution of two impulse responses. This can be considered as in fig 3.7.3.a for a filter impulse response which is only significant over 6 symbol periods for a Root 30% cosine roll-off filter (Appendix B). In the centre of the sequence the tails from all the filter impulse responses converge. For this region of interest it is possible to define the intermediate results for any input sequence of 6 symbols i.e. calculate the convolution results and store them in memory. If the sampling rate of the filter impulse response has been correctly chosen with a 6 symbol input word it is possible to store all combinations of results in a PROM. Since the two quadrature channels are both filtered by the same frequency response it is possible to use the same information for both channels. However care must be

taken such that the time sample chosen by each channel corresponds to the correct filter response and that no phase shifts exist between channels.

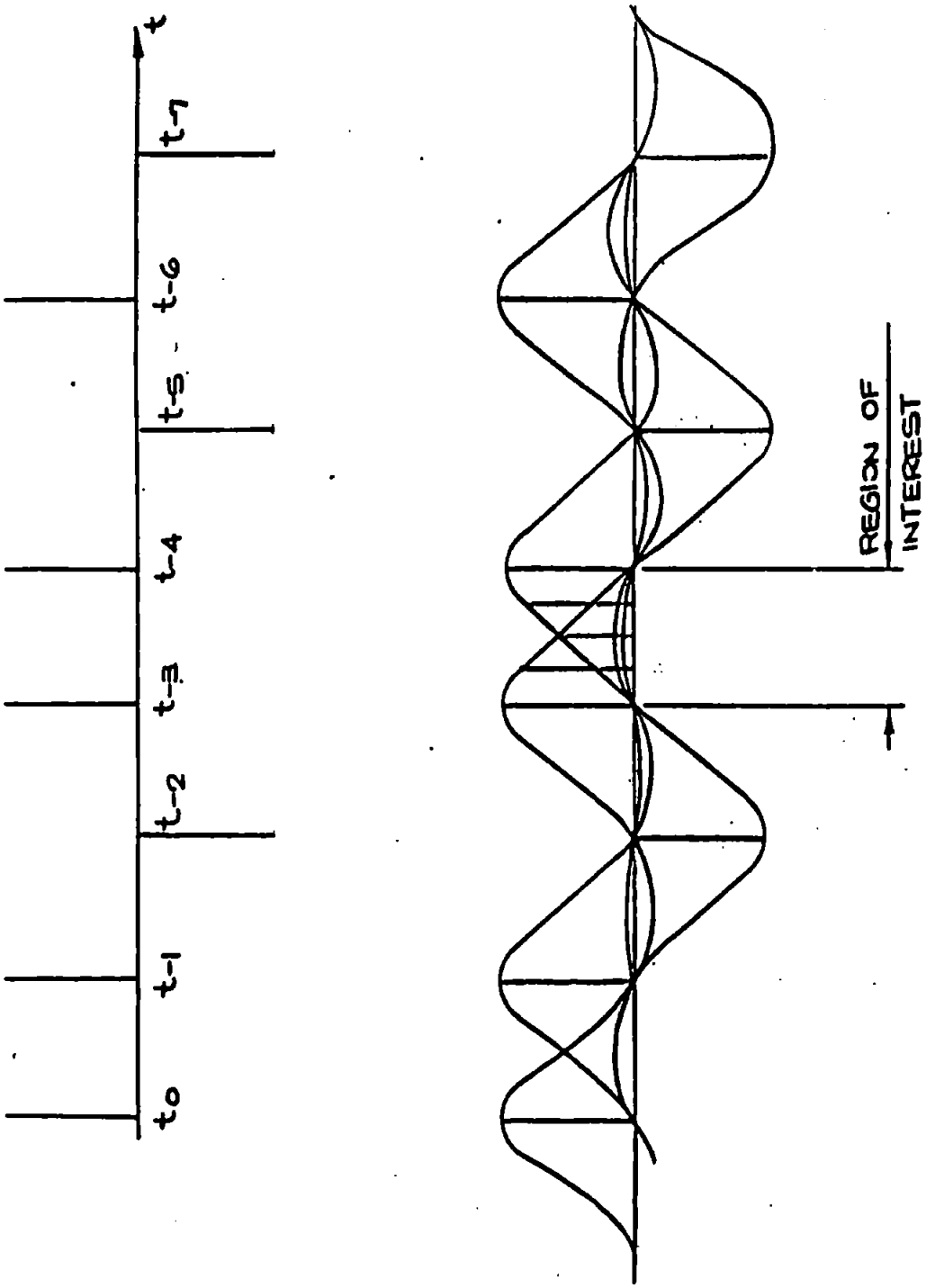


Fig 3.7.3.a Convolution of Data Impulse Stream with Filter Impulse Response

3.7.4. Digital Up-conversion to First IF.

Consider a sequence of 4.096MHz samples, of infinitesimal pulse width, output from the above digital filter PROM, fig 3.7.4.a and a carrier frequency of 1.024MHz is to be used. The sampled In-Phase carrier waveform can be consider to be as in fig 3.7.4.b. If these two waveforms are multiplied together the result is as in fig 3.7.4.c i.e. a 4.096MHz sample stream with alternate samples equal to zero. If another sample stream is multiplied by the quadrature carrier as shown in fig 3.7.4.b the resultant is as shown in fig 3.7.4.d. It can be clearly seen that again a 4.096MHz sample stream results, with alternate samples equal to zero. For the corresponding positions in the In-phase channel when the quadrature channel is zero the In-phase is non-zero and vice versa. The output modulated carrier signal therefore consists of the In-phase and quadrature channels added together which can be implemented by simply multiplexing the two channels. With the setting of the phase offset, L, for the data constellation, the modulated carrier signal can be synthesised within the Prom and hence carrying out both with the filter and modulator functions. The complete modulator circuit becomes that of fig 3.7.4.e. For a BPSK signal format the incoming data is only diverted to the In-phase channel and the quadrature channel output is set to zero. Other data rates can be accommodated by scaling the number of samples per symbol or the sampling rate. Scaling of the sampling rate causes the IF output frequency to change but this can be adjusted for by the use of frequency synthesisers in the analogue frequency up and down converters.

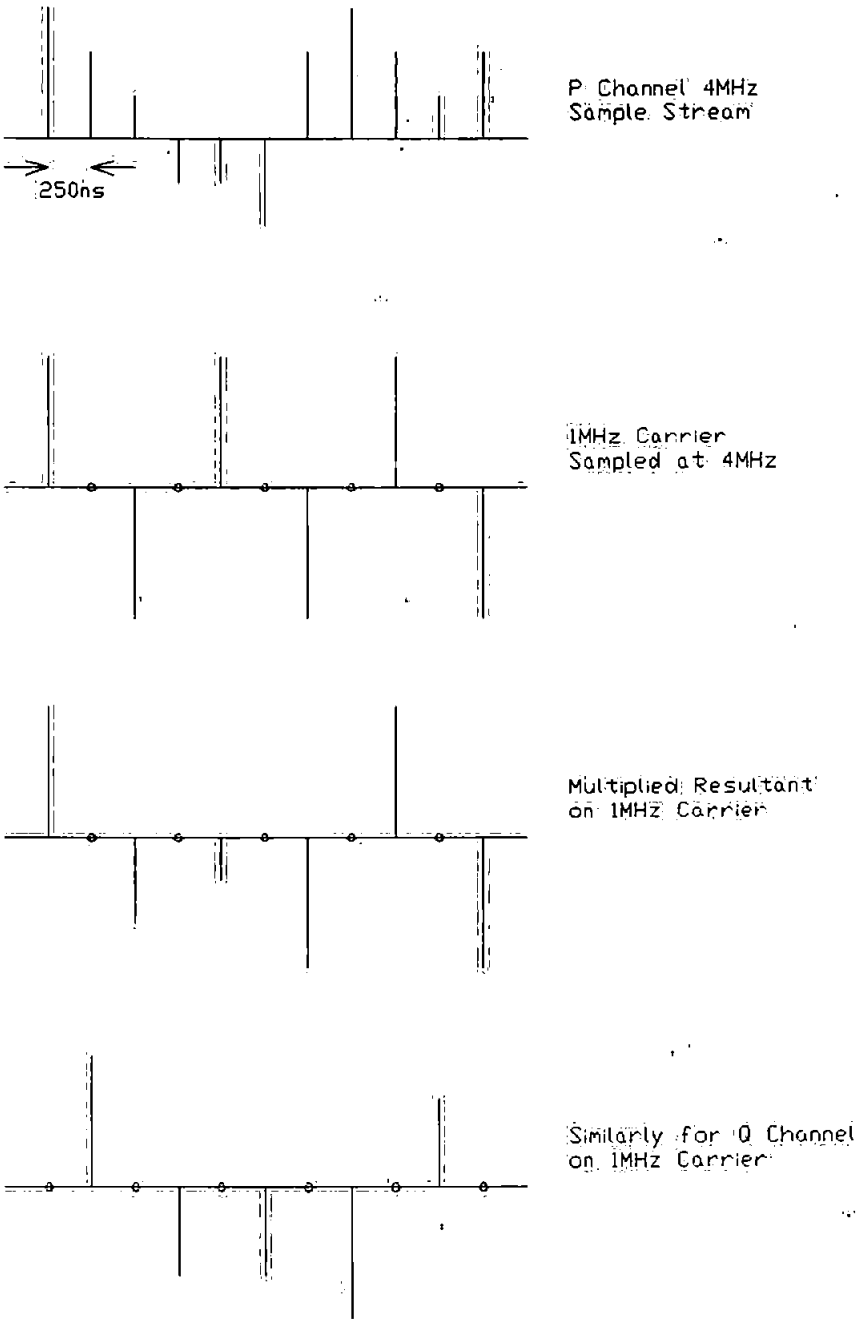


Fig 3.7.4.a/b/c/d An Example of the Quadrature Modulation Process in a Sampled Data System

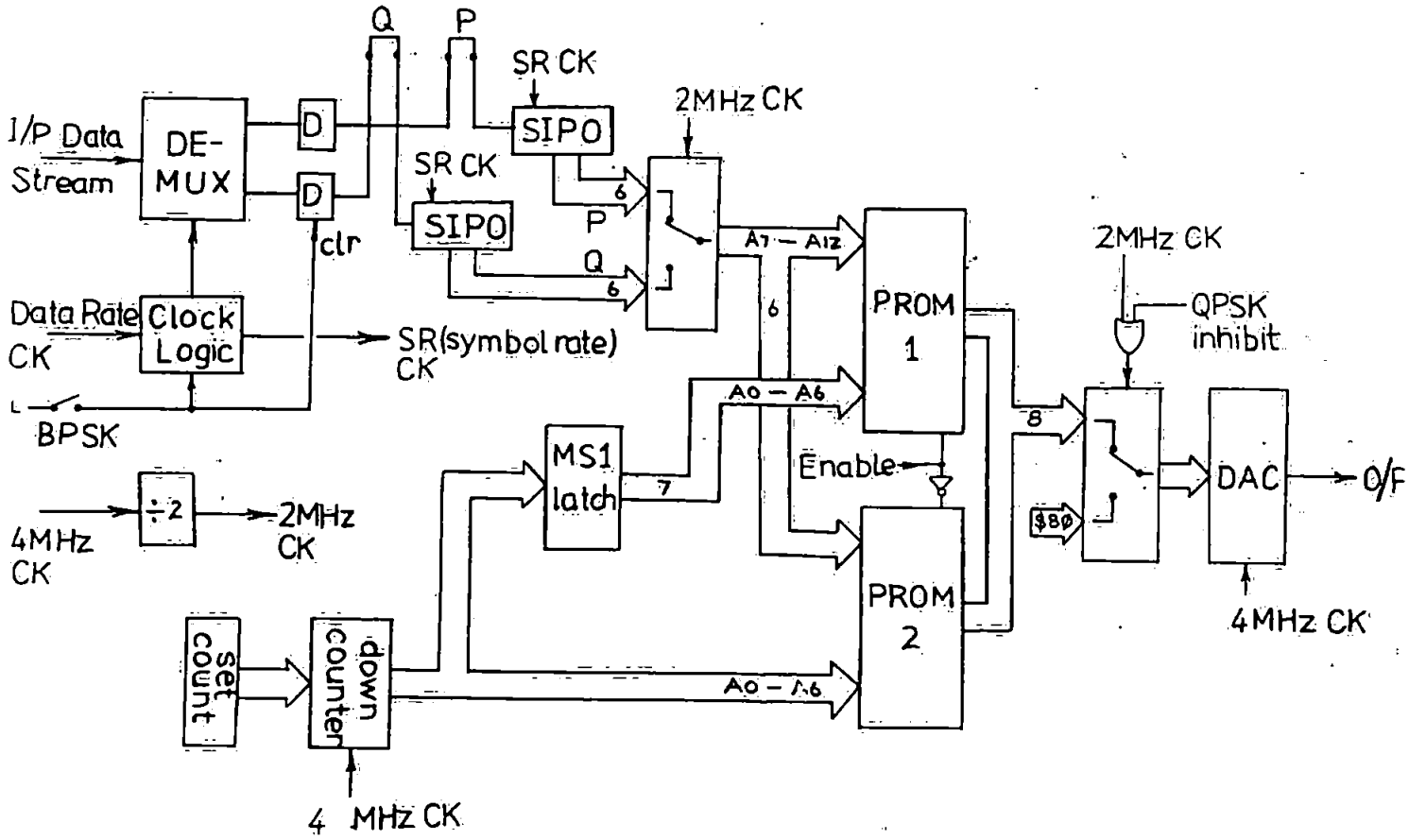


Fig 3.7.4.e Digital Modulator System Diagram

3.7.5. Effects of Sinc Function on Modulated Signal Frequency Spectra

The Modulator is designed to produce a shaped bandpass frequency response that conforms to the root 30% Raised Cosine spectrum. In the example given above the samples are of infinitesimal pulse width, as this width is increased a Sinc function, in the frequency domain corresponding to the pulse width, is superimposed over the desired frequency response. The response is therefore degraded by the effect of the sample and hold function produced by the DAC. The output sample rate is set at 4.096MHz and will therefore produce a (SIN (X))/X (Sinc function) response around 0 Hz with a first null at 4.096MHz. This response multiplies the required raised cosine response and a degraded frequency response is the result. This sinc function and its effect can be seen in fig 3.7.5.a. The effect of this asymmetric slope across the passband is to create cross quadrature interference. In the region of interest that the raised cosine response passband extends i.e.+/- 128KHz around 1.024MHz for a 256Ksymbol/sec data rate then the slope across the passband is 0.5dB. This could be removed by a number of different techniques:-

1) Increasing the sample rate to carrier frequency ratio throughout the system.

2) Equalise the asymmetry by the use of a digital equalization filter.

3) Decrease the hold time of the DAC by inserting a zero sample between each calculated output sample i.e. increase the output sample rate.

4) Consider the effect of the asymmetry to be negligible over the frequency range of interest.

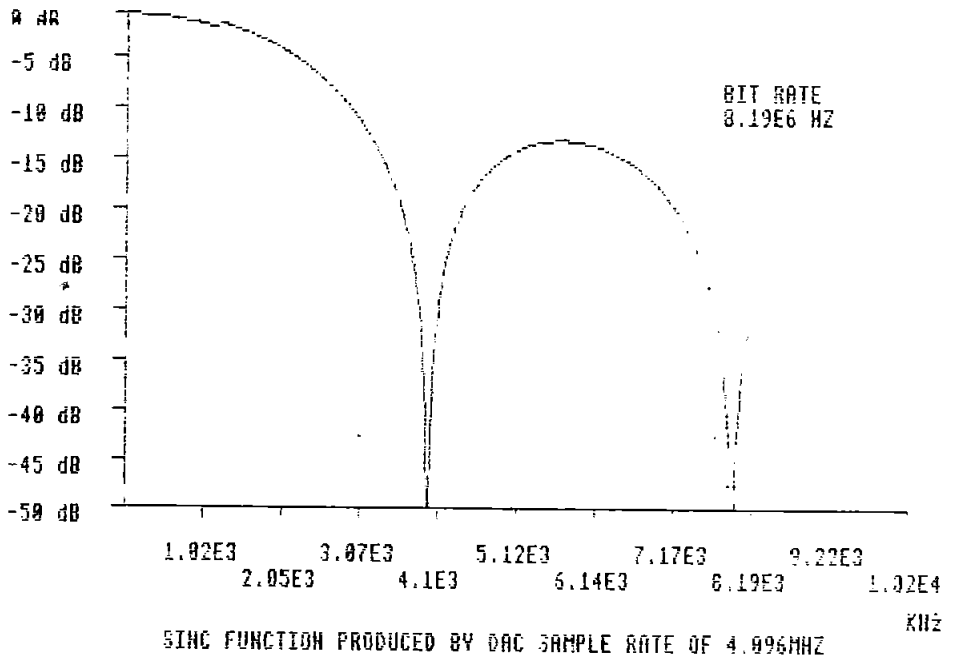


Fig 3.7.5.a Effect of Sampling on Output Frequency Response

Solution 4 was considered the most appropriate for this investigation with solution 3 favoured over 1 and 2 as this required a minimal amount of hardware. Solution 1 will increases the memory size of the PROM's to cover the required symbol rate range while also increasing the operating speed with solution 2 increasing the complexity of the modulator.

3.8. Multi-Rate Digital Demodulator.

From the previous discussions of the modulator it can be postulated that a demodulator can be based on a similar concept. The digital demodulator proposed consists of an input signal on a minimum carrier frequency of 1.024MHz being digitally converted to baseband in quadrature and then digitally filtered. Sample rate reduction will be used to enable Root 30% raised cosine filters to be implemented practically while covering a span of symbol rate from 16Ksymbols/sec to 256Ksymbols/sec. The final output sample rate from these filters is four times the symbol rate. From these four samples per symbol, one of the samples must be extracted as the peak sample instant corresponding to zero ISI (when in conjunction with a matched transmitter filter) and the center of the data eye where the SNR is a maximum. A coherent Carrier phase reference with the 1MHz IF is also required to enable data to be correctly decoded from the received phase angle. To enable both of these processes to occur the symbol timing must be recovered and a carrier reference established.

3.8.1. System Implementation.

The initial bandlimiting of the received signal to prevent aliasing prior to sampling and digitizing, is obtained by the use of a SAW filter having a symmetrical impulse response and hence good phase linearity, producing minimal group delay variations. The incoming signal, in the range 52-88MHZ, is first up-converted to 200MHz, the frequency being mainly dependent on the availability and cost of SAW filters and then bandlimited to a 700 KHz bandwidth. The bandlimited signal is then down converted to a 1.024MHz minimum carrier frequency before being digitally sampled by a 6 bit Analogue to Digital converter giving a mean-squared signal to quantization noise ratio of 31dB <5>. This is considered to be sufficiently below the noise floor of the received bandpass signal under normal operating conditions, approximately 4 dB SNR in 700 KHz for a 256 Kbit/sec signal with Eb/No of 8 dB, to produce no degradation. At higher signal to noise ratios the degradation will limit the final performance of the system but due to the steepness of the theoretical error curves at high signal to noise this degradation will be insignificant. The sampling takes place at a 4.096MHz rate to fulfil sampling theorem requirements and allow synchronization with the symbol clock. In practice it was found that a choice of sampling rate of two or three times the carrier frequency caused aliasing due to the spectrum extending out to 1.8 MHz for the particular SAW filter used. This sampled signal is then digitally mixed to baseband before filtering.

3.8.2. Digital Quadrature Down-Conversion

The method used for down-conversion to baseband is obtained by a similar argument as in the modulator. Assume that the incoming signal is on a carrier locked to a quarter of the sampling frequency. Down converting with a sampled 'local oscillator', synchronous with the sampling frequency such that the sampling points occur at the 0,90,180,270 degree points, produces alternate zero value samples in the Phase and quadrature baseband channels. This can be implemented by the multiplication of sample pairs by 1.024MHz local oscillator i.e. +1 and -1 which is implemented by complementing samples, demultiplexing these samples into the phase and quadrature channels, and adding a zero value sample in the opposite channel while one channel is being enabled. This zero term would normally be a product of the mixing process. The addition of the alternate zero term into a channel when the data sample is diverted into the other channel results in two baseband 4.096MHz sample streams and allows sampling to occur at half the rate that would normally be required for the 4.096 MHz sample rates per channel while maintaining the correct mathematical model.

3.8.3. Receiver Digital Filter Design

The digital filter design problem is similar to that of the modulator, however, now the filtered sample sequence is multi- level, i.e. 64 level for 6 bit A/D. If the concept of the pre-stored response is to be used the size of memory device required becomes prohibitive (in the order of 2^{36}) and a new approach is undertaken.

3.8.3.1. Choice of Raised Cosine Roll-off Factor.

The roll-off factor for the overall channel spectrum was chosen to be 0.3 or 30%. This figure is a compromise between minimum channel bandwidth requirements and timing jitter tolerances on the symbol timing recovery circuitry giving rise to ISI. Higher roll-off factors i.e. 50%, produce wider bandwidths but a higher timing jitter can be tolerated before the ISI significantly degrades the data eye.

3.8.3.2. Basic Digital Filter Implementation.

The Root 30% Raised Cosine Filter that is required for the receiver is linear phase and must hence be of a Finite Impulse Response Design (FIR), fig 3.8.3.2.a. Here the delayed input samples are multiplied by the appropriate coefficients and the results summed to produce the convolved output. The digital hardware implementation of a FIR filter can be undertaken in a variety of ways, i.e. Parallel multiplication and addition fig 3.8.3.2.b, parallel multiplication and serial addition fig 3.8.3.2.c, parallel multiplication and cascaded addition fig 3.8.3.2.d. The format used in the designed filters was as shown in fig 3.8.3.2.e and can be termed a Direct Form FIR Filter with Serial multiplication and Addition. The incoming sample stream is stored along a delay line, of length 15T ,which is constructed of 6 bit latches, clocked at the input sample rate $f_{si} = 1/T$. If the frequency response of the filter is restricted to $f_{si}/8$, i.e frequency components higher than 1/8 input sample rate are attenuated more than 20 dBs, then it is possible to sub-sample the output of the filter at $f_{si}/4$. This causes the filter response to repeat around the output frequency and the aliasing point now occurs at 1/8th of the input

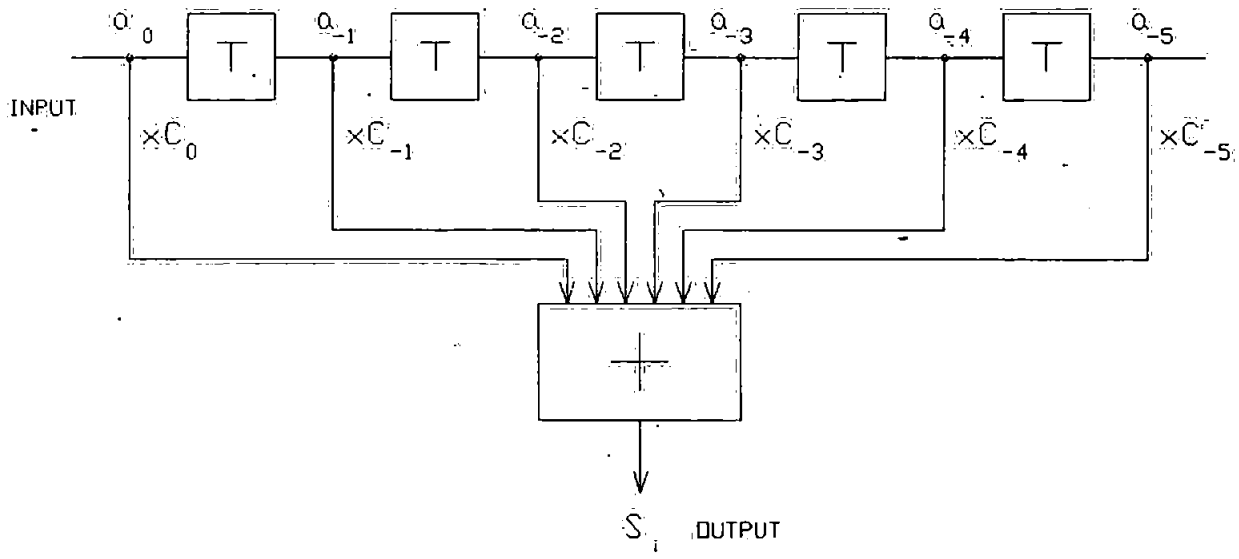


Fig 3.8.3.2.a Finite Impulse Response Filter Function Diagram

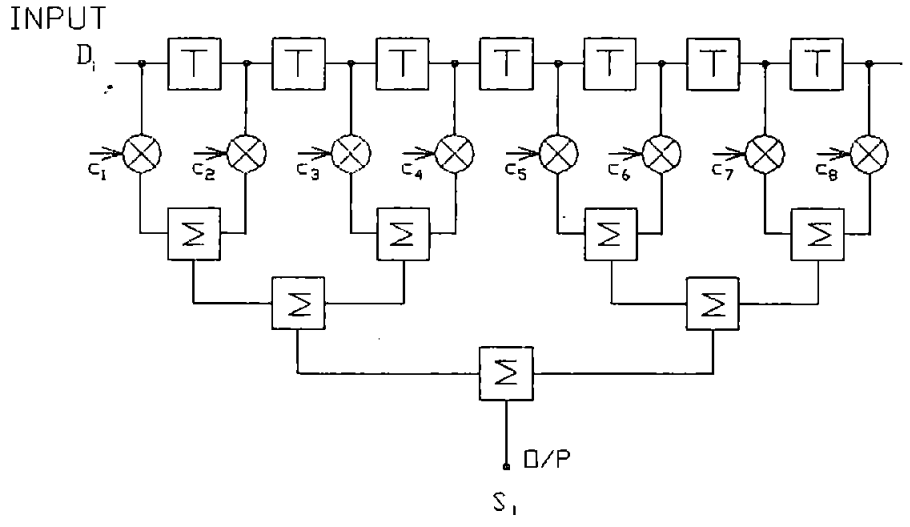


Fig 3.8.3.2.b FIR Filter Implementation with Parallel Multiplication and Addition

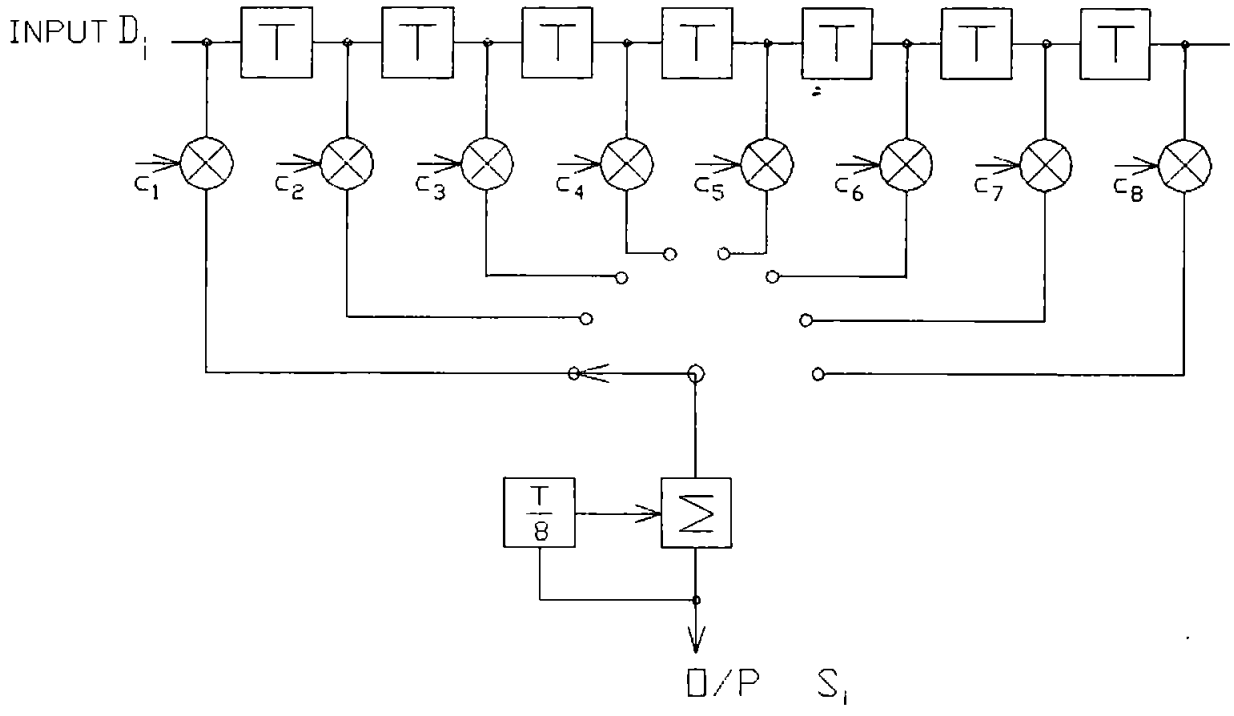


Fig 3.8.3.2.c FIR Filter Implementation with Parallel Multiplication and Serial Addition

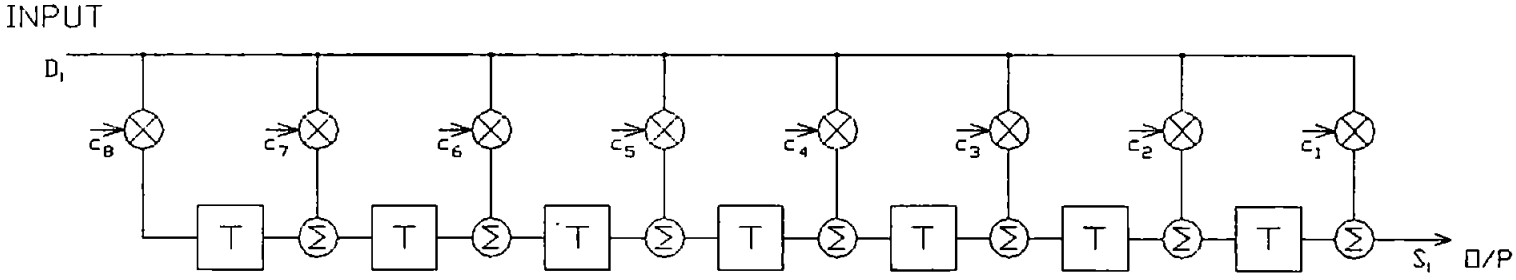


Fig 3.8.3.2.d FIR Filter Implementation with Parallel Multiplication and Cascaded Addition

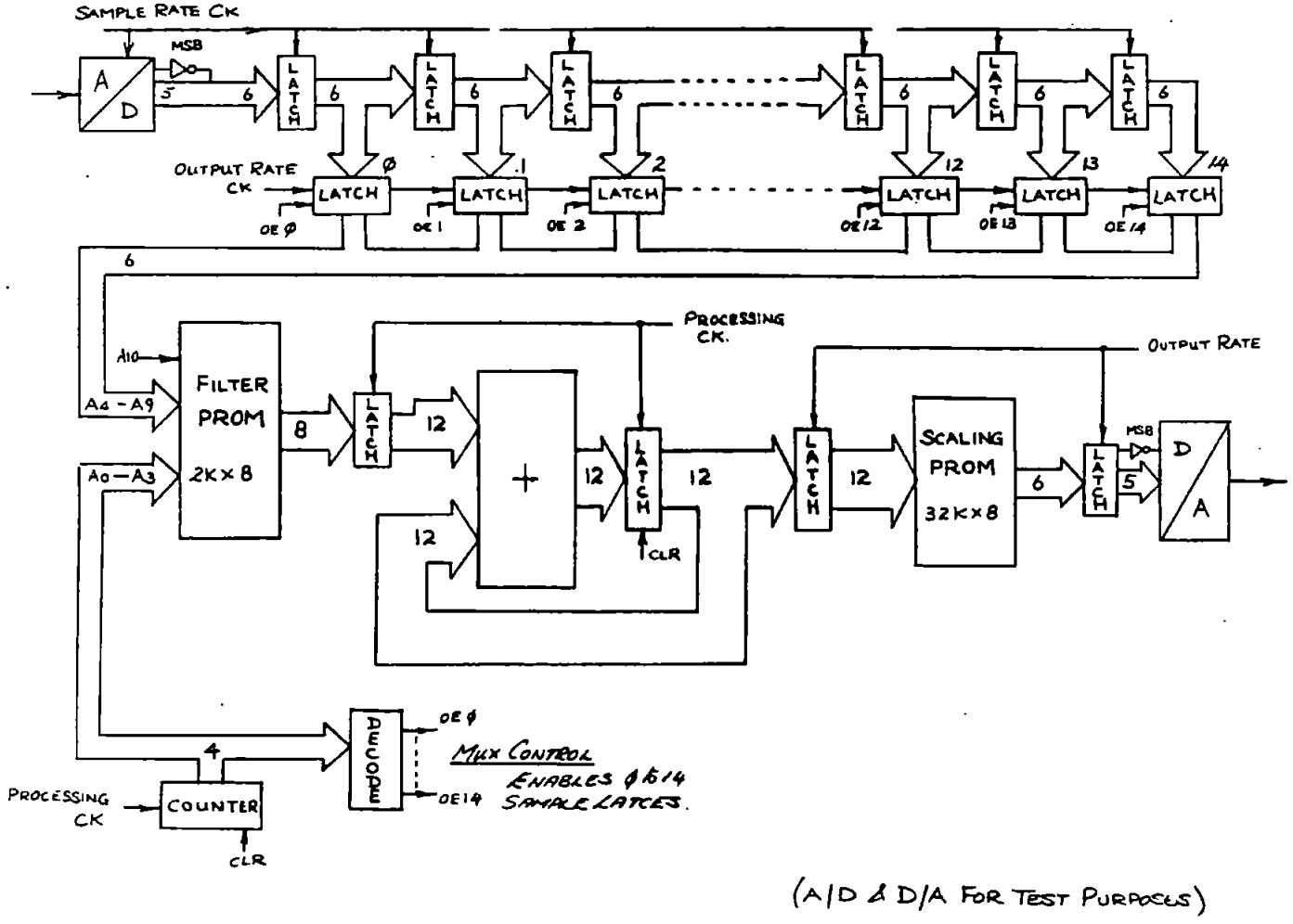


Fig 3.8.3.2.e FIR Filter Implementation used in Demodulator Filters

sample frequency. As the output sample rate is reduced by a factor of 4:1 then the redundant samples need not be calculated. This increases the time available to carry out the convolution process since only one in four of the input samples takes place in the convolution and hence allows high input rates to be practical. The selection of a set of input samples is dependant on the phase relationship between the input and output sampling signals. Any of the sets will create the required frequency response but since we require the central maximum sample, ie the centre of the data eye at the output of the filter, selection of these sample sets is critical. The required set of samples is taken from the delay line and latched onto a set of buffers during one input clock sample. This transfer to the buffers takes place at the output sample clock rate. The correct selection of input sample sets is required to enable the correct final impulse response. Adjustment is provided by changing the phase reference of the output/transfer clock with respect to the input sample clock. Once the required set of samples is present on the latches the convolution takes place. This is carried out by multiplexing the latched samples onto the PROM address lines, along with a number representing the effective coefficient that the sample is multiplied. The PROM is pre-programmed with the answers to the multiplication of the coefficient in form of the algorithm:-

$$R(A(i,j)) = i \times C_j \qquad \text{(Eq3.8.3.2.a)}$$

where R(A(i,j) = result from address A(i,j) given by:-

$$A(i,j) = i' \times 16 + j \qquad \text{(Eq3.8.3.2.b)}$$

j=coefficient number =0..15 i=sign/magnitude equivalent sample value of a 6 bit 2's complement input value, i', with equivalent maximum ,i, equal to +/-1. Hence A(i,j) can be represented by 10 bits. The output from the PROM is a 8 bit representation also in 2's complement format scaled to a maximum of +/- 1. The resulting products are accumulated over the coefficient number count 0 to 14, i.e. 15 coefficients. On count 16 the data is output to a scaling PROM and the accumulator cleared. A new set of input samples is now transferred to the multiplex latches. The accumulator can in these 15 additions in theory produce a maximum 12 bit result. The coefficients are not all at a maximum of 1, as they are a sampled version of the root 30% Raised Cosine filter impulse response, and consequently this maximum is never reached. Also for different coefficients corresponding to different data rates the accumulated maximum result will correspondingly change. To enable this output to be adjusted for maximum signal to quantizing noise the Scaling PROM allows the maximum input value to be adjusted to produce a maximum L bit output value. This is performed by the PROM being pre-programmed with the algorithm:-

$$R(A(j)) = \frac{j' \times N}{\text{Maximum input value}} \qquad \text{(Eq3.8.3.2.c)}$$

where maximum input value is magnitude of maximum value, j'=sign/magnitude equivalent of 2's complement input value j, $N=2^{L-1}$,where L= number of bits in output word, i.e. 6 bit output. This acts as a gain in the system, while also allowing the number of bits representing the output to remain at a manageable size. The error of re-scaling to the L bit output is therefore minimized and is only dependant on the output word size. The maximum error is therefore half a LSB but the signal is maximized to the maximum word length and hence preserving the signal to quantizing noise

ratio. This method also allows for increased flexibility in the usage of the basic filter design. Since the overall convolution technique relies on the coefficients of multiplication i.e. the desired impulse response being multiplied by the incoming samples, the rate at which this convolution takes place results in the scaling of the frequency response obtained from the filter. This design therefore has the flexibility to change the scaling of the frequency response by changing the processing rate of the filter. It is also possible to switch from a 15 coefficient convolution to a 7 coefficient convolution, by changing the multiplexing buffer addressing system (coefficient pointer) and stored coefficient multiplication results in the PROM. By the use of a larger PROM this can be achieved by a simple address selection.For a 15 coefficient filter 4 bit will be required for the coefficient pointer and with a 6 bit input word then a total of 10 bits is required for the multiplication. Hence a minimum memory size of 1Kbytes is required. The addition of another set of coefficients led to the choice of Signetics 82S191BN 2Kbyte Bipolar Proms with access times of 45 nSec enabling the filter to produce a multiplication and addition result in 61nS i.e. a maximum clock frequency of 16.384MHz.

3.8.3.3 Multi-Symbol Rate Receive Filter.

Consider the 16Ksymbol/sec case: the incoming sample stream is highly over sampled but cannot be reduced in sample rate due to aliasing constraints imposed by the SAW filter. One alternative is to change SAW filters. This can be very expensive for a large number of symbol rates. A better alternative was proposed and implemented. Digital bandlimiting filters known as primary filters reduce the bandwidth of the 4MHz sample stream such that sample rate reduction can take place as previously described i.e.. Sub-sampling of the output data producing a decimation in

time. The lowpass primary filters are cascaded in series to reduce the bandwidth of the received signal and the sampling rate until it is at a practical bandwidth and sample rate to allow the final shaping of the frequency response to take place. Each primary filter contains the same impulse response coefficients, but due to different sample rates used, produces scaled frequency responses. The frequency response used for the primary filters was obtained by a computer optimization trading passband flatness and stopband attenuation against impulse response length. The impulse and frequency responses of the primary filters can be seen in fig 3.8.3.3.a/b . The lower the data rate the greater the number of primary filters that must be used to reduce the bandwidth to a manageable level for pulse shaping.

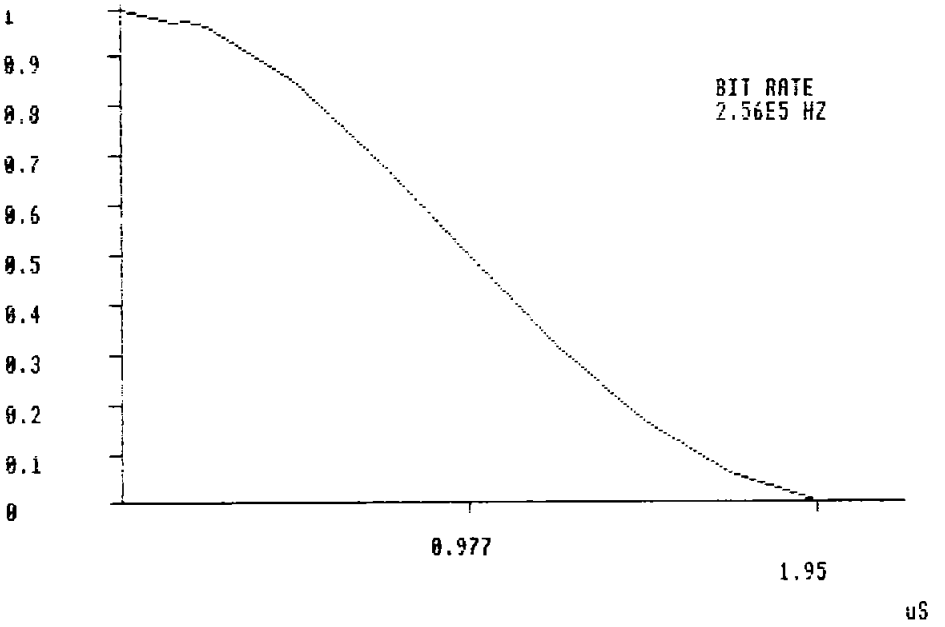


Fig 3.8.3.3.a Impulse Response of Primary Filters

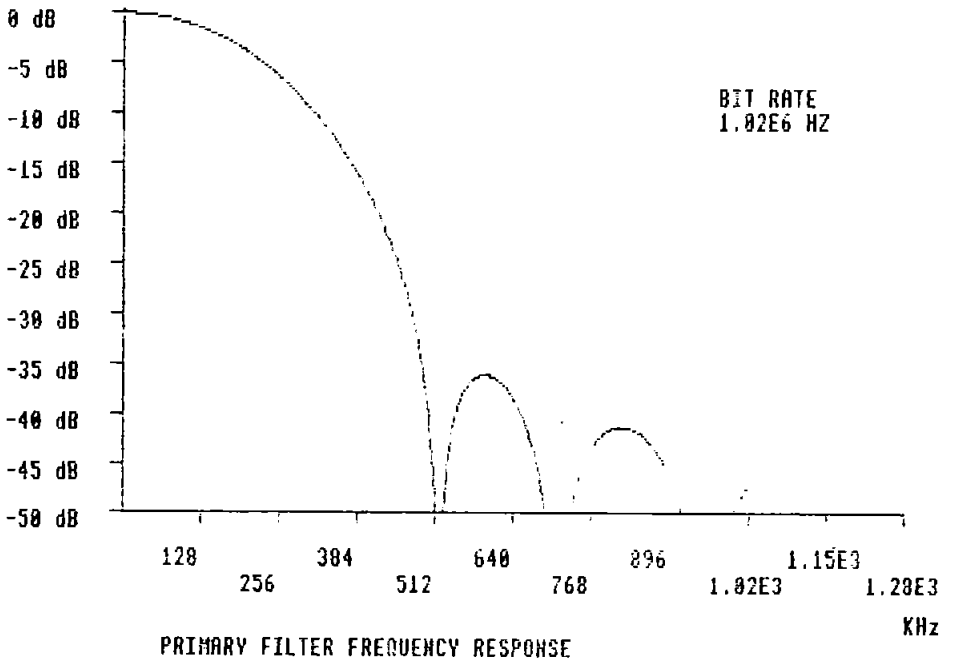


Fig 3.8.3.3.b Frequency Response of Primary Filters

3.8.3.4 Shaping Filter

The final filter is designed to equalize the frequency response of the primary filters and provide the final shaping to obtain the overall root 30% Raised Cosine Filter. The required frequency response of the shaping filter can be seen in fig 3.8.3.4.a. To enable this to be achieved with a target of the frequency sidelobes being greater than 30 dB down, the impulse response of the final filter is arranged such that alternate sample coefficients are set to zero. This enables the long impulse response of the equalizing filter to be convolved while only using 15 coefficients. The introduction of the zero terms results in the frequency response of the filter repeating around one half of the output rate, i.e. twice the symbol rate since the sample rate is effectively reduced by a factor of 2 but with a narrow sample width. The spectrum of the input signal to this filter must therefore be greater than 30 dB down over this frequency range otherwise the noise power at the output will increased. To facilitate these zero samples in the convolution process the input samples are latched into the multiplex buffers with a 2T delay between latched samples. This results in the delay line for the final filter being 31T samples long. The final output rate is four times that of the symbol rate and allows the spectrum to be correctly shaped for conformity to the overall 30% raised cosine response. The selection of Filters required for a multi-data rate Receiver from 16 Ksymbol/sec to 256 Ksymbol/sec can be seen in table 3.8.3.4.a

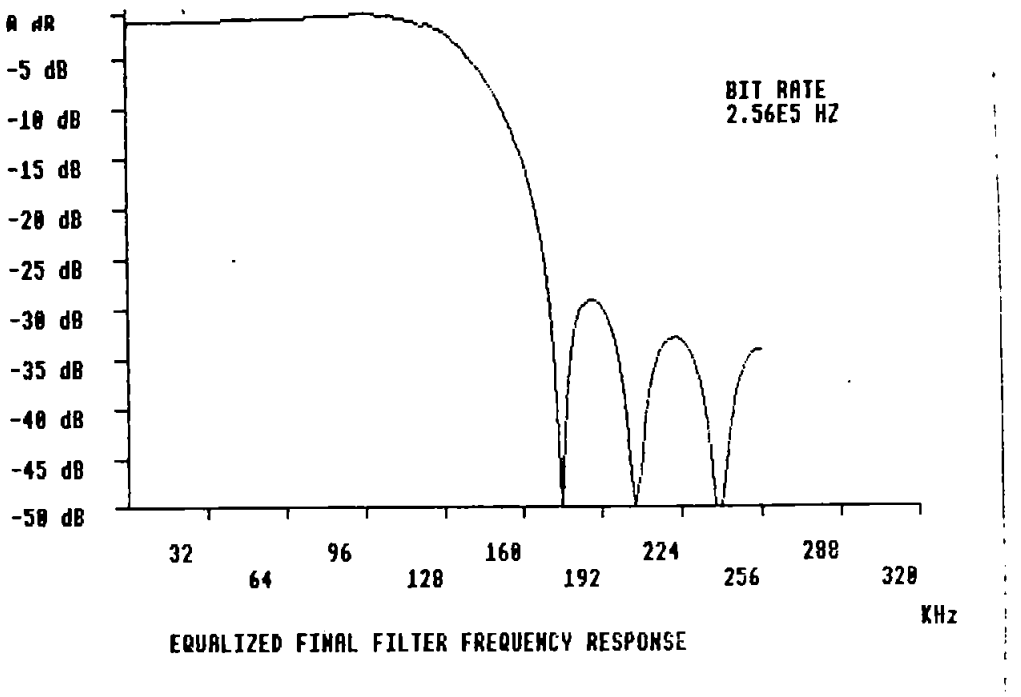


Fig 3.8.3.4.a Frequency Response of Shaping Filter

Symbol	Filter 1 *				Filter 2				Filter 3				Filter 4				Filter 5		
	I/P rate	O/P rate	Co	BW Hz	I/P rate	O/P rate	Co	BW Hz	I/P rate	O/P rate	Co	BW Hz	I/P rate	O/P rate	Co	BW Hz	I/P rate	O/P rate	Co
16K	4M	2M	7	1M	2M	512K	15	256K	512K	128K	15	64K	128K	64K	7	32K	64K	64K	15
32K	4M	1M	15	512K	1M	256K	15	128K	256K	128K	15	32K	BYPASS				128K	128K	15
64K	4M	2M	7	1M	2M	512K	15	256K	512K	256K	15	64K	BYPASS				256K	256K	15
128K	4M	1M	15	512K	1M	512K	15	128K	BYPASS				BYPASS				512K	512K	15
256K	4M	2M	7	1M	2M	1M	15	256K	BYPASS				BYPASS				1M	1M	15

* Requires switchable coefficient and scaling proms

Table 3.8.3.4.a Filters Required for a Multi-Data Rate Receiver from 16 Ksymbol/sec to 256 Ksymbol/sec

3.8.3.5 Frequency Spectra Due to Decimation in Time

Considering the sampled bandpass frequency spectra at the 1.024MHz input, the spectral shape to the components is defined by the SAW filter and will be distributed across the frequency spectrum as in fig 3.8.3.5.a. If the required signal is 256 Ksymbols/sec, three filtering operations are required with decimations in time from 4 MHz to 2 MHz to 1 MHz. Care must be taken to remember that the bandpass spectra repeats around the sampling frequency, and at multiples of the sampling frequency.

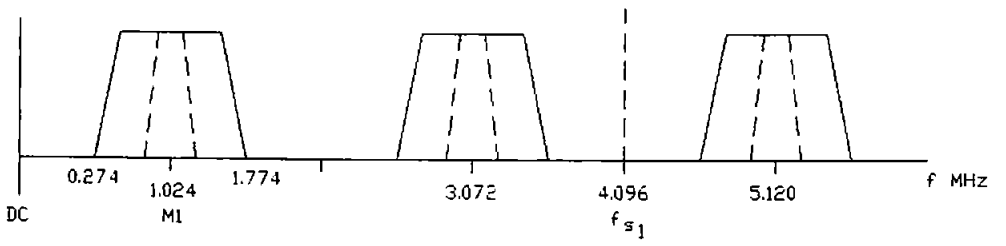


Fig 3.8.3.5.a Sample Bandpass Frequency Spectrum at A/D Input.

down converting this signal to baseband by mixing with a 1.024MHz 'Local Oscillator' results in fig 3.8.3.5.b.

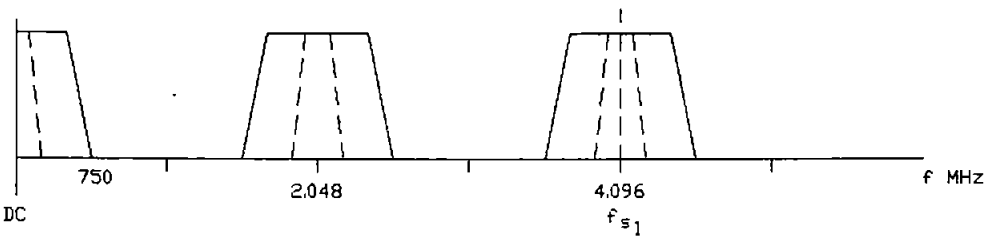


Fig 3.8.3.5.b. Down-Converted Spectra

Filtering this signal by the 1st primary filter with 7 active coefficients would produce the frequency spectra of fig 3.8.3.5.c. However this signal is sub sampled before it is generated at the output of the filter and hence the spectra now repeat around 2.048 MHz as in fig 3.8.3..5.d. The spectra response of the wanted signal is shown dotted within the filter response.

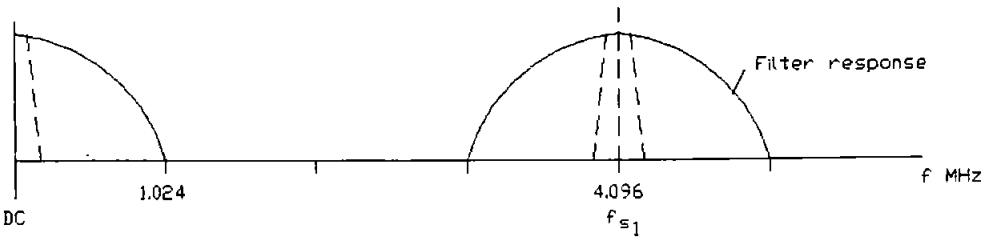


Fig 3.8.3.5.c Output from Primary Filter Before Sub Sampling Occurs.

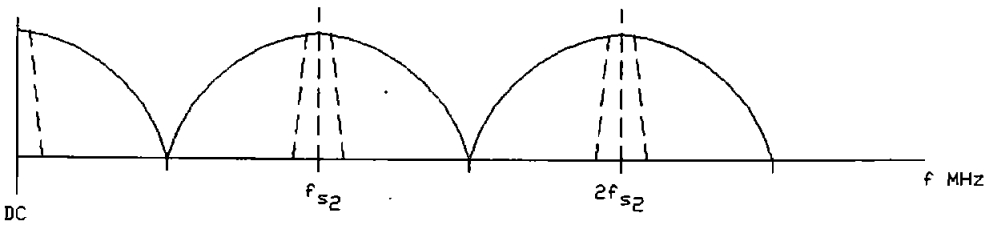


Fig 3.8.3.5.d Output from First Primary Filter After Sub-sampling.

The signal is the filtered again by a 2nd Primary Filter with 15 active coefficients producing the frequency spectra of fig 3.8.3.5.e before sub sampling at 1 MHz occurs.

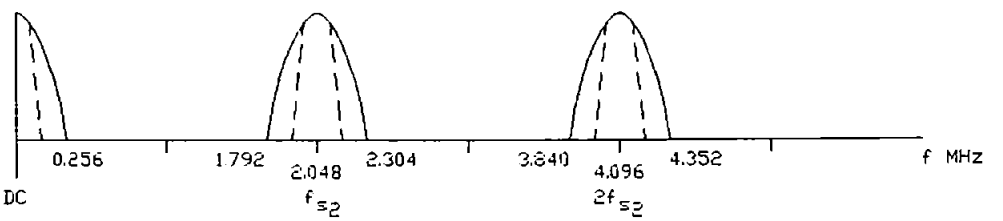


Fig 3.8.3.5.e Output from 2nd Primary Filter Before Sub-sampling at 1MHz Occurs.

The signal is then sub-sampled an again the frequency spectra repeats around the new sampling frequency of 1MHz as in fig 3.8.3.5.f.

Fig 3.8.3.5.f Output from 2nd Primary Filter after Sub-sampling at 1 MHz.

The signal is now of sufficiently low bandwidth and sample frequency that it is possible to obtain a root 30% Raised Cosine frequency response filter within the time constraints of the 15 active coefficients. To obtain good stopband performance, approximately - 30 dB, 31 coefficients are used with alternate coefficients being zero. This results in the filter frequency response repeating around half the sample rate. i.e. 512 KHz. This is because the 15 non zero coefficients can be classed as having been extracted from a continuous time response by a sampling signal of 512KHz but with a hold time of a 1.024 MHz sample. This results in the sinc function produced by the hold having its first null at 1.024 MHz but with the signal repeating around the sample rate of 512 KHZ. As all significant energy at this point has been removed by the

previous filters this does not cause a problem. The final frequency spectra produced by the filters can be seen in fig 3.8.3.5.g.

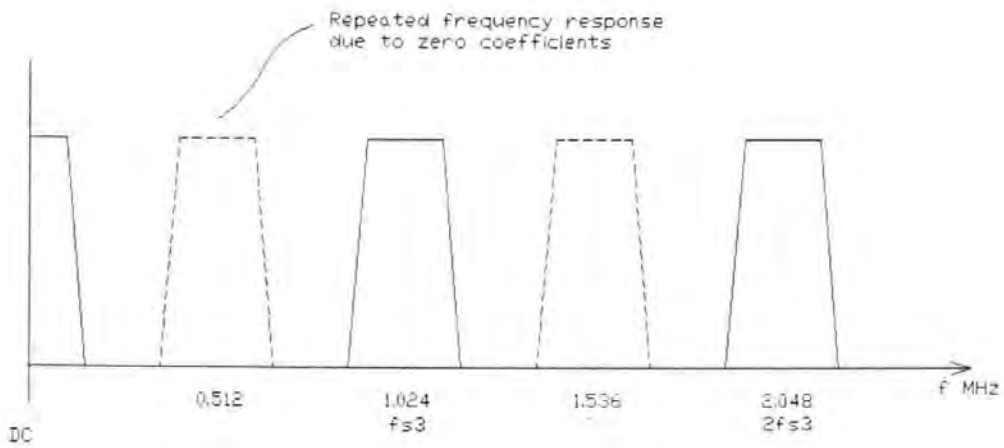


Fig 3.8.3.5.g Final Output from Shaping Filter, the Required Signal Response Being Shown by the Dotted Line.

3.8.4 Symbol Timing Recovery

In satellite data transmissions it is usual to scramble the transmitted data <9>. This is not for security reasons but to randomize the incoming data sequence, breaking up long sequences of '0's and '1's , and hence reducing the production of discrete spectral lines. This randomization aids symbol timing by breaking up long data sequences with little timing content, but the scrambling process itself can introduce limited sequences of 1's and 0's depending on the length and format of the scrambling technique used. Symbol timing recovery is not simply achieved and many methods have been produced <10> which often rely on non-linear methods. The method that was adopted is that of the Delay-Multiplier <11> which was implemented at the 200MHz IF of the SAW filter output,as shown in fig 3.8.4.a. The optimum delay was found to be Td=0 and represents a squaring function enhancing the envelope variations of the filtered waveform. This has been shown by Gardner <22> to maximize the recovered symbol timing component for Nyquist pulses. The effect of the squaring function is to spread the received spectrum over twice its bandwidth. If the low-pass equivalent spectrum of the data signal exists from 0 to 1/2T Hz then the equivalent low-pass resultant spectrum from the squaring function extends to roughly to 1/T Hz. However a signal component is now produced at the symbol rate. The amplitude of this component has an oscillatory behaviour depending on the delay time with maximum amplitude occurring at Td=0 <11>. An expression for the signal to noise ratio of the recovered component for

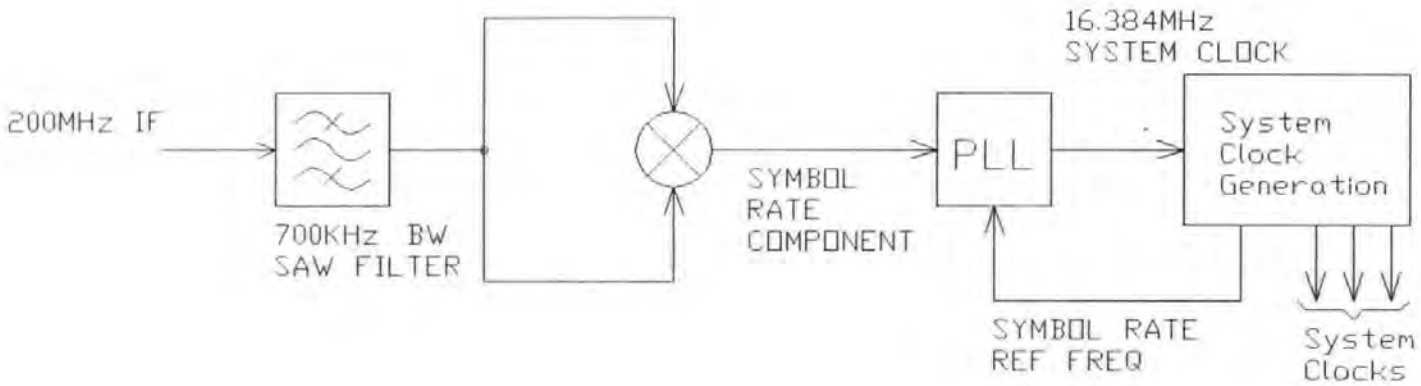


Fig 3.8.4.a Symbol Timing Recovery Circuit using Delay and Multiply Method, Td=0

QPSK is given by Palmer et al <14> for Nyquist pulses as :-

$$S/N = \frac{u.\ r.Eb.Rs}{4.No.Bn} \qquad \text{(Eq3.8.4.a)}$$

where u is the squaring loss given by u= 1/(1+(No/2Eb)), Rs = symbol rate, r is the roll-off factor and Bn = noise bandwidth of the second filter of bandwidth B. This function can be normalized to the signal to noise ratio per Hz and hence for a symbol rate of 256Ksymbols/sec and 30% Raised cosine filters, the spectral density for specified Eb/No can be found ,Table 3.8.4.a.

Received signal Eb/No dB	Recovered Symbol timing component SNR dBHz
10	52.6
8	50.5
6	48.3
4	46.0
2	43.6

Table 3.8.4.a Eb/No Against Recovered Symbol Timing Component

These results do not include the self noise due to ISI. We require for low ISI and low degradation in BER a rms timing jitter phase variation of less than 22 degrees <9> for rectangular pulses. The 30% Raised cosine impulse response shape therefore requires more accurate symbol timing sampling to obtain near zero ISI. This ISI results in a reduction in eye height. This reduction is not caused by the main pulse lobe since its shape is nearly constant over the family of curves but by the the zero crossing point no longer being zero at the sampling instant. Another

consideration in this system is that the 'local oscillator' and the master system clocks are also regenerated from this recovered symbol clock. The variance on the symbol clock must therefore be kept to a minimum as the symbol clock is multiplied in the regeneration of the local oscillator and hence its variance also increases by a similar factor. The theoretical timing error variance in radians squared obtain is given by Palmer et al <14> as,

$$\sigma^2 = 1/(2S/N), \qquad \text{(Eq3.8.4.b)}$$

examples of which are shown in Table 3.8.4.b.

SNR	Recovered Symbol clock Phase	
dB	variance rad^2	RMS Phase deviation degrees
10	0.05	12.8
15	0.016	7.2
20	0.005	4.1
25	0.001	2.3

Table 3.8.4.b Received SNR Compared with Recovered Carrier Phase Variance

These results do not include the jitter caused by self-noise due to ISI if any. Hence for a worst case of a 16 Ksymbol/sec data rate the ratio of symbol rate to carrier of 1.024MHz is 1:64. hence if the carrier requires a phase deviation of say 5 degrees then the symbol clock deviation is required to be 0.078 degrees and would require a symbol clock SNR of 54 dB indicating a noise bandwidth in the symbol timing recovery system in the order of 1Hz for a 10 dB Eb/No signal. Hence the recovered symbol timing rms

phase deviation must be kept to a minimum by the correct choice of output filtering. The recovered symbol rate frequency component is then locked to a reference symbol rate frequency, obtained by dividing down the master system clock, by a Phase Locked Loop. The voltage controlled oscillator (vco) of the loop is centred at 16.384 MHz and generates the master system clock. Division of this clock produces all clocking components for the system. As the symbol rate reference is obtained from this master clock, and the Phase lock loop always locks with the same phase reference, all the system clocks can be maintained with the correct reference to each other. The symbol rate reference is capable of phase correction by 16 MHZ clock samples (61nS) to correctly select the initial A/D sampling reference with respect to the symbol centre. The symbol rate reference can also be adjusted to correctly sub-sample the final shaping filter outputs.

3.9 Algorithm Development.

Throughout the design of the system, the algorithms used in the research and development of the digital modem have been implemented in Hardware and Software to produce a totally integrated system and meet the required specifications. This modelling and algorithm investigation was carried out using a number of personal computers. Initial investigations where carried out using a BBC Micro Computer and later on an IBM PC-AT. This required the production of a number of algorithms to model the various functions in software and enable the results to be extracted and used in the hardware implementation. The operation of each model will be discussed later but a general appreciation of the operation of the models will be considered.

As an example we will consider the algorithm to calculate the data that is to produce the transmitted waveform. This algorithm must perform a number of functions:-

i) Generate all possible 6 symbol data patterns and calculate intermediate sample values from the convolution of the symbol stream with the transmitter filter impulse response.

ii) Calculate the Transmit filter Impulse response from the Frequency response or vice versa.

iii) Modulate the Intermediate samples onto the 1st IF carrier.

These functions could be implemented in one large computer program but due to memory restrictions this was segmented into a number of smaller programs that could be progressively used to create the final set of data. This required that intermediate data from the various programs was saved to disk before being used in the next program. This produces a simple format for the saved data and hence enabled data to be passed from algorithm to algorithm without any problems. This system of modelling could be further enhanced to produce a complete system simulation allowing functions to added or removed as desired.

3.9.1. Transmitter Algorithms.

As previously discussed, the transmitter must convert a stream of binary NRZ pulses and spectrally shape these in such a way that the overall response of the satellite channel conforms to Nyquist filter criteria. This is achieved by baseband filtering with Root 30% raised cosine filters for each channel data stream and then modulating this filtered symbol stream in quadrature onto the 1st IF

carrier as previously discussed in section 3.7.3. This is all performed by a look-up table implemented within a PROM. To enable this to take place as required the PROM must be programmed with the correct information. This required the development of a number of algorithms.

3.9.1.1. Raised Cosine Frequency Response and Impulse Response Generation.

This is obtained by the calculation of sample values from either the closed form time expression or transformation of the Frequency expression for the Raised Cosine Filter <13>. The Time domain response of the Raised Cosine function is given by :-

$$x(n\tau) = \frac{\text{Sin } \pi n\tau/T}{\pi n\tau/T} \cdot \frac{\text{Cos } \pi n\tau/T\beta}{(1-(2\beta n\tau/T)^2)} \qquad \text{(Eq3.9.1.1.a)}$$

where $\tau = 1/f_s$, $T = k\tau$, $n = 0$ to Nk and N is the number of symbol periods over which the single sided impulse response exists.

The Frequency domain response given by:-

$$X(nf') = \begin{cases} 1 & 0 < nf' < (1-\beta)/2T \\ 1/2(\ 1-\sin(\pi(nf'-1/2T)/\beta)) & (1-\beta)/2T < nf' < (1+\beta)/2T \\ 0 & nf' > (1+\beta)/2T \end{cases}$$

(Eq3.9.1.1.b)

Where $f' = 1/N\tau$, $n = 0$ to $N-1$ and N is the transform size. The Root raised cosine is obtained by taking the square root of the raised cosine frequency response values. The impulse response can then be obtained by taking the Inverse Discrete Fourier Transform IDFT of the result.

3.9.1.2. A Discrete Fourier Transform and the Inverse Discrete Fourier Transform

The DFT and IDFT algorithms were obtain from <5> and modified to save processing time. This saving in processing time was only viable since the time response was always symmetrical, hence the imaginary part of the complex result is always zero, and that the impulse response was generally quite short with respect to the total number of transform samples. Since the remaining samples are all zero then these contribute nothing by completing the remaining processing. This reduced the processing time by reducing the generality of the transforms and enabled a result to be obtained after say M*N multiplications and additions,where M << N and M is the required number of output samples and N is the N point DFT, instead of N*N for the DFT or $N\log_2 N$ for the FFT.
Hence where

$$M < \log_2 N \qquad \text{(Eq3.9.1.2.a)}$$

then this form may be processed quicker than the FFT. This was not necessarily always the case but the simplicity of the program also increases the confidence in the final result. The DFT algorithm therefore became:-

$$F(iW) = \sum_{k=0}^{N-1} f(kT).\text{Cos } 2\pi ikT/NT \qquad \text{(Eq3.9.1.2.b)}$$

for i = 0 to M-1

and the IDFT becomes

$$f(iT) = \frac{1}{N} \sum_{k=0}^{N-1} F(kW).\text{Cos } 2\pi ikT/NT \qquad \text{(Eq3.9.1.2.c)}$$

for i= 0 to M-1

These programs were developed and integrated with the other software packages to enable any symmetrical sequence of real samples to be converted from the time domain to the frequency domain and vice versa.

3.9.1.3. Convolution of Data Impulse Response.

This is required to take the data impulse stream of 6 data bits, convert to an input data address, and calculate the convolved output for all intermediate samples and their appropriate PROM addresses.

$$I_n = S_D * h(T)$$

for n = 0 to N-1, where N= sample rate/symbol rate and where h(T) is the impulse response of the transmit filter sampled at the sampling rate i.e. 4.096MHz.
S_D is the sequence of impulses at period τ_s = symbol period given by the data word D

$$D = d_5 2^5 + d_4 2^4 + d_3 2^3 + d_2 2^2 + d_1 2^1 + d_0 2^0$$

and d_0 = lsb

(Eq3.9.1.3.a)

such that

$$S_D = d'_5 Z^5 + d'_4 Z^4 + d'_3 Z^3 + d'_2 Z^2 + d'_1 Z^1 + d'_0 Z^0$$

(Eq3.9.1.3.b)

and

$$d'_i = \begin{cases} 1 & d_i=1 \\ -1 & d_i=0 \end{cases} \qquad i = 0 \text{ to } 5$$

(Eq3.9.1.3.c)

with the address location given by:-

$$A(D,n) = D \times N + n \qquad \text{(Eq3.9.1.3.d)}$$

The resulting data can be modulated onto a carrier frequency 1/4 of the output sample rate by the inversion of alternate pairs of data samples. A pair consist of one P channel sample and one Q channel sample and forms one half cycle of the quadrature carrier output waveform. This enables the P and Q channels to use the same PROM but the data each channel accesses is offset by one sample period from the other,i.e. the P channel accesses even addresses while the Q channel accesses only the odd addresses.

3.9.1.4. Plotting of Eye Diagram, Impulse Response, and Frequency Response.

Software routines were produced to depict graphical results for data outputs. This reduced the probability of an undetected major error in the calculations since all the sampled data represents analogue processes which through experience can be evaluated more easily than tables of numbers.

3.9.2. Receiver Algorithms.

The receiver algorithms and software can capitalise on the programs already developed for the transmitter in that the Raised Cosine, DFT/IDFT, and Plot programs can all be utilised in the receiver designs.

3.9.2.1. The Primary Decimation Filters.

The primary filter perform two function:-

a) Bandwidth reduction
b) Output sample rate reduction.

The bandwidth reduction is obtained by choosing a series of coefficients that produce a sharp roll-off while preserving the signal passband, linear phase and short impulse response. This can be obtained by using a successive approximation technique whereby an initial frequency response is converted back and forth between its time and frequency responses reducing undesired effects until all criteria are met. This technique resulted in an impulse response as previously seen in fig 3.8.3.3.a and frequency response in fig 3.8.3.3.b. This impulse response can be implemented using 15 coefficients but it should be noted that the frequency response has a small degradation at the 3dB point of the signal. This can be equalized by tailoring the combine response of the Primary and Shaping filters to produce the root 30% raised cosine response required. The frequency response now forms a null at 1/8th of the input sample rate the output can be sub-sampled down to a minimum rate of 1/4 the input sample rate.

3.9.2.2. The Shaping and Equalization Filters.

From the frequency response of the primary filters, X(W), and the required frequency response of Root 30% Raised Cosine Filters, Y(W), it is possible to calculate the required frequency response of the shaping filter, H(W) by:-

$$H(W) = \frac{Y(W)}{X(W)} \qquad \text{(Eq3.9.2.2.a)}$$

By the use of the IDFT, the time response and hence the coefficients can be found. Software routines were developed for all these manipulations. This results in the frequency response and time response of the shaping filter being as in fig 3.9.2.2.a and fig 3.9.2.2.b respectively.

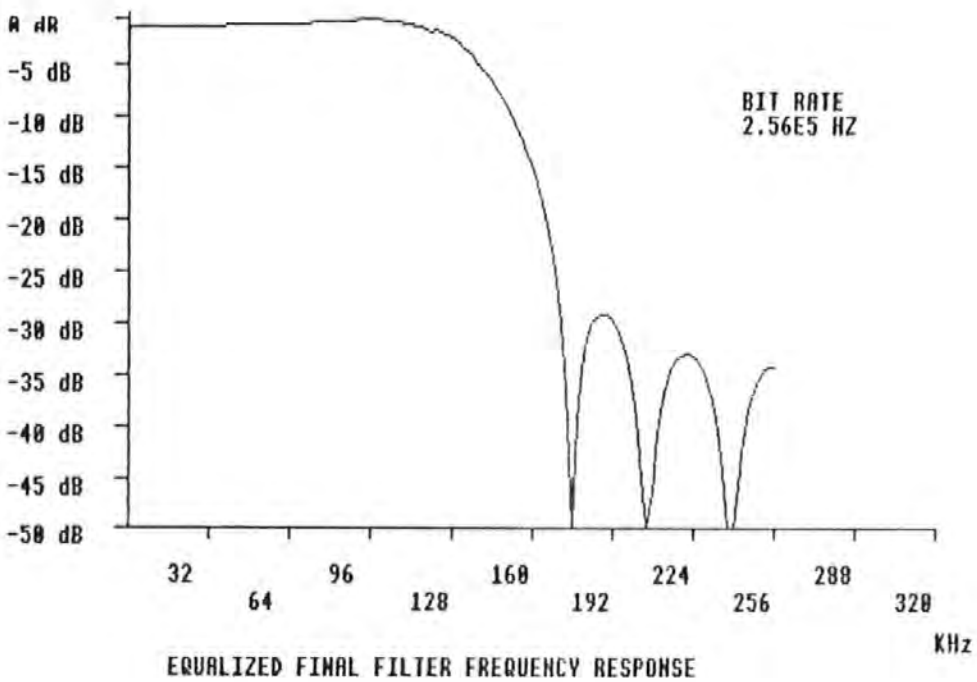


Fig 3.9.2.2.a Frequency Response of Final Shaping Filter with Equalisation.

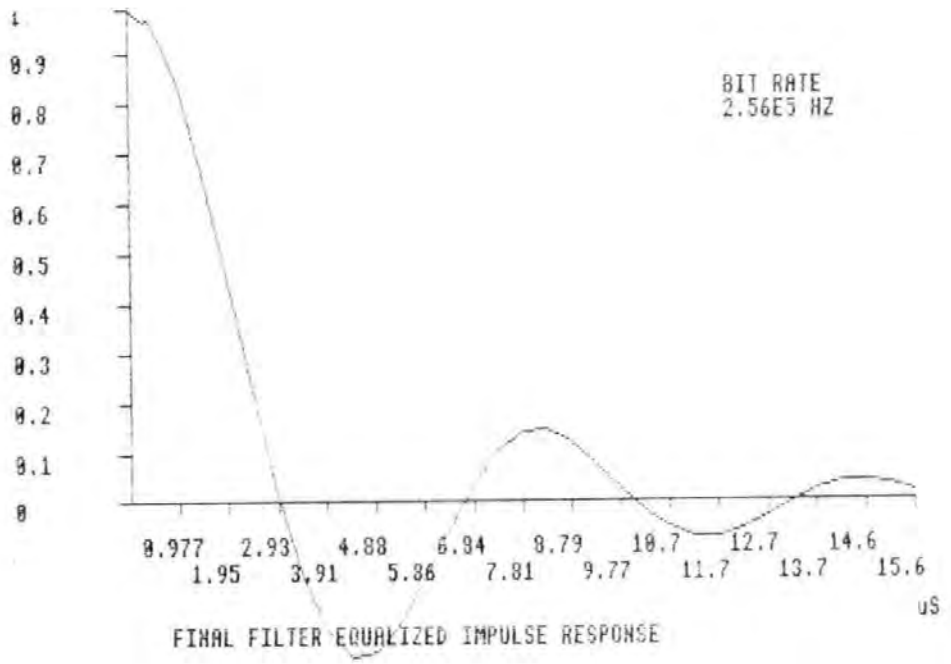


Fig 3.9.2.2.b Time Response of Final Shaping Filter with Equalisation.

3.9.2.2.1 Coefficient Multiplication in Digital Filter

The coefficients obtained for the received filters are not used directly but as a function in calculating a multiplication table to be stored in a PROM . This enables a fast calculation time only dependent on the access time of the PROM (less than 60ns). Each coefficient can be multiplied by all possible input sample words and hence for a 6 bit input word and 15 coefficients, a 1kbyte Prom is required, 2^6 x 16 = 1024.

This Look-up table can be produced by the function:-

$$M(i) = D' \times h(i) \qquad \text{(Eq3.9.2.2.1.a)}$$

where D' = sign magnitude representation of the 2's complement data word D = 0 ...63 for a 6 bit word.
h(i) = i th Coefficient value i=0 ...J-1 , and J= total number of coefficients.The address location for the multiplication result M(i) is given by:-

$$A_M(D,i) = D \times J + i \qquad \text{(Eq3.9.2.2.1.b)}$$

where $J = 2^k$ and k=4. A 2Kbyte Signetics 82S191 PROM was used and allowed for two filter responses to be resident within the same PROM.

3.9.2.3. Data Decode Prom

The Data decode PROM must take samples from the P and Q channels at the centre of each symbol and perform the conversion from rectangular coordinates to Polar coordinates. This can be easily achieved by the pre-programming of two PROMS, to convert to phase and amplitude respectively.

$$\Phi_a(PQ) = \Gamma^{-1}(Qn/Pn) \quad + x$$

(Eq3.9.2.3.a)

where $\Phi_a(PQ)$ is the output phase value Φ for address a(PQ), P,Q are the sample values for the centre of the symbol period of the P and Q channels nth sample respectively. The use of channel sign/ magnitude arithmetic allows the Γ-1(.) function as defined in Eq2.2.1 to be decoded for the full 2π radians. The function x =0, $\pi/4$ for BPSK and QPSK respectively rotates the space diagram at the output of the Γ-1(.) function such that the phase angles at the centre of the symbol are decoded onto the $0,\pi/2,\pi,3\pi/2,2\pi$ axis for QPSK, and $0,\pi$ axis for BPSK if the transmitter is locked to the receiver. The address conforms to the algorithm, for 6 bit sample values:-

$$a(PQ) = N.Pn + Qn \qquad P,Q=0..N-1$$

(Eq3.9.2.3.b)

$N= 2^k$, and k=6.

The algorithm to produce the conversionto amplitude values is given by:-

$$A_a(P,Q) = \sqrt{(P_n^{\,2} + Q_n^{\,2})}$$

(Eq3.9.2.3.c)

with the address location for the result given by Eq3.9.2.3.b.

4. Introduction to Carrier Phase Estimation

The classical solution for the demodulation of PSK information involves the setting up of a local reference signal to synchronously demodulate the received signal from its intermediate frequency. For a BPSK or QPSK signal this can be simply achieved by the use of a quadrature down conversion process. The reference signal is mixed in quadrature with the received signal such that the products produce no cross channel interference between the quadrature channels. Hence a phase reference must be created that maintains a $n.\pi/2$ radian phase offset (where n=0,1,2,3,4) to the received signal. This must be maintained to an accuracy, in the presence of noise, of less than 0.5 radians deviation <23>, so that a major degradation in the system performance does not occur. Examples of phase error and Eb/No degradation can be seen in Table 4. This is classically achieved by the use of a PLL technique such as the Costas loop <24>,or one of the many other schemes that have evolved <10>. The Costas loop is particularly well suited to a quadrature down conversion system since most of the components required for its construction already exist for the demodulation of the data. If this technique is used in the digital demodulator already described the digital filter outputs would be used to provide phase information to a Voltage controlled local oscillator to maintain the reference phase relationship with the received signal. This produces a major problem. Due to the nature of the digital filters a long time delay is introduced into the loop. The effect of this time delay is as described by Develet <25>, and produces false lock conditions with a greatly reduced frequency acquisition range, due to a reduced loop bandwidth to stabilize the loop . The use of a sweeping additive acquisition signal can alleviate the reduction in acquisition range but false locks can still occur and would have to be detected and

corrected. This system would also have the problem that if the signal became unlocked then the acquisition sequence would have to be reinitialized, hence losing data for this period. An alternative to the Costas loop method such as the xN mod 2π method <10> also suffers from similar acquisition problems due to the narrow bandwidths of filters (or PLL) required to achieve the degree of accuracy required in the reference signal phase variance.

In general, an ideal carrier recovery system should be able to achieve three major goals:-

1) A wide frequency acquisition range.
2) Small reference phase variance.
3) A continuous acquisition mode.

A fourth point that is desirable in this digital system is that the incoming carrier frequency, I.F. must be maintained to be synchronous to the digital system clocks and the symbol rate clocks to allow for the simplified method of down conversion already described. Synchronism with the system clocks could be achieved by one of the above methods but unless these are also synchronous with the symbol rate the symbol information will slip past the system clocks and some method of estimating the centre of each symbol would be required. This could be achieved by an interpolation filter and represents an increase in the number of samples/symbol which have previously been reduced and hence is not the desired solution.

Several systems were investigated both experimentally and by the computer simulation in order to find a system that overcame all the above problems. The preferred system only required synchronization with the symbol rate frequency, while the carrier IF could be close to but not locked to system clocks. This resulted in a novel carrier recovery system that is not phased locked. Instead a Phase Estimator

operates on a changing phase,tracking the phase changes as they occur. Phase variations are produced by a frequency offset from nominal, and phase noise contributions from all the frequency sources in the link. The process can also provide digital information of frequency offset errors so that these may be corrected for example by a microprocessor to synchronously change frequency, without causing any major phase jumps, to the corrected frequency. A similar phase processor has been proposed by Viterbi and Viterbi <26> and simulated by Mcverry <27>. However, this required the use of complex arithmetic and was prone to cycle slipping due to noise at low Eb/No. The major cause of cycle slipping is the small number of symbols averaged over to achieve fast acquisition times and mistakes in tracking the transitions from one quadrant to the next. In the reference above the use of differential encoding and decoding reduces the effects of cycle slipping so that it is not a limiting factor. The need for maximum satellite power efficiency has now placed emphasis on absolute phase encoding so the cycle slip performance is important. To reduce cycle slips the number of averaging samples must be increased and an algorithm is needed to correctly detect the quadrant boundaries. One such algorithm is used in the method described by Tomlinson and Bramwell <21> and also does not require the use of Complex arithmetic as the Complex representation is converted to a Polar representation with only the phase angle being processed. This results in a simplification of the hardware required and by the use of modulo arithmetic and Infinite Impulse Response Filters the acquisition time can be easily traded against cycle slipping performance at given values of Eb/No.

Eb/No dB	Regenerated Carrier SNR dB	Degradation from Theoretical dB
7	5	3.6
	10	1.5
	20	0.18
	30	0.02
6	5	3.0
	10	1.19
	20	0.13
	30	0.01
5	5	2.48
	10	0.95
	20	0.11
	30	0.01

Table 4.a Degradation from Theoretical Due to Noisy Phase Reference

4.1. Phase Incoherent PSK Detection

In previous discussions on the operation of the digital receiver it has been assumed that the incoming signal has been forced to be synchronous to the system clock frequencies allowing the use of the digital frequency translation technique described in section 3. This results in a precise phase relationship being maintained between the received signal's I.F. centre frequency and the system clock i.e. Local oscillator. The case is now considered where this is not true, where the signal is not phase locked but where the I.F. frequency is close to the reference signal and assuming no noise component is present for simplicity. It can be shown that the signal, at the symbol centre , after filtering of high order frequency components can be represented by:-

P. channel output

$$R_p(t) = P\,\mathrm{Cos}(\Phi(t)-\phi) - Q\,\sin(\Phi(t)-\phi) \qquad \text{(Eq4.1.a)}$$

Q. Channel output

$$R_q(t) = P\,\sin(\Phi(t)-\phi) + Q\,\mathrm{Cos}(\Phi(t)-\phi) \qquad \text{(Eq4.1.b)}$$

where $\Phi(t)$ is the time varying phase produced by the frequency offset ,ϕ is initial reference phase offset,and P and Q represent the data signal for a QPSK system. In the synchronous case, assuming $\phi = 0$ and $\Phi(t) = 0$ then:-

P. Channel, $R_p(t) = P$ (Eq4.1.c)

Q. Channel, $R_q(t) = Q$ (Eq4.1.d)

as expected. The significance of the equations 4.1.a and 4.1.b can be best obtained by an examination of the Argand

diagram that they produce. This can be seen in fig 4.1.a. As the value of $\Phi(t)$ changes with time the carrier phasor rotates with reference to the receiver phasor axis set up by the receiver local oscillator. This results in the phase angle of the transmitted symbol set rotating with the phasor $\Phi(t)$ mod 2π. It is necessary to be able to calculate this rotation of $\Phi(t)$ on a symbol to symbol basis and correct for this rotation when decoding the data signal to convert from a phase incoherent system to that of phase coherent. This rotation can be made easier to manipulate by converting the rectangular format of eq4.1.a and eq4.1.b to that of a Polar format and hence become:-

$$A^2 = R_p(t)^2 + R_q(t)^2 \qquad \text{(Eq4.1.e)}$$

$$\Phi'(t) = \Gamma^{-1}\left(R_q(t)/R_p(t)\right) \qquad \text{(Eq4.1.f)}$$

where $\Phi'(t) = \Phi(t)-\phi$. If it is assumed that the P and Q data are both constant at +1 then the function $\Phi'(t)$ represents a linear ramp over the range 0 to ∞ mod 2π where the rate of change of phase with time $d\Phi'(t)/dt$, the slope, is proportional to the frequency difference between the I.F. and the local oscillator fig 4.1.b. The addition of data to the signal results in values $\Phi'(t)$ constructed of the basic ramp caused by the frequency error, $\Phi(t)$, and the data phase change $\phi_d(nT)$. Hence to obtain any estimate of $\Phi(t)$ the data must first be removed.

$$\Phi'(t)= \Phi(t)+\phi_d(nT)-\phi \qquad \text{(Eq4.1.g)}$$

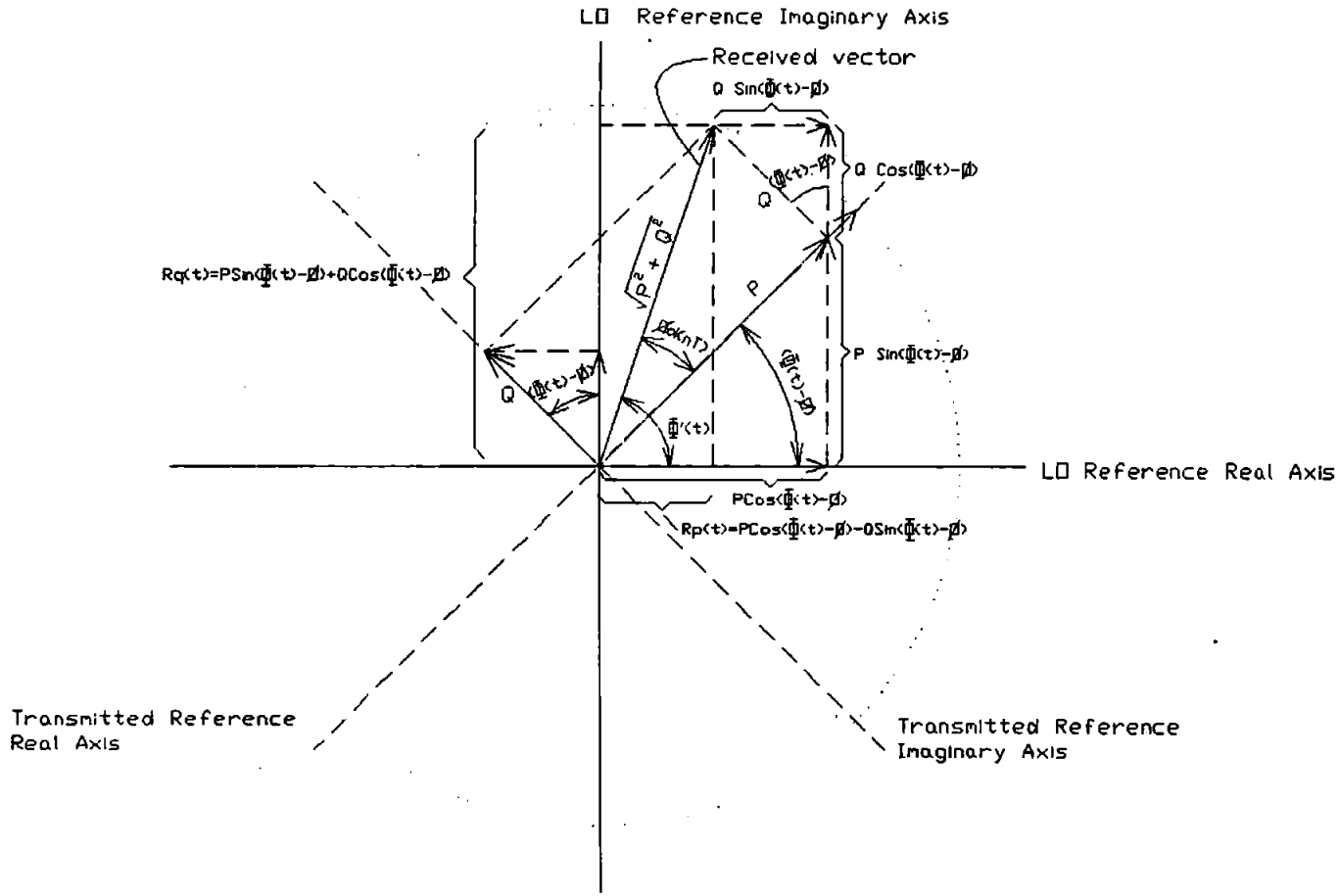


Fig 4.1.a Argand Diagram of Received Vectors Against Local Oscillator Reference.

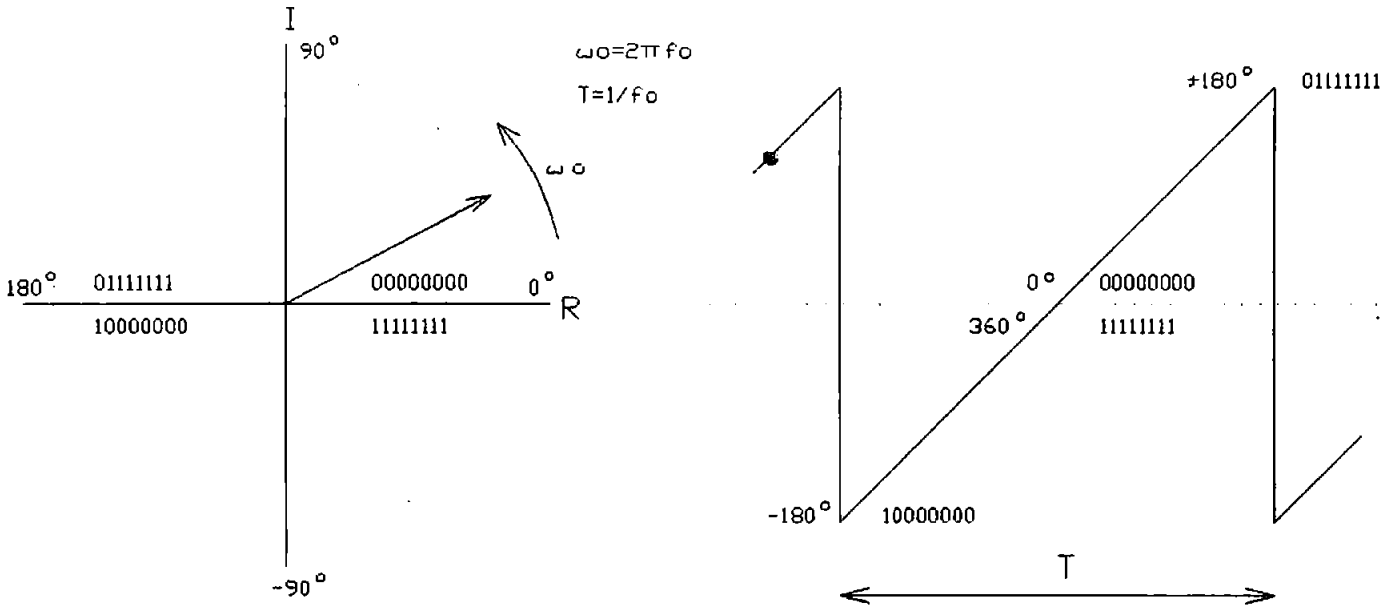


Fig 4.1.b Linear Representation of Frequency Offset as Changing Phase with Time.

4.2. Data Removal

Assume that available at the output of the $\Gamma^{-1}(.)$ function there is an estimate available of $\Phi(t)$,$\hat{\Phi}(t)$. If this is subtracted from $\Phi'(t)$ then letting $\phi = -\pi/4$ the result is :-

$$p(t) = (\Phi'(t) - \hat{\Phi}(t)) \bmod 2\pi \qquad \text{(Eq4.2.a)}$$

$$= (\Phi(t) + \phi_d(nT) + \pi/4 - \hat{\Phi}(t)) \bmod 2\pi \qquad \text{(Eq4.2.b)}$$

$$= (e(t) + \phi_d(t) + \pi/4) \bmod 2\pi \qquad \text{(Eq4.2.c)}$$

where e(t) represents the error between the actual value of $\Phi(t)$ and the estimated value $\hat{\Phi}(t)$. If QPSK data is considered where the data is defined by phase angles $\phi_d(nT)$, at the centre of the symbol ,of $m.\pi/2$, for $m=0..4$, then data can be removed by the use of a x4 mod 2π function. Division by 4 now results in p'(t) being restricted to a +/- $\pi/4$ range with the usual phase ambiguity.(Similarly in BPSK but with a x 2 mod 2π function.)

$$p'(t) = (p(t) \times n \bmod 2\pi)/n$$
$$= (((e(t) + m.\pi/2 + \pi/4) \times n) \bmod 2\pi)/n$$
$$= ((n.e(t) + n.m.\pi/2 + n.\pi/4) \bmod 2\pi)/n$$
$$= (n.e(t) \bmod 2\pi + n.m.\pi/2 \bmod 2\pi + n.\pi/4 \bmod 2\pi)/n$$

where n= 4

for $m=0..4, n=4$

$n.m.\pi/2 \bmod 2\pi = 0$,

and $n\pi/4 = \pi$

hence

$$p'(t) = e(t) + \pi/4 \bmod \pi/2 \qquad \text{(Eq4.2.d)}$$

It has therefore been possible to produce an error signal with which to control the estimate of $\hat{\Phi}(t)$. Filtering by a function h(t) and then integrating results in an ideally linearized phase lock loop as shown by Jaffe and Rechtin <28> with the acquisition time dependent on the time constant of the integrator and the loop filter h(t). However if the phase estimate deviates from the received phase by $\pm\pi/4$ then an ambiguity arises and the loop will not function correctly and fall out of lock. Large frequency errors can not be corrected by the phase processing loop. However, a supplementary frequency tracking loop can be used to correct major frequency errors. The system is therefore in lock or out of lock, lock being indicated by the phase estimate following the received phasor, i.e. with a error signal tending to zero, with only the usual ambiguity of $n.\pi/4$ for QPSK n=0..4, or similarly $\pi/2$ for BPSK.

4.3. Phase Processor

Consider a system as in fig 4.3.a, where the data has been removed from the rotating phasor $\Phi(t)$ and obtain a value p'(t) representing the error between the rotating phase $\Phi(t)$ and the estimated value $\hat{\Phi}(t)$. Addition of p'(t) to the phase estimate now results in a value for the rotating phase $\Phi(t)$ since:-

$$p'(t) = (e(t) + \pi/4) \bmod \pi/2 \qquad \text{(Eq4.3.a)}$$

if $|e(t)| < \pi/4$ then

$$p'(t) = e(t) + \pi/4 \bmod \pi/2$$

$$p'(t) = (\Phi(t) - \hat{\Phi}(t) + \pi/4) \bmod \pi/2$$

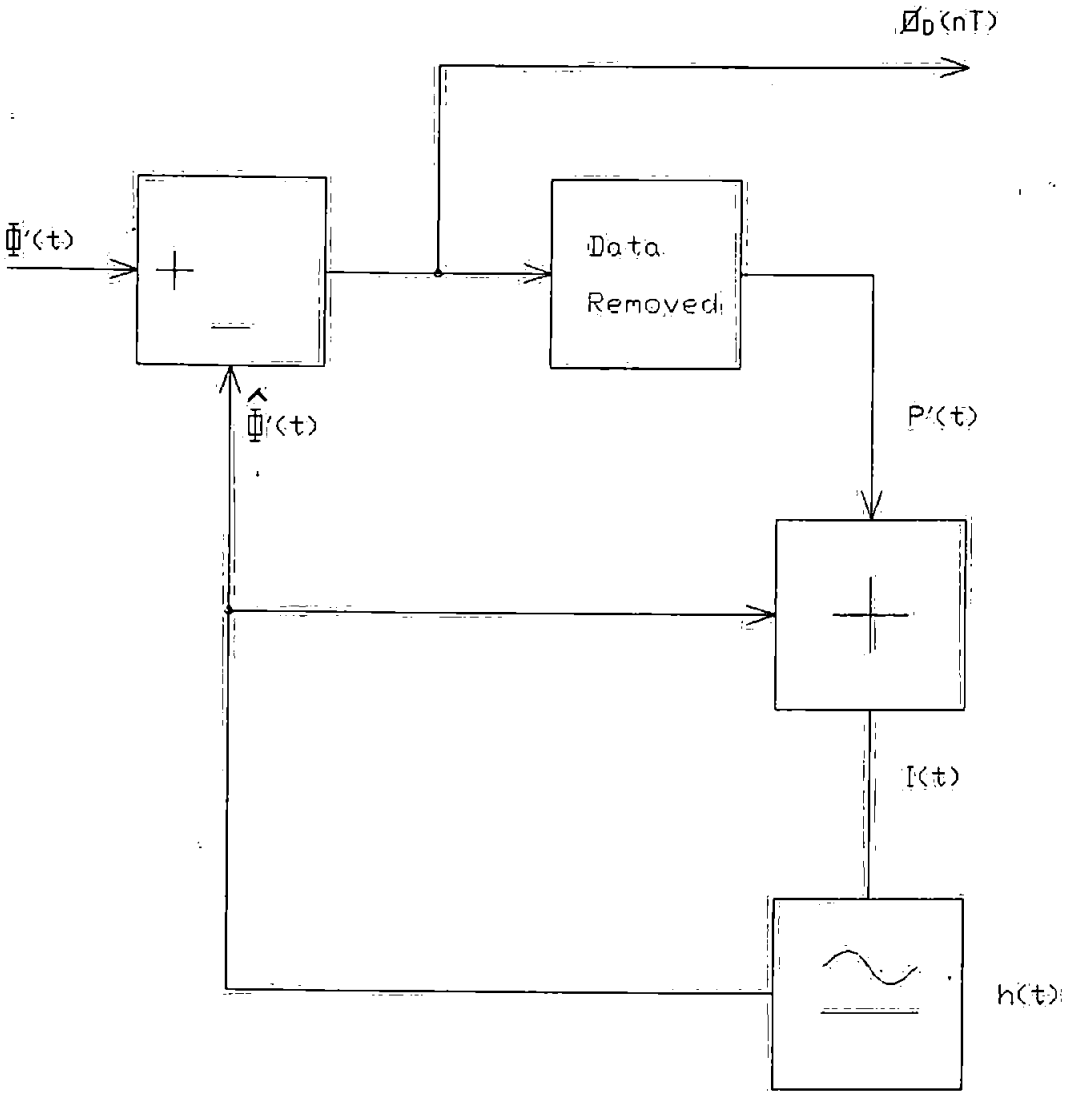


Fig 4.3.a Basic Phase Processor Block Diagram

hence

$$\hat{\Phi}(t)+p'(t) = (\hat{\Phi}(t) + \Phi(t)-\hat{\Phi}(t)+ \pi/4) \bmod \pi/2$$
$$= (\Phi(t) + \pi/4) \bmod \pi/2 \qquad \text{(Eq4.3.b)}$$

The phase estimating loop is now only dependent on the filter time constant and not the integrator as before. If this is considered with a noise component Eq4.2.c then becomes :-

$$p(t) = (e(t) + \phi_d(nT) + \phi_n + \pi/4) \bmod 2\pi \qquad \text{(Eq4.3.c)}$$

where ϕ_n = phase noise, with zero mean, and bi-variate gaussian distribution, and variance σ^2. and Eq4.3.b becomes:-

$$\hat{\Phi}_n(t) = \hat{\Phi}(t)+p'(t)$$
$$= (\Phi(t) + \phi_n + \pi/4) \bmod \pi/2 \qquad \text{(Eq4.3.d)}$$

i.e. phase, $\Phi(t)+\phi_n$, is limited to $\pm\pi/4$ but variations greater than $\pi/4$ re-enter at 0 or $\pi/2$. This is represented as a sawtooth waveform with amplitude 0 to $\pi/2$ and repeating four times for every revolution of the received phasor. The rotating phasor has therefore been recovered with phase noise variance superimposed. We wish to obtain the expected value of $\hat{\Phi}_n(t)$. Hence :-

$$E[\hat{\Phi}_n(t)] = E[(\Phi(t) + \phi_n + \pi/4) \bmod \pi/2] \qquad \text{(Eq4.3.e)}$$

$$= E[\Phi(t) \bmod \pi/2] + E[\phi_n \bmod \pi/2] + E[\pi/4 \bmod \pi/2]$$

and

$E[\phi_n(t)] = 0$ therefore $E[\phi_n(t) \bmod \pi/4] = 0$

$$E[\hat{\Phi}_n(t)] = E[\Phi(t) \bmod \pi/2] + \pi/4 \qquad \text{(Eq4.3.f)}$$

The expected or mean value being defined as:-

$$E[x]= \int_{-\infty}^{\infty} xp(x)dx \qquad \text{(Eq4.3.g)}$$

where p(x) is the probability of x .

The expected value of $E[\hat{\Phi}_n(t)]$ is therefore the mean of the signal $\Phi(t) \bmod \pi/2 + \pi/4$. If the function $\hat{\Phi}_n(t)$ is filtered by a filter with impulse response h(t) the output becomes:-

$$y(t) = \hat{\Phi}_n(t) * h(t)$$

where $\hat{\Phi}_n(t) = \hat{\Phi}(t) + \phi_n(t)$ and $\phi_n(t)$ is a sampled function of a stationary stochastic process .

$$y(t) = \frac{1.}{2\pi} \int_{-\infty}^{\infty} \hat{\Phi}_n(\omega).H(\omega)e^{j\omega t}d\omega$$

$$y(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \hat{\Phi}_n(t)e^{-j\omega t}dt.H(\omega)e^{j\omega t}d\omega$$

$$y(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} [\hat{\Phi}(t)+\phi_n(t)]e^{-j\omega t}dtH(\omega)e^{j\omega t}d\omega$$

$$y(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty}\hat{\Phi}(t)e^{-j\omega t}dt + \int_{-\infty}^{\infty}\phi_n(t)e^{-j\omega t}dt\right]H(\omega)e^{j\omega t}d\omega$$

let $\hat{\Phi}(t)$ be a constant = Φ

$$y(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\left[\Phi\int_{-\infty}^{\infty}e^{-j\omega t}dt + \int_{-\infty}^{\infty}\phi_n(t)e^{-j\omega t}dt\right]H(\omega)e^{j\omega t}d\omega$$

now

$$\int_{-\infty}^{\infty}\phi_n(t)e^{-j\omega t}dt = \phi_n(\omega)$$

and $$\int_{-\infty}^{\infty}e^{-j\omega t}dt = 2\pi\delta(\omega)$$

therefore

$$y(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}[2\pi\delta(\omega)\Phi + \phi_n(\omega)]H(\omega)e^{j\omega t}d\omega$$

$$y(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}2\pi\delta(\omega)\Phi H(\omega)e^{j\omega t}d\omega + \frac{1}{2\pi}\int_{-\infty}^{\infty}\phi_n(\omega)H(\omega)e^{j\omega t}d\omega$$

as the bandwidth of $H(\omega)$ tends to 0 then

$$\frac{1}{2\pi}\int_{-\infty}^{\infty}\phi_n(\omega)H(\omega)e^{j\omega t}d\omega \text{ tends to } 0$$

and $$\frac{1}{2\pi}\int_{-\infty}^{\infty}2\pi\delta(\omega)\Phi H(\omega)e^{j\omega t}d\omega \text{ tends to } \Phi\delta(0)H(0)$$

where $\Phi\delta(0)$ represents a dirac impulse at DC of height Φ and $H(0)$ represents the DC gain function of the filter. If $H(0)=1$ then the output value of the filter :-

$$y(t) = \Phi \qquad \text{(Eq4.3.h)}$$

If the bandwidth of $H(\omega)$ does not tend to zero then the value of Φ is degraded by the inclusion of the noise term. If $\hat{\Phi}$ (t) is not constant then $\Phi(\omega)$ has a finite bandwidth and again the mean value is degraded. Eq 4.3.h shows that it is possible to recover the value of $\Phi(t)$ from a noise plus data phase signal. Practically the filter function does not reduce the bandwidth to zero and hence the random component of the noise is not reduced to zero but only close to it, depending on the bandwidth of the filter. This manifests itself as a random variation around the expected mean of the rotating phasor. The use of the filter introduces a time delay Td into the process. This effectively means that the estimated value of $\hat{\Phi}_n(t)$ lags the value of the rotating phasor by Td. If the rotating phasor's rate of change is slow compared to Td then this has little effect. However as the rate of changes increases then the estimated value $\hat{\Phi}_n(t)$ has an increasing phase error difference with $\Phi(t)$ and the estimation will then become ambiguous as this phase error increase to $\pm\pi/4$ radians for QPSK or $\pi/2$ for BPSK. This can be overcome by projecting the phase estimate over the delay time Td. This requires a knowledge of both the time delay Td and the estimate of the rate of change of $\Phi(t)$ i.e. the slope of the phase ramp or the frequency offset. The maximum possible offset frequency range for the mean phase filter system alone therefore occurs when the phase delay around

the phase filter loop is less than π/M :-

$$\phi_{delay} < \pi/M$$

$$2\pi f T_d < \pi/M$$

$$f < \frac{1}{2MT_d} \qquad \text{(Eq4.3.i)}$$

4.4. Frequency Offset Calculation

At the output of the $\Gamma^{-1}(.)$ function, for a received M'ary signal plus noise with a frequency offset, it has been shown that by the subtraction of an estimated phase, data removal and filtering that it is possible to update that phase estimate. If an output is taken from the $\Gamma^{-1}(.)$ function in parallel to this, and the data phase changes are removed from it by the use of a xM mod 2π function, a $M.\Phi_n(t)$ phase output can be obtained, where $\Phi_n(t)= \Phi(t)+\phi_n$. Against time on a linear scale this can be represented as a ramp with period T=1/Mfo where fo is the offset frequency, over a phase change of 0 to 2π radians; 0 and 2π positions are the same as the function is mod 2π and can be represented on a unit circle. It is required that an estimate for the rate of change of phase is to be found, $d\Phi_n(t)/dt$.

$$\frac{d\Phi_n(t)}{dT} = \frac{\Phi_n(t)-\Phi_n(t-T)}{T} \qquad \text{(Eq4.4.a)}$$

Due to the effect of noise on the phasor, $\Phi_n(t)$, there is a probability density distribution about the $\Phi(t)$. One method of obtaining the most accurate value of $d\Phi_n(t)/dt$ is by the evaluation of the sum of least squares error <29>.

The least square error slope, m, can be shown to be given by :-

$$m = \frac{\{n(\Phi(n)-\Phi(1))+ (n-1)(\Phi(n-1)-\Phi(2))+(n-2)(\Phi(n-2)-\Phi(3))+...\}}{2(n^2+(n-1)^2 +(n-2)^2 +...1)}$$

(Eq 4.4.b)

$n=(N-1)/2$ where N=total Odd number of samples.

Hence by sampling $\Phi_n(t)$ at regular points it is possible to calculate the best slope passing through these points. With the calculation of this slope value it is now possible to evaluate a projected value m.X, where m.X is defined in Eq 4.7.3.2.b for the rate of change of $\Phi(t)$ of the delay time of the mean filter.i.e. Calculation of expected value of $\Phi(t)$. By the addition of this projected slope value it is now possible to compensate for any known constant time delays within the system and estimate a value of $\hat{\Phi}(t)$ which is very close to that of the incoming rotating phasor. The final system model becomes that of fig 4.4.a. The advantage of placing the slope detection outside of the main loop is that once the slope has been correctly deduced then this aids the locking of the mean loop. If the slope detector is within the loop unless the system is locked the the projected slope can enhance the error in the estimated value and the system may never acquire lock.

4.4.1 Phase Estimator Recovered Phase Deviation

By analysis of the Final Phase estimator diagram as shown in Fig 4.4.a it can be shown that the deviation on the Estimated Carrier Phase can be given by:-

$$\sigma_T^2 = \frac{2.(m'.M)^2.\sigma_n^2}{L_f} + \frac{2\sigma_T^2 + \sigma_n^2}{L_p} \qquad \text{(Eq 4.4.1.a)}$$

where σ_T = the estimated carrier Phase, σ_n = the phase

deviation due to AWGN,m'= frequency offset slope multiplier, M = M'ary PSK (2 or 4) and L_f and L_p the number of symbols over which the frequency and Phase means are averaged over. i.e 16 for Phase coefficient = 15/16 and 128 for Frequency coefficient = 127/128.

This can be shown to be:-

$$\sigma_T^2 = \sigma_n^2 \left[\frac{2.(M.m')^2.L_p + L_f}{L_f(L_p + 2)} \right] \qquad \text{(Eq 4.4.1.b)}$$

For L_p = 16, L_f = 128, m'= 0.547, and M = 4 for QPSK

$$\sigma_T = 0.3\sigma_n$$

For BPSK m' = 1.094, M = 2

$$\sigma_T = 0.3\sigma_n$$

This is the same as for QPSK and is due to m'M being equal for BPSK and QPSK as expected. For an input signal to noise of 7 dB Eb/No for BPSK a Phase deviation (1σ) is equal to 19.5 degrees. The expected Phase estimator deviation is therefore 5.8 degrees.

4.5. Conversion from Phase Incoherent to a Phase Coherent System

With the system as described above in lock the error between the estimated value $\hat{\Phi}(t)$ and $\Phi(t)$ the rotating phasor is small. Neglecting the phase ambiguity created by the data removal process, the resultant value p(t) consists of the phase data information plus noise. Since the Amplitude of the phase data sample is known it is now possible to convert from the present polar representation to that of a rectangular representation by the use of the functions:-

$$P = A \cdot \cos p(t) \qquad \text{(Eq4.5.a)}$$

$$Q = A \cdot \sin p(t) \qquad \text{(Eq4.5.b)}$$

where P and Q are the data signals as before. Hence the Phase incoherently received signal has been transformed and processed to produce a phase coherent result. The ambiguity in the absolute phase must be resolved by the usual methods, i.e Differential encoding, use of an FEC codec, or the transmission of a unique word.

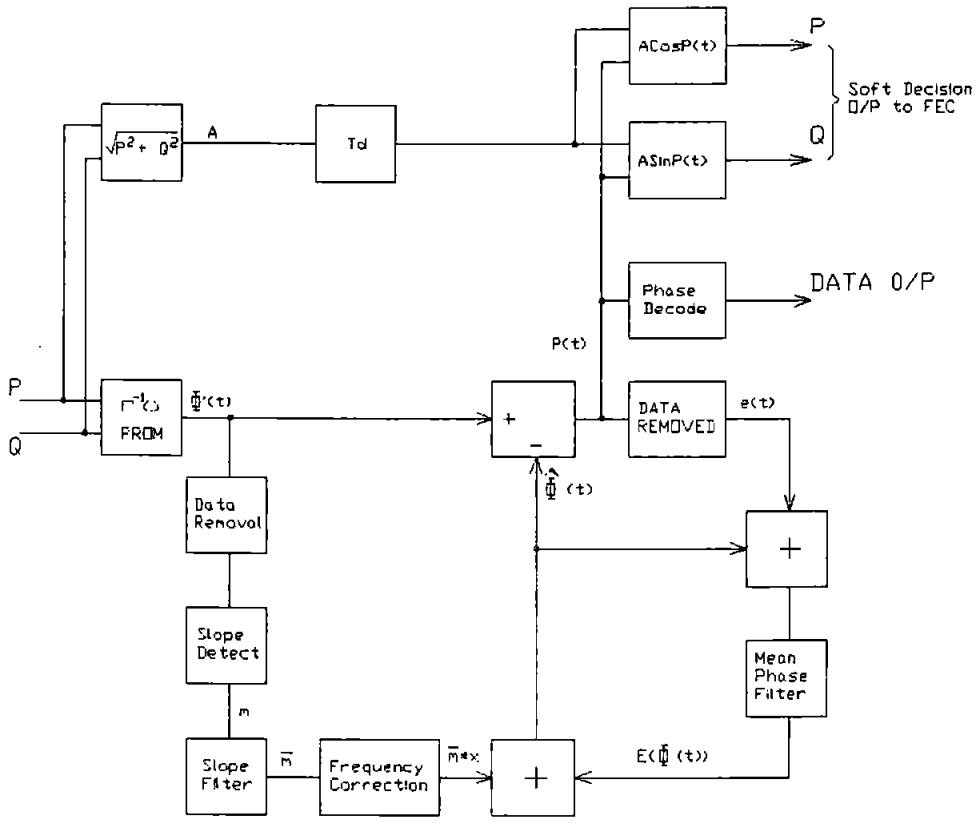


Fig 4.4.a Final System Diagram for Phase Estimator

4.6. Phase Estimator Mathematics

The operation of the Phase estimator has already been described, however the mathematics associated with the filtering operation must be considered. The arithmetic that is used relate to values constrained within 0 and 2π, i.e. a circular function, and not to values in the theoretical range of $-\infty$ to $+\infty$ as is usually the case in filtering. Let us consider the unit circle as in fig 4.6.a, the diagram is marked with degrees and quadrants and can represent the path taken by a rotating phasor. Consider a point P1 on the circle. If we add to P1 another point P2 which also lies on the circle the resultant Pr will also lie on the circle and can be represented by the formula:-

$$Pr = P1 + P2 \bmod 360 \qquad \text{(Eq4.6.a)}$$

The function mod 360 confines the result to lie on the circle. Hence the addition of two points can be simply achieved. To perform a subtraction around the circle another factor must be known. If we consider points P1 and P2 again and tried to calculate the difference between phasors we can obtain two results. One if the difference is taken along the route P2->90->P1 and another if the route is taken along P2->180->270->360->P1. Hence the filter must be able to distinguish which is the preferred route. From sampling theory it is known that a signal must be sampled at least at twice its highest frequency for the signal to be preserved without aliasing. The frequency that rotational speed around the circle represents is the inverse of the time taken for one complete revolution and hence twice this frequency indicates that the circle will be correctly sampled at points 180 degrees apart. If this is now the case it can be simply seen that it is the shortest route that produces the correct result.

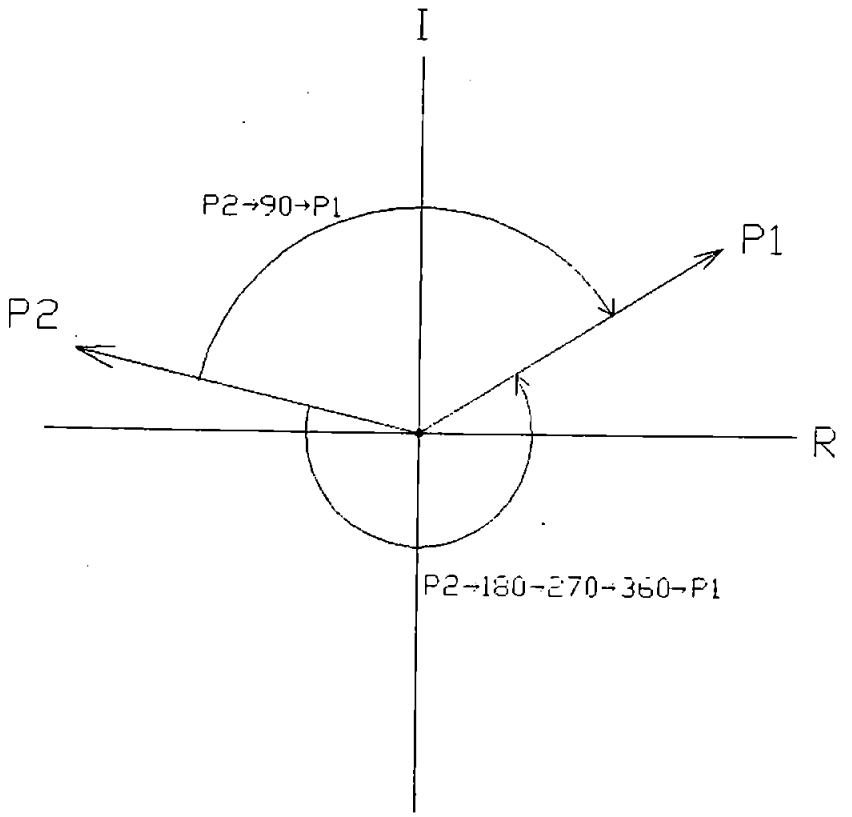


Fig 4.6.a "Shortest Route" Subtraction Around the Unit Circle

It is necessary to establish the arithmetic rules which always allows this to be the case. This can be achieved by the use of Complementary Arithmetic, for example consider the case of P1=32 degrees and P2 =211 degrees, the difference between the two points, minimum distance is 179 degrees.

$$Pr = P2 - P1 \qquad \text{(Eq4.6.b)}$$

In complementary arithmetic this could be calculated as

$$Pr = P2+(360-P1) \bmod 360 \qquad \text{(Eq4.6.c)}$$

ie

$$\begin{aligned} Pr &= 211+(360-32) \bmod 360 \\ &= 211+328 \bmod 360 \\ &= 539 \bmod 360 \\ &= 179 \end{aligned}$$

If P2 is increased to 213 degrees then the result becomes 181 degrees which in complementary form represent -179 degrees, hence it is possible to distinguish the direction of the minimum distance as well as its magnitude. The function Pr, eq4.6.b can be simply represented in digital circuitry by the use of 2's Complement arithmetic and the modulo function accomplished by allowing the overflow of adders not to be acknowledged.

4.7. Implementation of the Phase Estimator.

The phase estimator requires in its implementation a number of algorithms which will be considered individually.

4.7.1. The $\Gamma^{-1}(.)$ Function

The conversion from rectangular to polar coordinates is achieved by the use of two PROMs. These are pre-programmed with algorithms to convert the sign magnitude quadrature signals to a Amplitude and Phasor representation. The Γ^{-1} information is a look-up table which conforms to the algorithm :-

$$\Phi_a(PQ) = \Gamma^{-1}(Qn/Pn) \quad + x \qquad \text{(Eq4.7.1.a)}$$

where $\Phi_a(PQ)$ is the output phase value Φ for address a(PQ), P,Q are the sample values for the centre of the symbol period of the P and Q channels nth sample respectively. The function x =0,π/4 for BPSK and QPSK respectively, equivalently rotates the space diagram at the output of the Γ-1 function such that the symbol centre phase angles are decoded onto the 0,π/2,π,3π/2,2π axis for QPSK, and 0,π axis for BPSK if the transmitter is locked to the receiver. This allows the xM MOD 2π function to fold all phase errors around the 0 radian axis. The address conforms to the algorithm, for the two 6 bit sample values:-

$$a(PQ) = 64.Pn + Qn \qquad P,Q=0..63, \qquad \text{(Eq4.7.1.b)}$$

Hence a minimum 12 bit address prom is required for each data format with a 8 bit output word. This gives an output accuracy of 1.41 degrees and a error deviation of 0.26 degrees (table 4.7.1.a) . Increasing the output word width

has very little effect on the overall performance of the system since this is already approximately 48dB S/Nquant and the system is limited by the quantization of the input to the $\Gamma^{-1}(.)$ function. However the accuracy, and monotonicity of the output signal can be improved by the use of 7 bit Channel words at the address inputs (as shown in table 4.7.1.b).

n Bits Output Quantization	Phase error deviation in degrees
3	8.14
4	4.12
5	2.06
6	1.04
7	0.52
8	0.26

Table 4.7.1.a The Effects of Output Quantization on Phase in Rectangular to Polar Conversion

n Bits Input Quantization	Phase Error True Output	Deviation in degrees 8 bit Quantized output
4	8.94	8.94
5	4.06	4.06
6	3.07	3.08
7	1.25	1.28
8	0.37	0.45

Table 4.7.1.b The Effects of Input Quantization on Phase Output in Rectangular to Polar Conversion

It can also be seen from the table 4.7.1.a and 4.7.1.b that the total output phase deviation σ_{qo} conforms to the rules of power wise addition:-

$$\sigma_{qo}^2 = \sigma_{iq}^2 + \sigma_{oq}^2 \qquad \text{(Eq4.7.c)}$$

where σ_{iq} = phase error deviation due to input quantization

and σ_{oq} = phase error deviation due to output quantization. Hence an overall 16 kbyte PROM could provide the decoding of the $\Gamma^{-1}(.)$ for both BPSK and QPSK without any major increase in cost whilst producing a reduction in phase error variance which is additive to the noise created variance. The transformation to an amplitude function can be similarly performed using a PROM look-up table pre-programmed with the algorithm:-

$$A_a(PQ) = \sqrt{(Pn^2 + Qn^2)} \qquad \text{(Eq4.7.1.d)}$$

the address a(PQ) being given by Eq4.7.1.b as before.

The 8 bit output words from the $\Gamma^{-1}(.)$ PROM are in 2's Complement form to minimise the hardware for the processor implementation. The block diagram for the processor now being as in fig 4.7.1.a, and all values of degrees are scaled to an 8 bit representation of 256 levels or quanta, given by:

$$X \text{ quanta} = \text{Deg.m}/360 \qquad \text{(Eq4.7.1.e)}$$

where $m=2^n$,n=8 bits, and deg is in the range 0 to 360 degrees. The output from the $\Gamma^{-1}(.)$ PROM, at the data Symbol rate is fed into the Mean Estimator and the Slope Estimator and the operations carried out in parallel.

4.7.2. Mean Estimator.

The mean estimate is obtained following a number of sub-processes as described below.

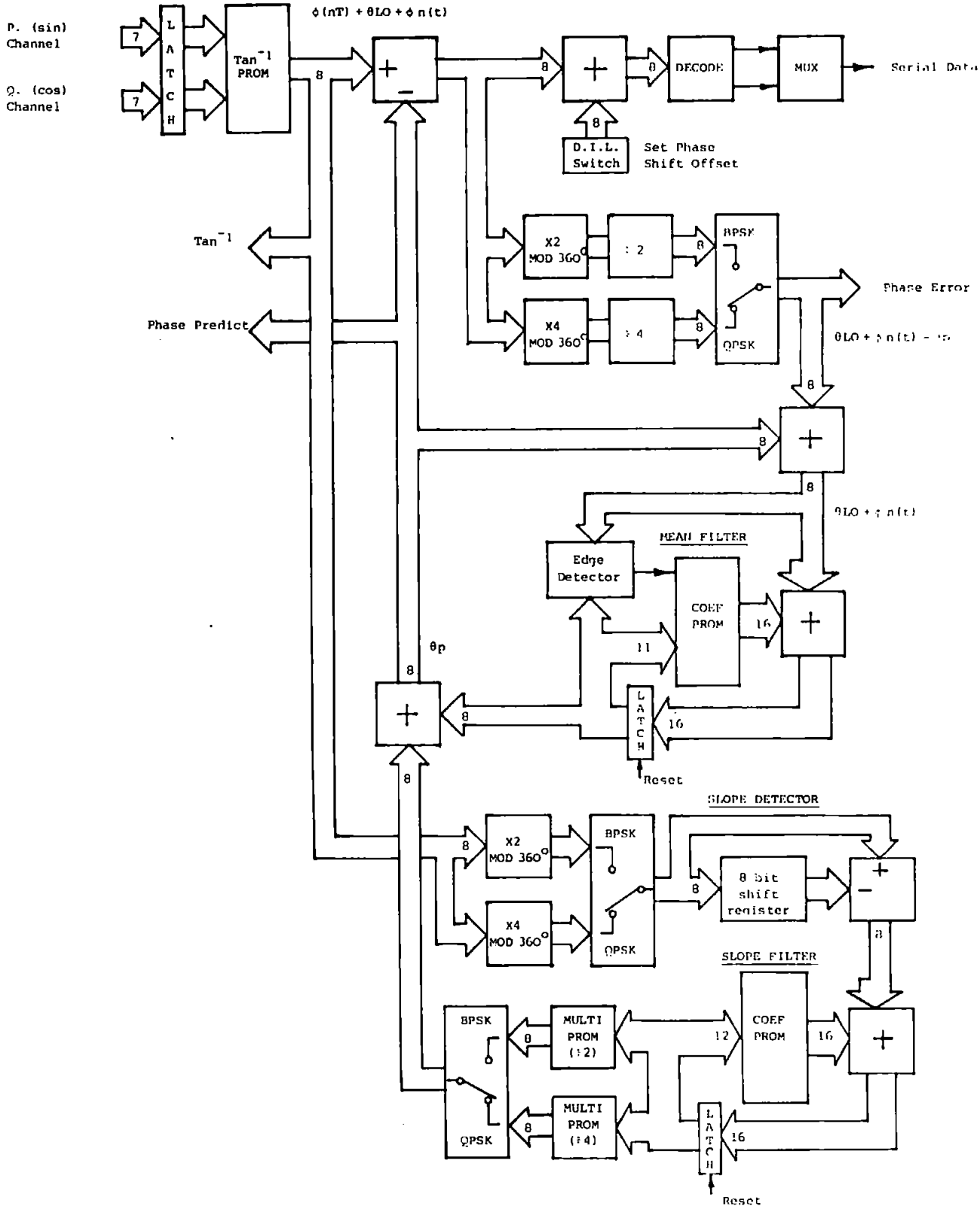


Fig 4.7.1.a Block Diagram of the Digital Phase Processor Implementation

4.7.2.1. Subtraction of Current Phase Estimate $\hat{\Phi}$ (nT).

The first operation is the subtraction of the current phase estimate $\hat{\Phi}$ (nT), from the output of the $\Gamma^{-1}(.)$ PROM, $\Phi'(nT)$. This is achieved by the use of 2's complement arithmetic. The subtrahend, $\hat{\Phi}$ (nT), is first complemented and then added to $\Phi'(nT)$ in an 8 bit adder formed from 2 off 74F283 I.C.s. The complementation is performed by inverting the sample word and adding one, the addition being performed as a 'carry-in' to the adders. The output is allowed to overflow if required fulfilling the requirements of the circular function.

4.7.2.2. Data Symbol Removal.

Data is removed by the use of the xM mod 2π operation, where M= 2,4 for BPSK and QPSK respectively. This is simply carried out for a 2's Complement number by the shifting of the data $\log_2 M$ bits to the left, i.e. multiplying by M. Bits that are shifted into position 9, MSB+1 are lost and bits shifted into position 0, the LSB are set to zero. The 8 bit output now represents the phase scale (+/-180) x M degrees so to readjust to the correct scale the result must be divided by M. This is done by bit shifting the word back to the right with the MSB's tied to MSB-1 for BPSK and MSB to MSB-1 and MSB-2 for QPSK. This can be implemented by the use of a selector IC 74F157 as in fig 4.7.2.2.a

4.7.2.3. Re-Addition of $\hat{\Phi}$ (nT)

This is achieved by the use of 2 off 74F283 adders. The 8 bit words being added together without the use of overflows to perform the circular function addition. The output is still an 8 bit word.

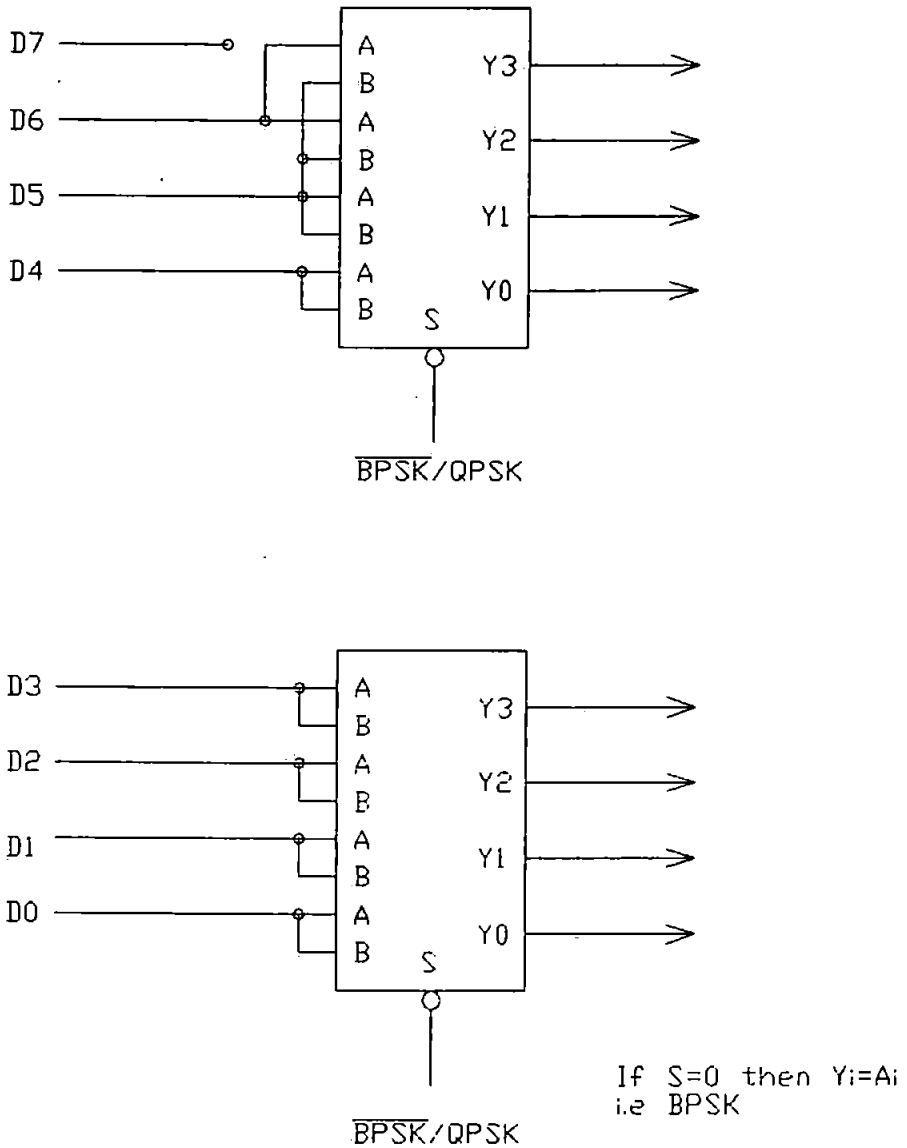


Fig 4.7.2.2.a Implementation of Data Removal From Received Phasor

4.7.2.4. Mean Filter

The mean filter is required to filter the rotating phasor without the filtering of the transition created by the MOD operator. The operation of the filter will be described in detail later. The filter is of the Infinite Impulse Response Type ,IIR, and consists of an 8 bit input word and an 8 bit output word.To maintain the accuracy of the calculation and to allow for the changing of filter coefficients the filter addition is carried out with 16 bits resolution. The input word of 8 bits is adjusted to 16 bit addition by the connecting the MSB's together. Shifting the position of this 8 bit input word relative to the 16 bit feedback word allows the maximum attainable value ,with reference to the input word, to change by factors of 2. This therefore allows the coefficient K to be simply changed by factors of 2 and hence allows the bandwidth of the filter to be simply changed. The accumulator answer is stored as a 16 bit word. The storage is performed by 2 off 74F374 latches, strobing the data when it has settled to a correct value. These latches also provide the one symbol period time delay of the filter. Enough bits, to be economically viable while preserving the accuracy of the filter, are fed back to the coefficient multiplying PROMs where the input address representation is multiplied by K, the coefficient. This is at present 11 bits in the mean filter with 1 bit representing a transform condition to modify the accumulator history. This bit effectively swaps the coefficient lookup table to one which has the accumulator data complemented before the coefficient multiplication was performed. The use of 2 off 2732 ,4096 byte PROMs is therefore required. Two PROMs are used in parallel to obtain a 16 bit output, with one PROM containing the High byte information and the other the Low byte information. It is this 16 bit word that is fed back to the adders.The output is taken from the accumulated

value stored on the latches and is adjusted back down to 8 bit representation by dividing this stored value by the coefficient denominator, m where K = m-1/m. By careful choice of m and K this can be a simple bit shift right operation, with zero entering the MSB positions. The 8 bit output word from this filter is now ready to be compensated for the frequency offset by the projected Slope Estimate.

4.7.3. Slope Estimation

Slope estimation or frequency offset estimation is performed by a number of sub processes as described below.

4.7.3.1. Data Removal

Data is removed from the $\Gamma^{-1}(.)$ output value $\Phi'(nT)$ by the same principle as in the Mean Filter section 4.7.2.2. The output is processed in the xM state to increase accuracy in the estimation process.

4.7.3.2. Slope Calculation.

Although it has been shown that the optimum fit to a series of points can be achieved on a least squares principle, the performance benefit of such a scheme is at a considerable cost in complexity and is therefore questionable. Another factor for consideration is that each increase in the accuracy of the 'fit' requires more points, increasing cost, and complexity. A simpler method is to take a first case approximation of the least squares approach given in Eq4.4.b and the calculation reduces to:-

$$m(nT) = \frac{\Phi'(nT)-\Phi'((n-k)T)}{k} \qquad \text{(Eq4.7.3.2.a)}$$

where n & k are integers. Careful choice of k can produce a simple solution,i.e let $k= 2^{L}$,L= 0,1,2.. and this is

easily implemented in hardware. Filtering of this function can therefore be used to to limit the bandwidth of such a signal to optimize the estimated value. The filter is performed by an IIR filter of similar design as before but since it is the difference between phasor samples that is being taken the transition does no longer exist. The IIR filter therefore requires no transition detection and 12 bits of the accumulator result can be used to access the multiplied resultant of the 16 bit output. Shifting of the input word relative to this 16 bit word also allows coefficient changes to be implemented as before. The output from the accumulator must then be scaled and projected forward in time to compensate for all time delays introduced around the mean estimator by :-

$$\delta\Phi(nT) = \frac{m(nT).T_{td}}{k.M}$$

(Eq4.7.3.2.b)

where $T_{td} = T_d + 2T + 0.5T$, the total time delay in Mean Estimator path; T_d is the delay due to the Mean Estimate filter; 2T is the extra delays in the Mean Estimator due to the digital logic timing constraints in this case; 0.5T allows calculations of the value to the centre of the next sample period with k = the number of samples over which the difference is taken and M compensates for the xM scaling factor. This is implemented by the use of a scaling PROM using 12 bit input addresses with the output given by the representation of the input address value being multiplied by the factor $d\Phi(nT)/dk$ and adjusted to give an 8 bit output. The 8 bit output word can now be added to the Mean Estimate to obtain the compensated phase estimate $\hat{\Phi}((n-1)T)$.

4.7.4. Conversion from Polar to Rectangular Representation.

Having removed the rotating phasor by the subtraction of the instantaneous phase estimate the resultant signal represents the phase positions of the data plus a noise component. Since the Amplitude of this phasor is also known it is possible to reconvert back to a rectangular representation by the use of the functions given in eq4.7.4.a/b. These can be implemented by the use of two PROMs pre-programmed to form a look-up table containing the functions:-

$$Pn(A,\Phi) = An.\ Cos\ \Phi n \qquad \text{(Eq4.7.4.a)}$$

$$Qn(A,\Phi) = An.\ Sin\ \Phi n \qquad \text{(Eq4.7.4.b)}$$

where $Pn(A,\Phi)$ and $Qn(A,\Phi)$ are the Amplitude values of the P and Q data samples at the nth instant given by the address created by the functions A and Φ, such that

$$Address(A,\Phi) = 256.A + \Phi \qquad \text{(Eq4.7.4.c)}$$

This indicates that a 14 bit address line is required and hence a memory space of 16 Kbytes. This will then give a soft decision output with 256 levels. This is considerably more than the accuracy normally required for a soft decision FEC decoder.It has been shown that there is only a 0.25 dB degradation in performance if only 3 bit (8 level) quantization of the soft decision output is used compared to the unquantized output <30>. However,the interface to such a decoder could possibly be of an analogue nature and hence conversion form Digital to analogue may be required. Accuracy can be preserved and at minimal extra cost since 8 bits are readily available from the PROMs.

4.8. An IIR Filter Filtering a Circular Function

The circular function representation although being limited to the range 0-360 degrees by the MOD operator is in fact representing a continuous linear slope, proportional to the frequency offset, stretching to infinity . It is only the need to represent this as a finite number that reduces it to a saw tooth waveform produced by the MOD operator. It is required that the rotating phasor is filtered to remove phase variations due to noise but that the slope is preserved. If we simply filter this waveform in the conventional manner then the bandwidth of the filter will cause the transition point at 0/360 degrees to be modified from a sharp transition to that of a gentle transition with the introduction of a smoothing of the waveform. This would not occur if the waveform was a continuous slope to infinity and hence the output waveform from such a filter function would be incorrect. We must therefore modify the filter function such that the 0 and 360 degree points are classed as the same point and hence no smoothing of the transition occurs, only the reduction in Phase variations on the slope, caused by noise. To achieve this the transition edge must be detected whenever it occurs, not just once per cycle since noise can cause the phasor to oscillate around any point and hence the transition can occur many times unpredictably. Careful attention to the signal processing carried out here is important otherwise unnecessary cycle slip events will be introduced. With the detection of this transition point the history of the slope must be modified such that it appears that the information that is contained within the memory of the filter now comes from a continuous line and not the sawtooth ramp. The requirement to modify the history of the waveform stored by the filter must be able to be achieved on an instant to instant basis and the most suitable form of filter for this type of operation is that of a digital single pole,

recursive, Infinite Impulse Response filter or IIR filter. Other forms are not ruled out but can have enormous complexity with little apparent benefit. This is as depicted in fig 4.8.a, the small signal (1 σ phase error <45 deg) transfer function of such a filter being given by:-

$$H(Z) = \frac{1}{1 - K.Z^{-1}} \qquad \text{(Eq 4.8.a)}$$

substituting $Z^{-1} = e^{-j\omega T}$ Eq 4.31 becomes:-

$$|H(W)|^2 = \frac{1}{1-K^2 - 2.K.\cos \omega T} \qquad \text{(Eq4.8.b)}$$

where ω = angular frequency, T = sample rate of IIR filter (equal to the modem symbol rate), and K = feedback coefficient of the form K= (m-1)/m. By varying the value of the coefficient K it is therefore possible to adjust the frequency response of the filter as shown in fig 4.8.b, and can be shown (Appendix C) to have a constant time delay, Td, for any filter coefficient K of:-

$$Td = \frac{-K.T}{1 - K} \qquad \text{(Eq4.8.c)}$$

Hence with Td constant it is possible to correctly calculate the projection factor for the slope coefficient. From Eq4.3.i ,this therefore gives a maximum mean phase filter tracking range for QPSK at 256Ksymbols/sec of less than 1032 Hz. The large signal response due to frequent transition errors will be quite different from the above due to the non linear nature of the filter arithmetic. However this condition only pertains to SNR values i.e. -1 dB, much less than the operating point i.e 2 dB and hence the small signal transfer function can be regarded as valid for the range of interest.

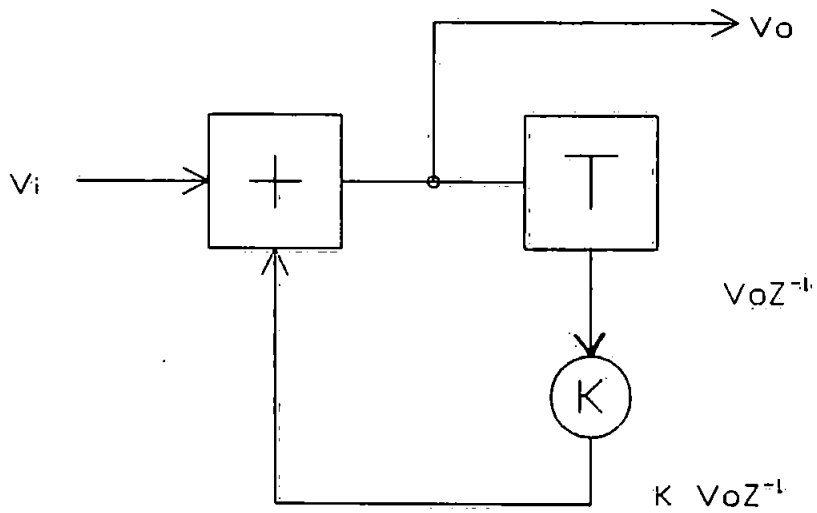


Fig 4.8.a Single Pole IIR Filter

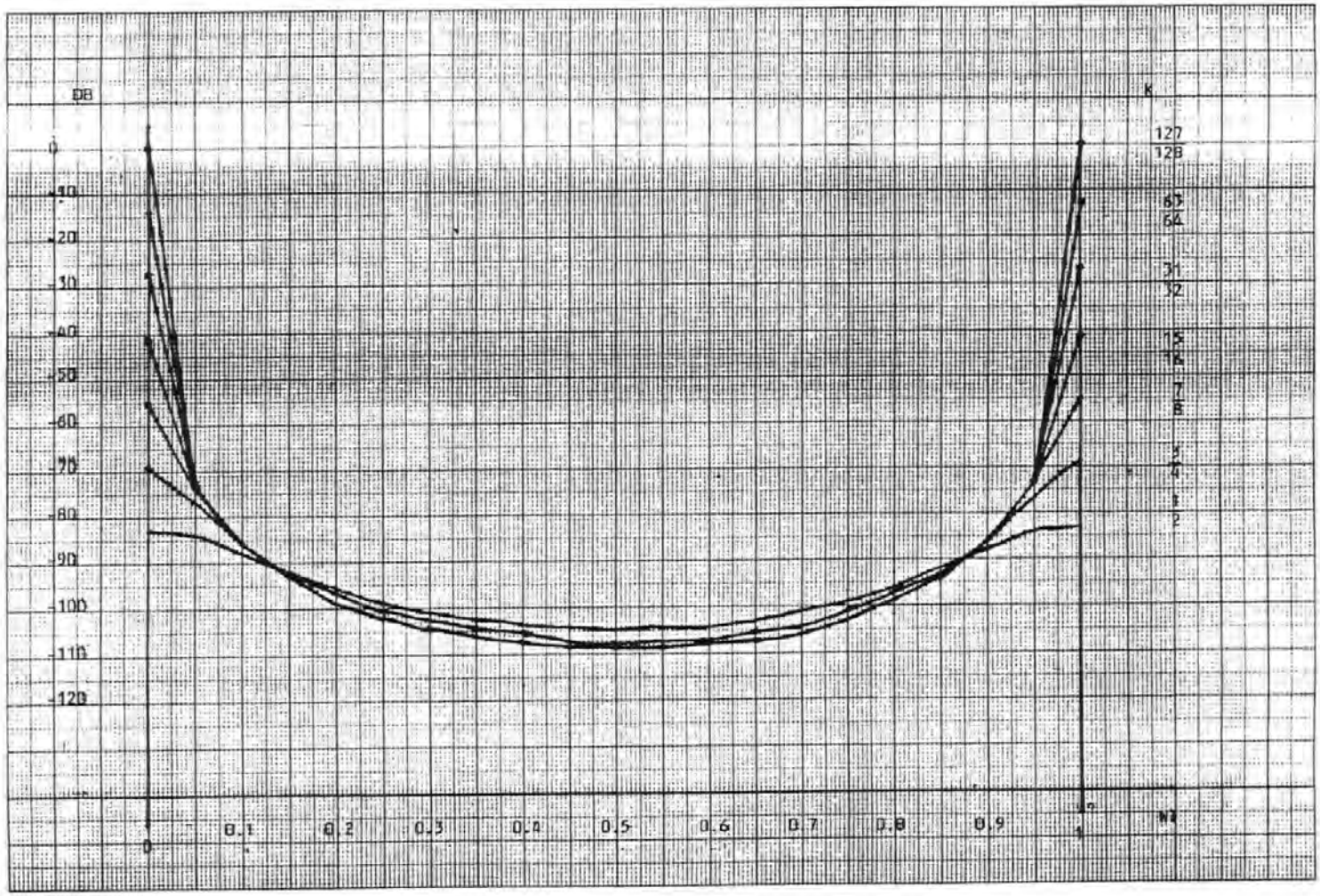


Fig 4.8.b Frequency Response of Single Pole IIR Filter for Values of Coefficient K

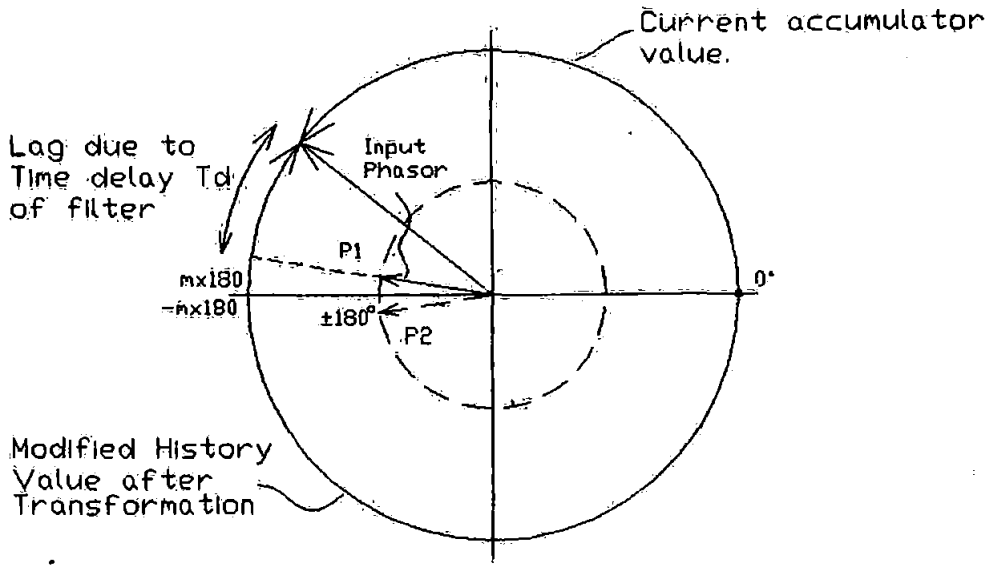
4.8.1. Transition Transformation.

To enable the filter to operate without smoothing of the sawtooth waveform generated by the MOD function it must be possible to detect the fact that a transition is occurring and modify the accumulator result in such a manner that this appears to change the history of the input function such that no transition appears to occur. Once the transition has been detected the modification of the accumulator must take place. Consider the circle of fig 4.8.1.a the inner circle represents the sample values of the rotating phasor that are permissible and the outer circle the accumulator values which will tend to a maximum value of m times the maximum input value for an infinite series addition. For convenience the transition area is transferred to the +/- 180 degree point and hence the maximum value for the accumulator is +/-m x 180. This accumulator value will also lag the rotating phasor by the time delay Td. Hence if the phasor was approaching the 180 degree point in a positive direction i.e. anticlockwise. Then the accumulator value would be lagging behind it but also approaching m.180 from a positive direction. As the transition is reached and the phasor becomes negative the accumulator value, which is positive, must be transformed to appear to the new sample that the phasor has been approaching from more negative values but still going positive. This can be achieved by the transformation:-

$$\hat{A}(nT) = \begin{cases} A(nT) - m.360 & \text{for } 0 =< A(nT) =< m.180 \\ A(nT) + m.360 & 0 > A(nT) > -m.180 \end{cases} \qquad \text{(Eq4.8.1.a)}$$

The algorithm which detects whether the history and input signal are on opposite sides of a transition boundary is defined as:-

$$Ed = (Vo/m)Z^{-1} - Vi \qquad \text{(Eq4.8.1.b)}$$



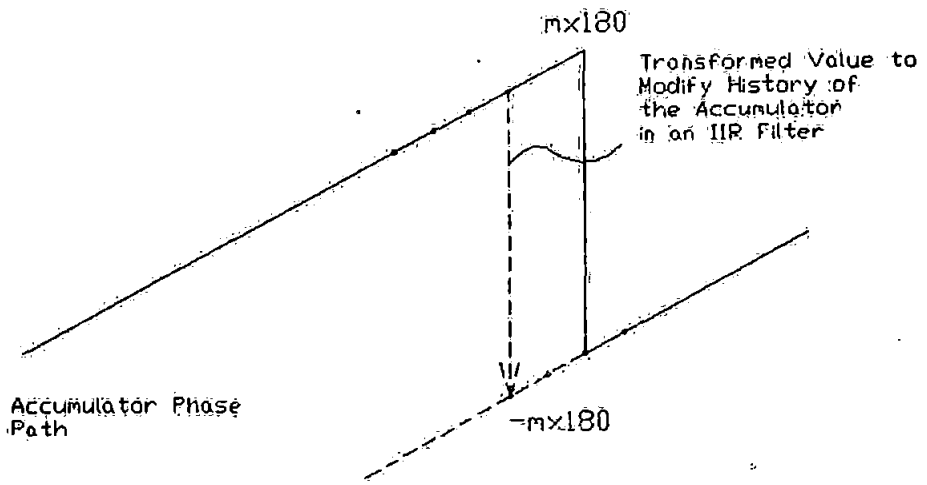


Fig 4.8.1.a Transition Detection Around the Unit Circle

This can therefore be used to create a 'History Modifier' which modifies the 'sense' of the filter history dependent on the Edge detection function Ed such that:-

If Ed > 180 then C=1
if Ed <-180 then C=-1
Else C=0

By the use of this transformation with the detection of the transition area the it is now possible to low pass filter the rotating phasor as if the transition did not occur. Considering the following conditions:-

I/P = Vi in range +/- 180
Multiplication Coefficient K = (m-1/m)
O/P = Vo in range +/- Ma , where Ma = m x Vi_{max}

The single pole IIR Filter is defined by :-

$$Vo = Vi + K(Vo + C.Ma)Z^{-1} \qquad \text{(Eq4.8.1.c)}$$

$$Vo = \frac{Vi + K.C.Ma\ Z^{-1}}{1-KZ^{-1}} \qquad \text{(Eq4.8.1.d)}$$

where C is the History Modifier.

4.8.1.1. Digital Implementation of Transition Detection

The 2's Complement numbering system being used throughout the phase processor allows the performing of addition and subtraction conforming to the rules of the circular function and the implementation of sign magnitude arithmetic by considering the operation of overflow bits. The detection circuit is required to detect the transition created by the MOD operator. To detect this transition it is only required that the difference is taken between an input sample value and the appropriately scaled accumulator value to detect if the sample value has passed the transition while the accumulator has not. This can be undertaken on a sign magnitude basis and is defined by equation eq4.8.1.b. The magnitude, if greater than 180 degrees, suggests that a transition has been reached while the sign indicates the direction that the phasor has moved, i.e. from the second quadrant to the third or vice versa. Noise will corrupt this decision but the probability of a noise sample causing a change of 180 degrees, without the transition being present, is less than 2.2 e-7 at 3dB Eb/No. It is the following of these transitions which allow the filtered phase output in a xM MOD 360 degree format i.e 4 times the carrier phase change/symbol for QPSK, to be converted to the linear ramp representing the phase change/symbol of the nominal carrier. Without careful monitoring of the transitions the phase estimate will cycle slip in relation to the received carrier phase and produce errors in the demodulation of the data. This is analogous to the squaring circuit used for the recovery of a carrier signal in BPSK. The output is twice the carrier frequency and hence must be filtered and divided by 2 before use in the local oscillator. If this division by 2 is undertaken in the presence of noise cycle slips could occur and hence the recovered data would be in error.

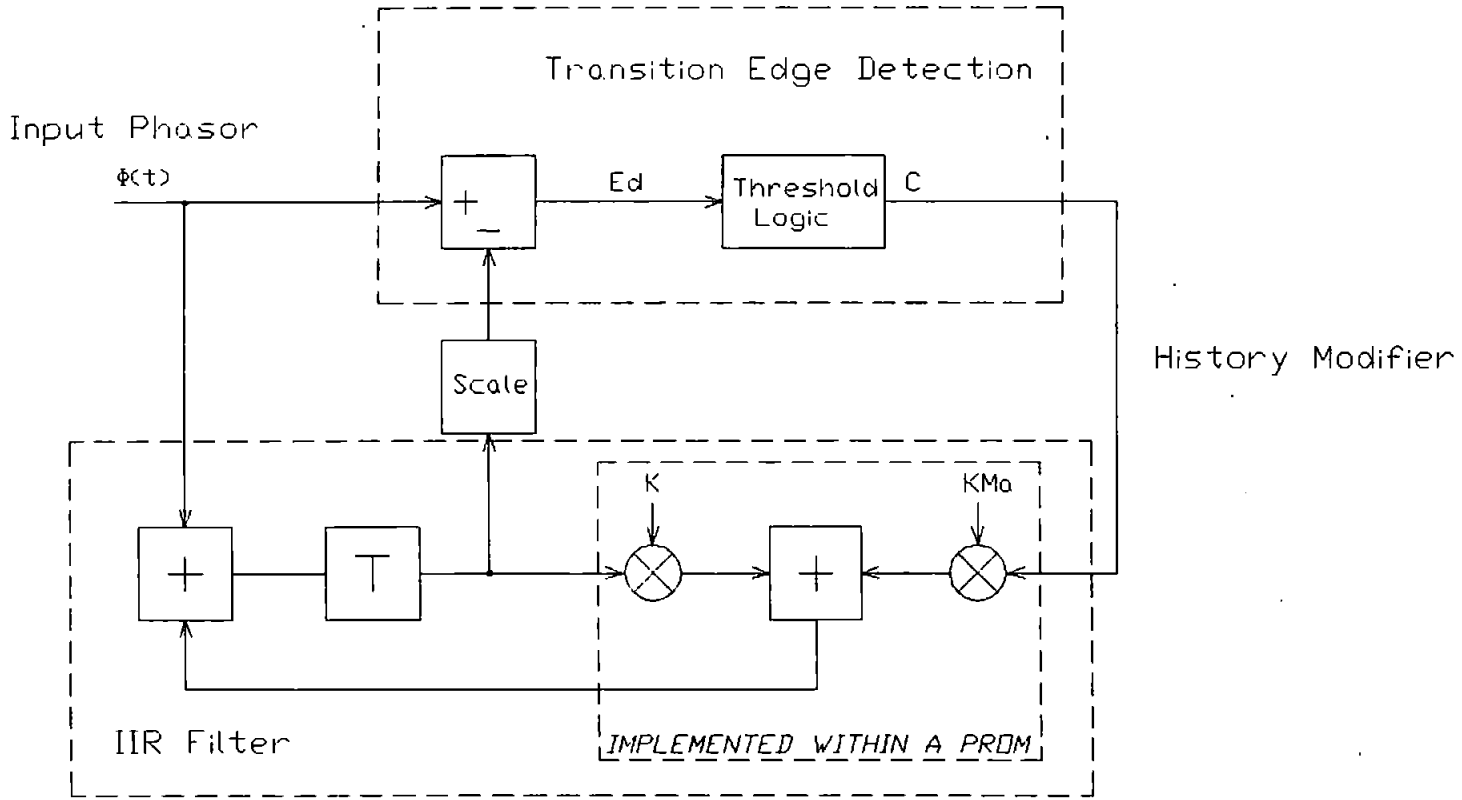


Fig 4.8.1.1.a Transition Detection Circuit used in Phase Processor

4.8.2. Digital Implementation of the IIR Filter.

The IIR filter is implemented using a 16 bit accumulator with the Time delay being combined into a 16 bit latch to strobe the data once the result is stable. The output from the latch is then fed back to a 12 bit PROM which acts as a coefficient multiplier and also provides the transformation of the accumulator result. These results are pre-programmed into a look-up table with the following memory map, table 4.8.2.a. With the address given by :-

$$\text{Address} = H.2048 + \text{Acczmn1} \qquad \text{(Eq 4.8.2.a)}$$

where H=0 for C=1 and H=1 for C=-1. The value of 'Multin' is the decimal representation in sign/ magnitude form of the 2's complement accumulator delayed by 1 symbol period and truncated to an 11 bit address word, Acczmn1, and hence '2048' is the maximum number of values required. In the Off line calculation process for the lookup table, this word is converted from 2's complement to sign magnitude, multiplied by the coefficient and history modified as required. It is then reconverted from a sign magnitude representation to a 2's complement representation in a 16 bit wide word, the two bytes being stored as a High byte and a Low byte answer for the same input address.

I/P address	Operation
FFF C00	multin= -1024 to -1, output=(multin +2048)x K
BFF 800	multin = 0 to 1023 output = (multin-2048)x K
7FF 400	multin = -1024 to -1 output = multin x K
3FF 000	multin = 0 to 1023 output = multin x K

Table 4.8.2.a. Memory Map for 12 bit Addressed Coefficient Multiplication PROMS inclusive of Accumulator Transformation.

This memory map can be extended for greater accuracy by the use of larger PROMs and the appropriate scaling of the input address representation and adjustment factor. It should be noted that whatever the number of bits used to represent specific results that the answer is a scaled representation of 0 to 360 degrees ,or multiples of 0 to 360 degrees. Care must be taken to ensure that the correct representation is maintained. One method of arranging this is to enter and leave the filter as 8 bits but maintain all calculations within the filter at 16 bits. Hence a 12 bit representation in decimal scaled to a 16 bit word will have a maximum of 65520 ($4095 \times 2^{16-12}$) and 1 bit increments of the 12 bit word will produce 16 increments of the 16 bit representation. The output, from the filter, is taken from the latched accumulator value and scaled to 8 bits by bit shifting right. This can only be performed accurately because the coefficient K is of the form $K=(m-1)/m$, where $m=2^{L}$, and L=0,1,2,.. . This results in a series of possible

coefficients of 1/2, 3/4, 7/8, 15/16, 31/32, 63/64, etc. Practical values range 15/16 to 127/128 and these have been found to be the most appropriate for the phase estimator to date.

4.9. Computer Simulation of Phase Estimator.

4.9.1. Introduction

To investigate the performance of the Phase Estimator and to allow for a degree of optimization to take place the operation of the Phase Estimator was simulated by Computer. The computer used was an IBM PC-AT ,Containing 512 Kbytes of RAM and a Mathematical Co- Processor to increase the speed of floating point calculations. The Simulation was written in the Pascal language implemented by Prospero Software Ltd. The Pro Pascal compiler complies with all the requirements of ISO 7185, level 0 with extensions for string handling, double-precision floating point arithmetic, random access files and separate segment compilation. The compiler is a true compiler generating native machine execution code <31>.

The simulation consists of a number of parts:-

1) The generation of Phase noise statistics .

2) The use of these statistics to evaluate the performance of the Simulated Phase Estimator.

3) The Display of the Phase noise statistics and the Estimated Phase statistics as Probability Density Functions.

4.9.2. Generation of Phase Noise Statistics

To evaluate the performance of the Phase Estimator the effect of the received noise on the filtered and quadrature down converted signal must be examined so that an accurate model of this noise can be produced while also being able to relate it to the standard measurement for data signals of Eb/No, the ratio of energy per bit to the noise spectral density. To enable this to be carried out the effect of the quadrature Matched Filters on the received noise power must be assessed. To enable the same data to be used for all forms of M'ary PSK the input signal to the quadrature down-converter is expressed as a value of Es/No,where Es is the energy per symbol. The bit energy can be related to the symbol energy by:-

$$Eb = Es / S \qquad \text{(Eq4.9.2.a)}$$

where S = number of bits per symbol. It can be shown that the output SNR from each matched filter at the correct sampling point, is equal to 2Es/No.

P channel

signal energy o/p = $Es.Cos^2 \phi(t)$ (Eq4.9.2.b)

noise density o/p = No/2

Q channel

signal energy o/p = $Es.Sin^2 \phi(t)$ (Eq4.9.2.c)

noise density o/p = No/2

Therefore:-

$$\text{total Signal energy} = Es.Cos^2 \phi(t) + Es.Sin^2 \phi(t)$$

$$= Es(Cos^2 \phi(t) + Sin^2 \phi(t))$$

$$= Es$$

$$\text{total Noise density} = No/2 + No/2 = No$$

Therefore combined Signal to noise = Es/No

(Eq4.9.2.d)

Hence the energy per symbol,per filter is increase by a factor of 2, due to the matched filtering but the combined output SNR for the two quadrature filter outputs remains constant at Es/No. With this information it is now possible to numerically calculate the probability density function of the Phase noise, correctly scaled to the input Es/No value. This makes use of the independent quadrature noise components and their individual Gaussian PDF's, by the use of a rectangular to polar conversion.

4.9.2.1. Probability Density Function of White Gaussian Noise

Consider the time representation of bandpass Gaussian noise <5>, if the bandwidth of the noise is small compared with that of the centre frequency it is possible to represent the noise components as phasors and hence in the form:-

$$n(t) = n_c(t)Cos\ W_c t - n_s(t)Sin\ W_c t$$

(Eq4.9.2.1.a)

and it can be shown that this representation can be

extended to low pass noise and that the mean-squared value of n(t) equals the mean-squared value of the cosine noise term and equals that of the mean-squared sine noise term, i.e.

$$\overline{n^2}(t) = \overline{n_c^2}(t) = \overline{n_s^2}(t) \qquad \text{(Eq4.9.2.1.b)}$$

From Eq4.9.2.1a the mean-squared value of bandpass noise is given by:-

$$\overline{n^2(t)} = \frac{\overline{n_c^2(t)}}{2} + \frac{\overline{n_s^2(t)}}{2} \qquad \text{(Eq4.9.2.1.c)}$$

Hence the noise is equally divided between the phase and quadrature channels and can be represented in phasor form as in fig 4.9.2.1.a. If this quadrature phasor is added to the signal phasor, assuming the signal phasor is 0 degrees, or that all data has been folded onto the 0 degree axis by a linear process, then the resultant deviation from 0 degrees is the phase noise error. If the deviation from nominal is calculated for a series of sample values of nc and ns taken from the Gaussian distribution, and the probability of this deviation occurring , the probability density function of phase error can be found.

Since the pdf of each of the noise components is an independent gaussian random variable, the probability of an event taking place which is dependant on two statistically independent functions is given by:-

$$P(A.B) = P(A).P(B) \qquad \text{(Eq4.9.2.1.d)}$$

Hence for any x_i, x_j, where x_i =i th sample of $n_s(t)$, x_j= jth sample of $n_c(t)$:-

$$\phi(i,j) = \Gamma^{-1}\left[\frac{x_i}{A + x_j}\right] \qquad \text{(Eq4.9.2.1.e)}$$

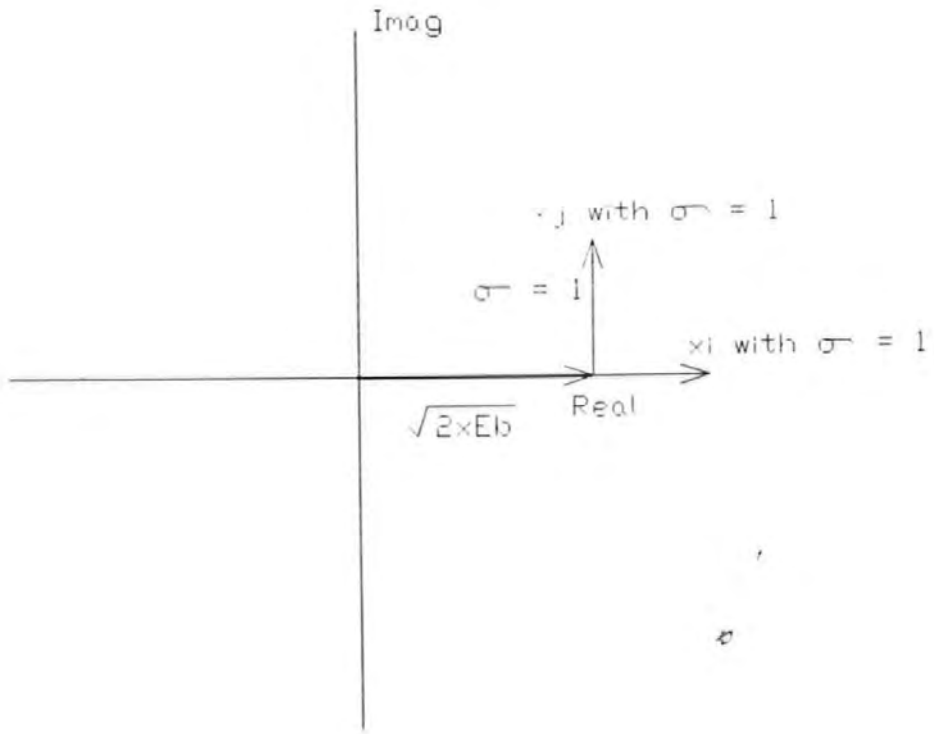


Fig 4.9.2.1.a Phasor Representation of Quadrature Noise Components

where A= rms signal Amplitude. The probability of ø(i,j) occurring :-

$$P(\text{ø}(i,j)) = P(x_i).P(x_j) \qquad \text{(Eq4.9.2.1.f)}$$

and the probability of ø=x is given by the summation of all values of $P(x_i).P(x_j)$ producing the result ø=x:-

$$P(\text{ø}=x) = \Sigma_i \Sigma_j \; P(xi).P(xj) \; \Big|_{\text{ø}=x} \qquad \text{(Eq4.9.2.1.g)}$$

If this is carried out over the range of x= -π to π the pdf of the phase error due to noise can be evaluated. It is therefore required that the Pdf of the noise samples x_i, x_j can be individually calculated and scaled for a particular value of Es/No. The probability of a sample x is given by the formula <5>:-

$$p(x) = \frac{e^{-(x-m)^2/2\sigma^2}}{\sqrt{(2\pi)}\sigma} \qquad \text{(Eq4.9.2.1.h)}$$

where m, σ^2 are the mean and variance respectively. The factor $1/\sqrt{(2\pi)}\sigma$ is to produce unit area under the curve. The function given in equation 4.9.2.1.h can be numerically evaluated by representing sample values of x in terms of σ,(since white gaussian noise m=0), and normalizing σ = 1. After calculating for a range of x say from x=-5σ to + 5σ, the cumulative distribution should tend to unity. However this is not the case due to its finite number of samples and hence the function must be scaled by the inverse of this cumulative value i.e.

$$F(x) = \sum_{x=-5\sigma}^{x=5\sigma} e^{-x^2/2\sigma^2} \qquad \text{(Eq4.9.2.1.i)}$$

and

$$C = 1/F(x) \qquad \text{(Eq4.9.2.1.j)}$$

Hence equation 4.9.2.1.h becomes:-

$$p(x) = \frac{e^{-x^2/2\sigma^2}}{C} \qquad \text{(Eq4.9.2.1.k)}$$

The probability of a 5σ event occurring is 1E-11 and a 6σ event is 2E-16. A 5σ event range is therefore considered to cover the range of probabilities of interest.It is therefore now possible to calculate the individual probabilities of samples x_i, x_j occurring and hence the probability of them occurring together by the use of equation 4.9.2.1.f. It is now required that the value of ø(i,j) be calculated for a particular value of Es/No. For this to be achieved the value of Es must be considered in terms of σ. If a value of Es/No, the signal to noise ratio of a symbol phasor, is given as x dB then this can be converted to a power ratio and split into its quadrature components. Scaling by a factor of 2, for each quadrature channel, allows σ to be maintained equal to unity. Hence the value of ø(i,j) can be calculated correctly scaled for values of Es/No and hence evaluated for Eb/No.

By the use of these techniques the pdf of ø(i,j) can be evaluated over the range -π to +π. To obtain an accurate representation of this pdf it is necessary to evaluate a large number of points and to enable this to be economically achieved within the confines of the computer memory the a Histogram of the pdf is calculated. This results in all values of ø(i,j) that fall within certain cells to be summed together. The cell width defines the accuracy to which an error function can be calculated, and the number of event results within a cell improves the accuracy of the probability calculation for that cell. It was found that a cell size of 0.5 degrees and a noise step range from 5σ to -5σ in 0.03σ steps produce a accurate

answer when compared to standard error function tables. The results obtained are stored on disc and can be called from other programs to enable the results to be plotted or the simulation of the Phase Estimator evaluated.

4.9.3. Phase Estimator Simulation.

To evaluate the performance of the phase estimator a computer simulation was produced that simulated the operation of the hardware design but with the added flexibility of allowing changes in various functions to be performed. The simulation is constructed within a shell of menus which allow the flexible functions of the simulation to be initialized. The menus include various trapping functions and error counting facilities. The phase noise samples are obtained from the pdf of the phase error for the particular noise level under investigation. The pdf of the SNR of interest is read from the disk and stored as an array. This array is then adjusted and weighted such that uniformly distributed sampling of the array produces noise samples of random deviations with an overall distribution given by the initial pdf. This is achieved by having the area under the pdf being equal to unity. By the generation of a set of Uniformly Distributed Number's (UDN's),i.e numbers with uniform distribustion, between zero and unity it is possible to integrate the pdf of the noise to a point where the integral is equal to the UDN. The histogram cell in which this point rests therefore gives the appropriate phase noise point. This can reproduce the correct bi-variate distribution without inhibiting the possibility of low probability samples. This is only the case over a large number of samples and hence the simulation must run for a long period of time and be monitored such that it can be determined when the result has settled to an accurate enough value. Jeruchim <18> provides a confidence limit estimation on the Bit Error Rate and suggests that for a BER of 10^{-k} for a 99% confidence to achieve a small error

deviation in BER, 10^{k+2} bits must be observed. Hence for an error rate of 1 $x10^{-6}$, an observation of 1 $x10^{8}$ bits is required. This of course requires a large amount of computational power and time. However the area of interest is more towards that of low Eb/No's and hence high error rates. This is therefore not a significant limitation. Once the noise samples can be created and distributed, according to the original pdf, then the simulation can add these noise samples, when required, to a set of samples which describe the operation of the signal,i.e. whether the simulation is operating on a coherent or incoherent signal. The Phase Processor simulation then manipulates these samples and decodes the results into phase angles. The phase results that are produced can be displayed and stored to enable errors or cycle slips to be evaluated. An error count can also be performed from the simulation and hence the BER calculated. The simulation also stores the estimated phase $\hat{\Phi}(t)$ and this can be evaluated as a variance of the estimated value to the rotating phasor value $\Phi(t)$. This result is also stored on Disk to allow plots to be produced of this variation. The operation of the Phase Estimator Simulation is not included in this account but a full program listing can be seen in Appendix D.

4.9.4. Display of Phase Error Statistic

The results obtained of the phase estimate errors and the actual phase values used by the simulation are stored to disk to allow for post- processing along with the calculated phase error distribution. This post-processing is mainly to allow the result obtained to displayed in a fashion which can be readily understood i.e. in a graphical form. Let us consider the phase noise pdf as calculated. The Phase estimate results can be processed in a similar

fashion. The Plot program is again menu driven and allowing the plot requirements to be initialized. This can include magnification of the plot, multiple or single plots and filtering of the distribution.

4.9.4.1. General Description of Pdf Distribution Plot Program

The results of the phase noise pdf calculation are stored to disk with a heading consisting of a number indicating the length of the file, the cell width in degrees,and the standard deviation of the pdf. It is this information which is first read back into the plot program to enable the program to correctly plot the graph. This is followed by the reading into an internal array of the data describing the pdf. With this information all available to the program the pdf is plotted to scale and annotated with the rest of the data.

4.9.4.2. Smoothing Function

Due to the random sampled nature of the results if these are plotted directly the graphical result is is not always clear. To improve the display the data can be filtered by a function operating over a number of samples. This is achieved by the use of an algorithm which representing a single pole, IIR filter response with coefficient K in the form :-

$$K= m-1/m \qquad \text{(Eq4.9.4.2.a)}$$

where m=1,2,3...10. and

$$Y(i) = \frac{Y(i-1).K + X(i)}{m} \qquad \text{(Eq4.9.4.2.b)}$$

where Y(i) is the filtered output sample and X(i) is the current input sample.

4.9.4.2.1. Magnification

In the x1 magnification mode the pdf displayed covers the range from - 180 to 180 degrees. This is often too large a range for large values of Eb/No i.e.Eb/No >8dB and hence it is desirable to magnify the plot. The range can therefore be expanded by an expansion factor in the sequence 1,2,3,5,10. Since the mean is always zero this occurs around the 0 degree point and hence expands the range of peak distribution. An example of the output from the Plot program can be seen in fig 4.9.4.2.1.a.

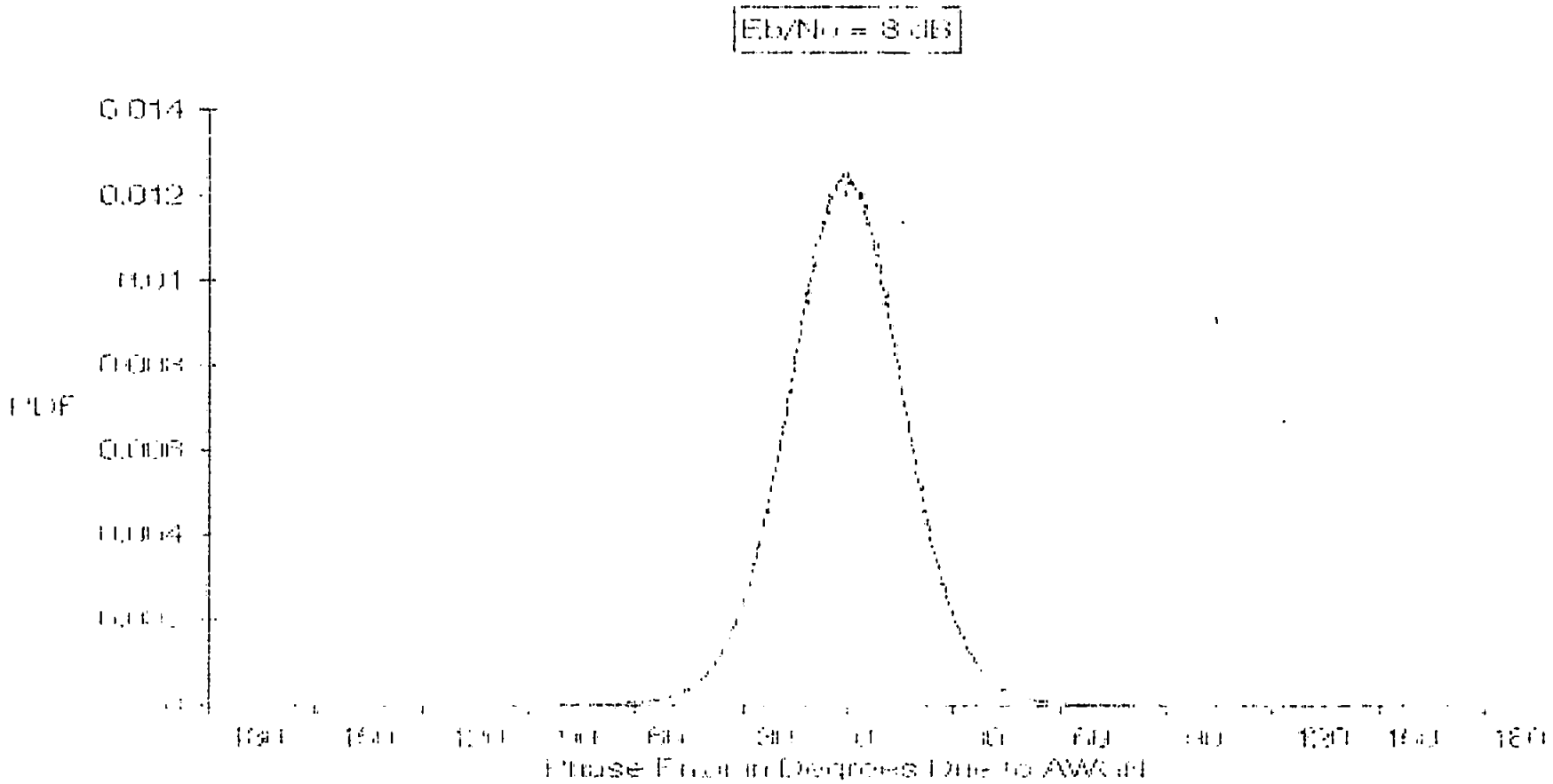


Fig 4.9.4.2.1.a Example of Plotted Phase Deviation due to AWGN

5. Results

The results will be documented in two major sections:-

a) The computer simulation results and predictions.
b) The experimental results.

These two main sections will then be broken down into sub-sections governing the various attributes of the implementation performance.

5.1. Computer Simulation of Carrier Estimation

5.1.1. Noise Statistics

The accuracy of the phase noise statistics was ascertained by the calculation of Probabilities of Error for a BPSK signal and compared to theoretical. For a BPSK signal to be in error the phase noise must be greater than 90 degrees. The probability of the phase being greater than 90 degrees is therefore the summation of the area under the curve from 90 to 180 degrees and -90 to -180 degrees. By comparison of this result with theoretical it is possible to evaluate the accuracy of the simulated noise distribution. It is therefore possible to choose the required number of samples that have to be calculated to produce an accurate phase distribution. The number of samples used in the calculation is governed by the noise step size. The evaluation of the noise step size can be seen in Table 5.1.1.

Histogram cell size = 0.5 degrees			
BPSK at Eb/No = 8 dB	Probability of Error* = 1.909072×10^{-4}		
Noise step size	No. of Samples	P(ø>90)	% error
1σ	121	8.133E-5	57
0.5σ	441	6.056E-5	68
0.3σ	1179	2.351E-4	-23
0.1σ	10201	1.836E-4	3.8
0.05σ	40401	1.9306E-4	-1.1
0.03σ	111779	1.8648E-4	2.3
BPSK at Eb/No = 9dB	Probability of Error* = 3.362664×10^{-5}		
0.05σ	40401	3.41817E-5	-1.6
0.03σ	111779	3.44983E-5	-2.6
BPSK at Eb/No = 3 dB	Probability of Error* = 2.287841×10^{-2}		
0.05σ	40401	2.36409E-2	-3.3
0.03σ	111779	2.27417E-2	0.6
0.02σ	251001	2.31075E-2	1.0
BPSK at Eb/No = 5 dB	Probability of Error* = 5.953867×10^{-3}		
0.05σ	40401	5.66557E-3	4.8
0.03σ	111779	6.04744E-3	-1.5
BPSK at Eb/No = 6dB	Probability of Error* = 2.38829×10^{-3}		
0.05σ	40401	2.31450E-3	3.1
0.03σ	111779	2.41674E-3	-1.1
*Reference :-Lindsey and Simon.			

Table 5.1.1 Comparison of Accuracy of Noise Statistics

From the above calculation it was decided that a noise step size of 0.03σ gave an accurate answer with an error on average of 0.5%. The time taken to calculate the 11779 noise sample using the IBM-AT at 6 MHZ Clocking and with a Mathematical Co-processor was approximately 20 minutes per Eb/No point. To increase the number of sample points was considered unnecessary when comparing the trade off between accuracy and processor time. A selection of phase noise statistics obtained by the simulation can be seen in fig 5.1.1.a and it can be seen that the phase noise distribution is a bi-variate gaussian distribution . If we consider various SNR then the distributions can be seen to agree with those results obtained by Prokais <13> by the evaluation of the joint probability density function.

5.1.2. Performance of Phase Estimator with AWGN, Frequency offset= 0 HZ.

The simulation was designed to enable a number of variables that an operating modem would encounter to be set up on initialization. ie.data rates, signal type, (BPSK, QPSK), Signal to noise ratios, frequency offsets, phase and frequency filter coefficient, number of bits in the filter implementation etc. This results in a very large number of combinations to test the simulation under all conditions and would imply a considerable time being spent. This time can be considerably reduced by limiting the number of variables and performing initial tests to obtain an understanding of the appropriate areas of interest. This can result in a decision directed test plan and hence reduce the number of variables to a manageable level. The initial tests will therefore consist of simulations of BPSK and QPSK with zero frequency error over a range of SNR's with varying coefficients on the phase and frequency filters. The data rate is fixed at 256Ksymbols/sec and all other data rate

performances can be scaled as appropriate. The number of bits in the filter implementations is set at 16. This will enable an optimum choice of phase and frequency coefficients to be selected. The key to the figures indicates the value of phase and frequency coefficients used, ie. p15f127 indicates a phase coefficient of 15/16 and a frequency coefficient of 127/128. A BPSK signal will first be considered.

5.1.2.1. BPSK Performance for Varying Coefficients

Initial simulation conditions:-

i) Type = BPSK
ii) Symbol rate = 256Ksymbols/sec
iii) Eb/No 3dB to 7 dB
iv) Number of filter feedback bits = 16
v) Frequency offset = 0 Hz

Consider fig 5.1.2.1.a and fig 5.1.2.1.b. It can be seen that the performance predicted by the simulation is close to theoretical and in some instants is better than theoretical! This cannot be the case but it is within a factor of 2 in BER,, which is generally considered a reasonable uncertainty <18>. Expressed as an error from theoretical in dB's gives a worst case indication of approximately 0.5 dBs. The simulation time of this simulation needs to be increased to obtain a more accurate answer if required. As this is only an experiment to ascertain the general operation of the phase estimator then this accuracy to within 0.5 dB is considered satisfactory and further accuracy is not required. Practically, measurement within 0.5 dB at low signal to noise ratios is difficult and requires very accurate measurement and long time averaging. One hypothesis that can be inferred from these performance figures is that the estimator will operate close to theory and at Eb/No's of 3dB's and

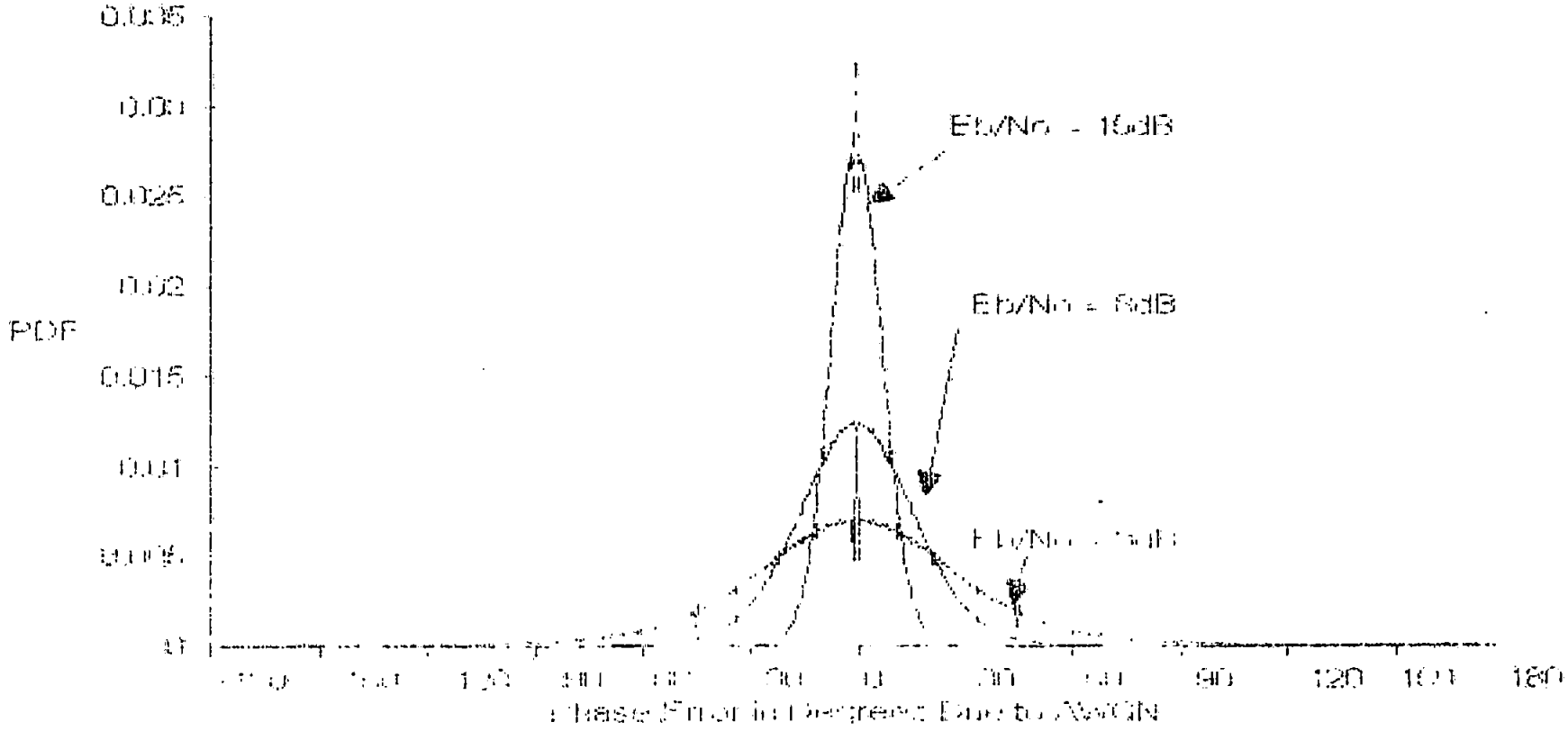


Fig 5.1.1.a Simulated Phase Deviations PDF produced by AWGN Source

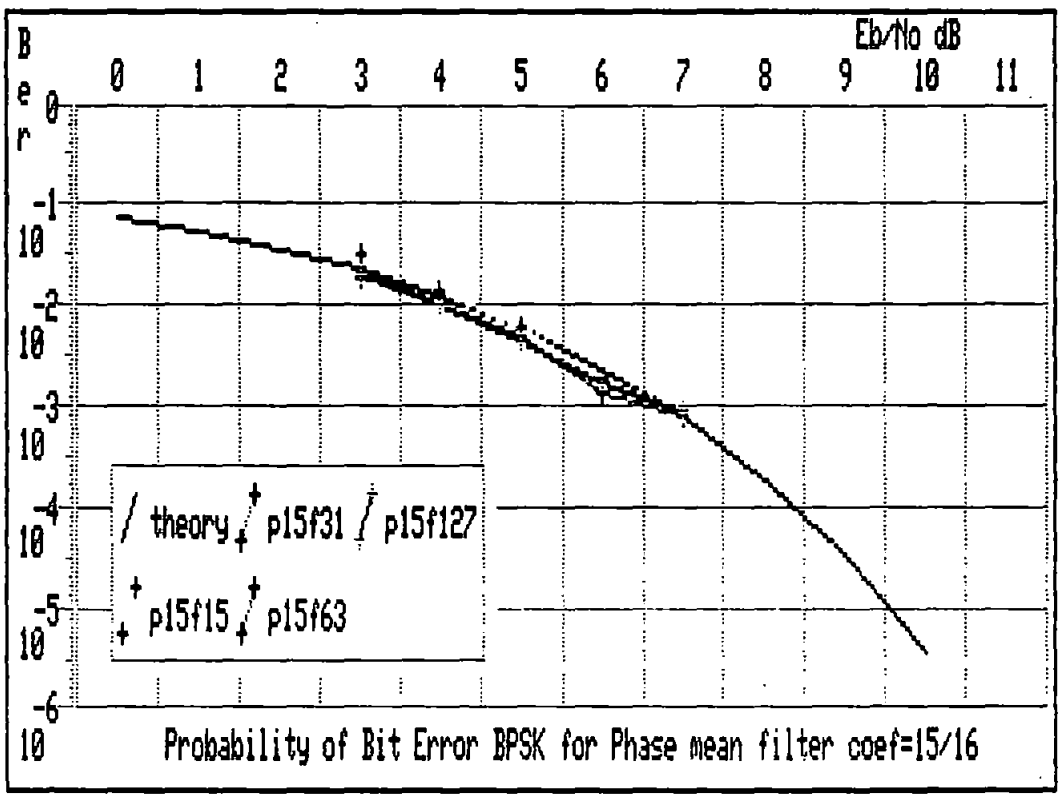


Fig 5.1.2.1.a Simulated BPSK Phase Estimation Performance for $f_{offset} = 0$ Hz

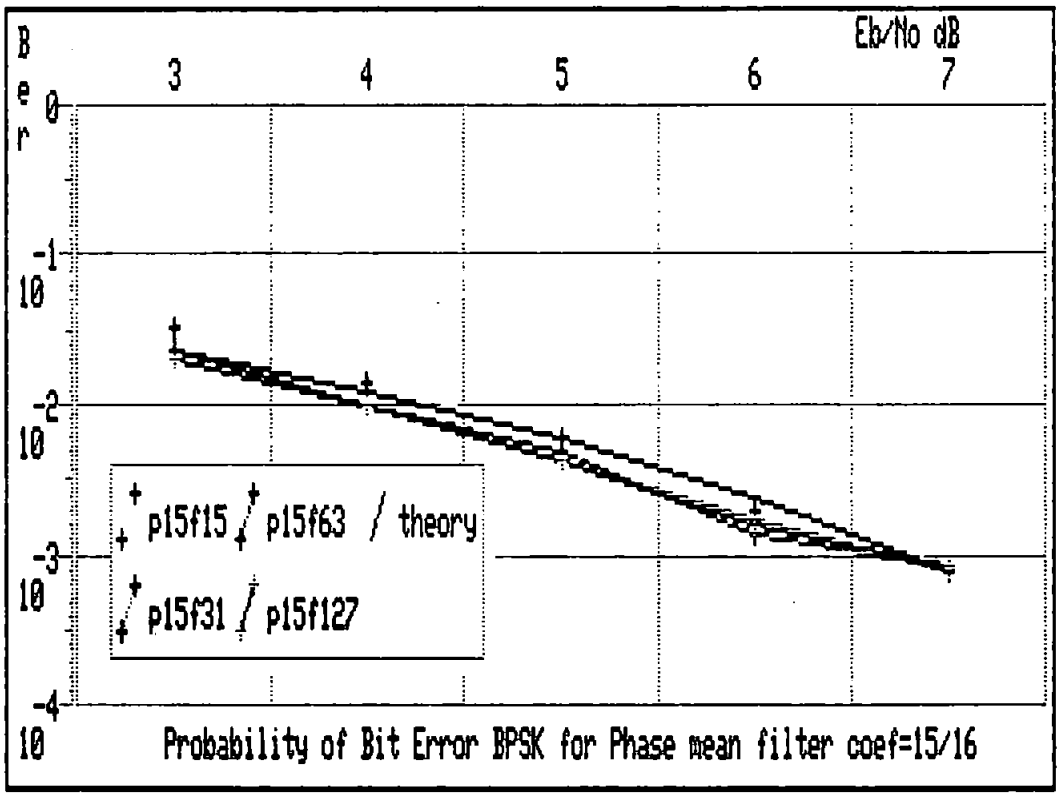


Fig 5.1.2.1.b Simulated BPSK Phase Estimation Performance for $f_{offset} = 0$ Hz

greater. We cannot draw any conclusions as to the required phase and frequency coefficients.

If we now consider fig 5.1.2.1.c and fig 5.1.2.1.d more can be inferred as the the nature of the coefficient requirements. If we consider p31f15 then the frequency estimate is filtered less than the phase estimate and as such is less accurate. Since this frequency estimate is used to project the mean phase estimate then an inaccuracy in this calculation will increase the likelihood of an error in the phase estimate $\hat{\Phi}$ (t) and hence the performance is degraded by approximately 1.5dB. As the frequency estimate coefficient is increased then the degradation becomes less until again the curve lies close to that of theoretical. This implies that the greater the accuracy of the frequency estimate the better the performance of the Phase estimator. If we now consider fig 5.1.2.1.e the phase coefficient has been increased to 63/64 and again lower values of frequency coefficient produce a degradation in performance. It must also be remembered that as the phase mean filter coefficient increases, the delay time around the feedback loop also increases and hence the projection increases. This results in any small errors in the frequency estimate being magnified and degrading the performance. Considering fig 5.1.2.1.f the phase mean coefficient is now 127/128 and even with a large frequency filter coefficient the degradation is considerable. The conclusions that must be drawn from these simulation results are :-

a) The phase mean filter coefficient must be kept low to reduce the projection time.
b) The phase mean filter coefficient must not be too low otherwise the noise performance is impaired.

c) The frequency filter coefficient must be close enough to unity such that an accurate frequency estimate is obtained. When projecting the phase this estimate must not degrade the overall performance. However if the frequency filter coefficient is close to unity then the acquisition time of the phase estimator ie. the time taken for the frequency estimate to be valid , increases. It may therefore be necessary to trade acquisition time for estimator performance.

d) Near unity frequency filter coefficients may also reduce the frequency following ability of any rapid short term frequency variations.

From these conclusions the best choice of phase mean filter coefficient is 15/16 with a frequency filter coefficient of 127/128. This performance can be seen in Fig 5.1.2.1.b.

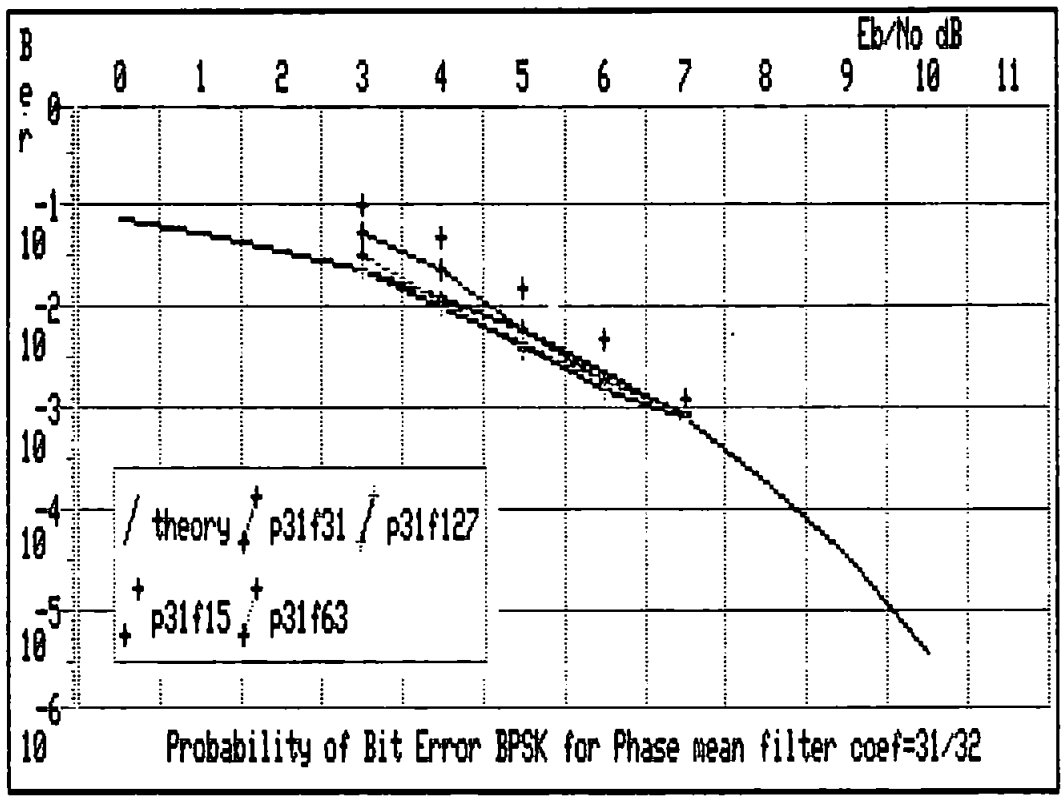


Fig 5.1.2.1.c Simulated BPSK Phase Estimator Performance for $f_{offset} = 0$ Hz

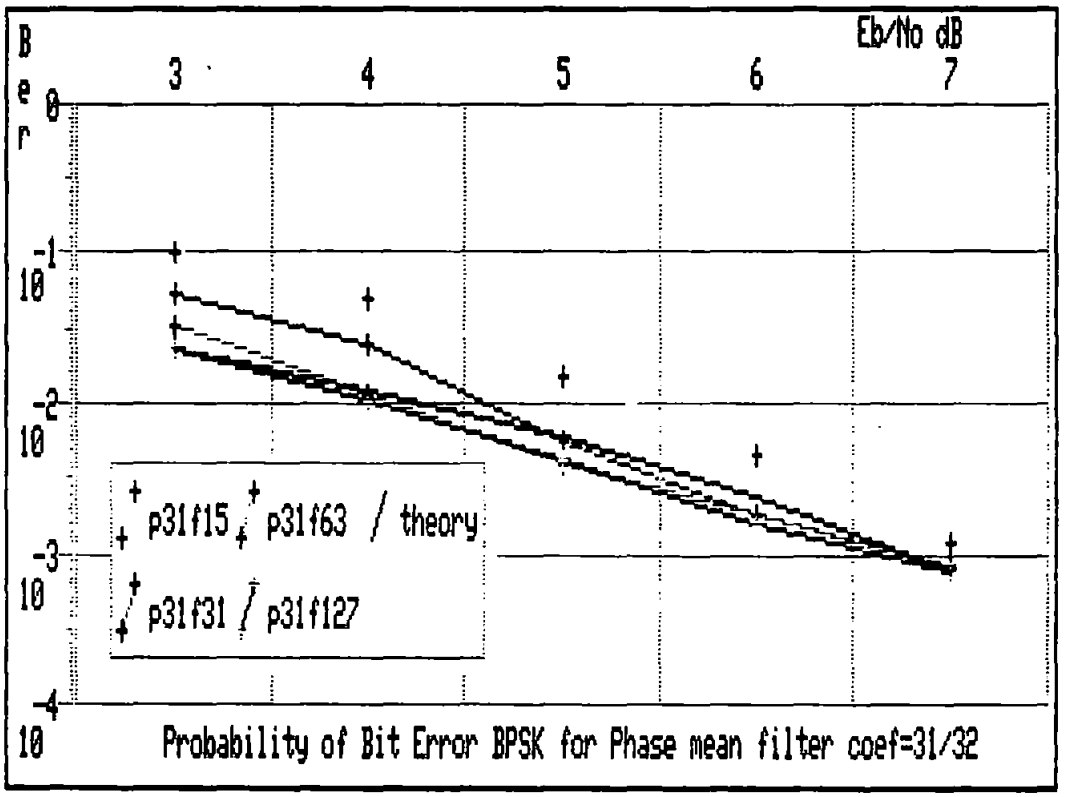


Fig 5.1.2.1.d Simulated BPSK Phase Estimator Performance for $f_{offset} = 0$ Hz

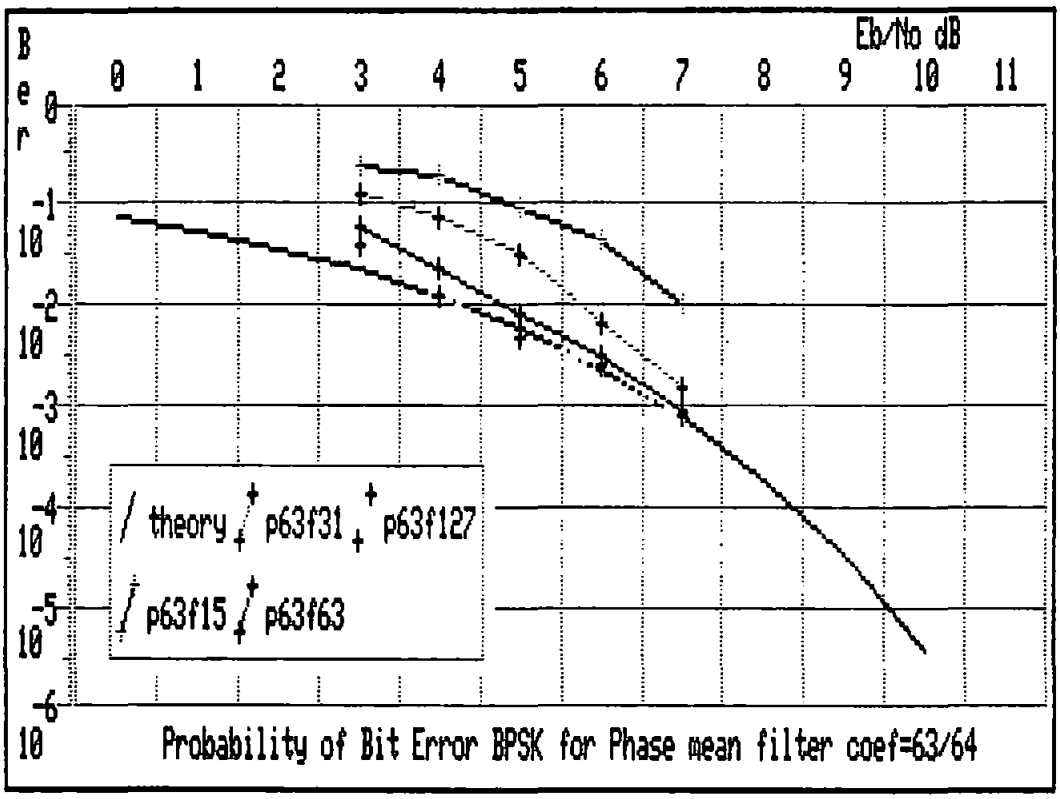


Fig 5.1.2.1.e Simulated BPSK Phase Estimator Performance for $f_{offset} = 0$ Hz

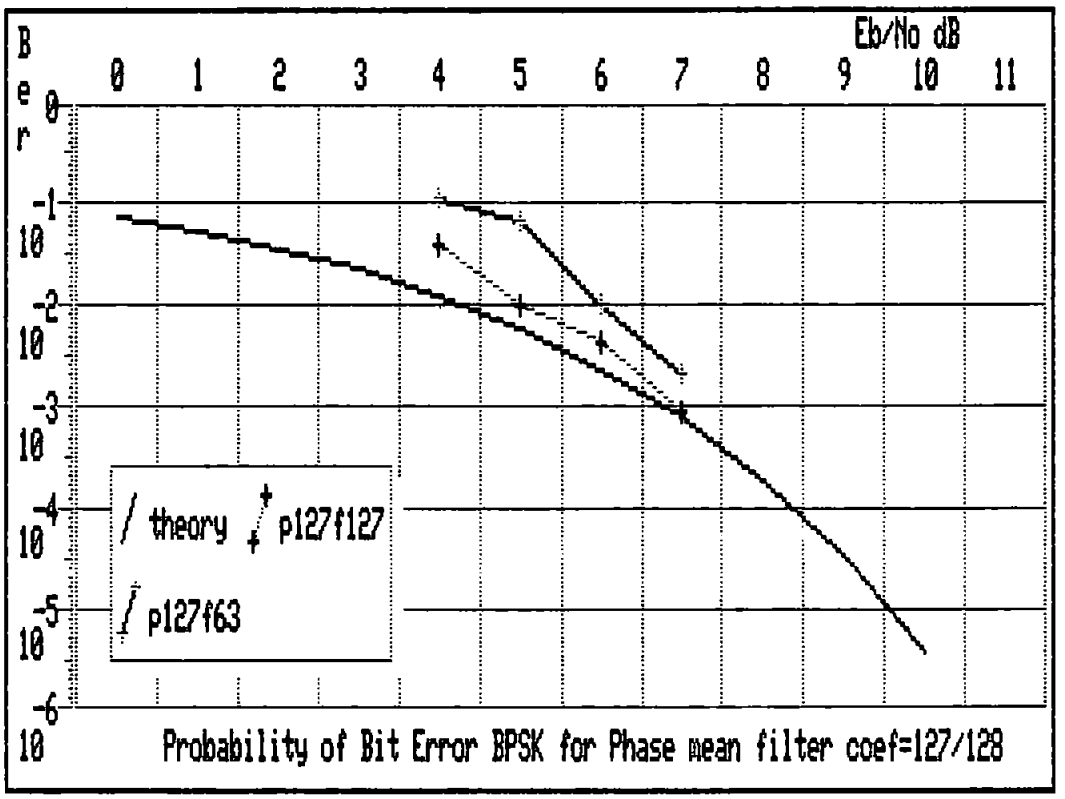


Fig 5.1.2.1.f Simulated BPSK Phase Estimator Performance for $f_{offset} = 0$ Hz

5.1.2.2. QPSK Performance for Varying Coefficients

Initial simulation conditions:-

i) Type = QPSK

ii) Symbol rate = 256Ksymbols/sec

iii) Eb/No 3dB to 7 dB

iv) Number of filter feedback bits = 16

v) Frequency offset = 0 Hz

If we consider figures 5.1.2.2.a through to 5.1.2.2.f then it can clearly be seen that the same arguments apply for the QPSK signal as for the BPSK signal. The phase estimator operates from an Eb/No of 3dB's or greater with the optimum performance coming from a small phase mean filter coefficient of 15/16 and a frequency filter coefficient of 127/128.

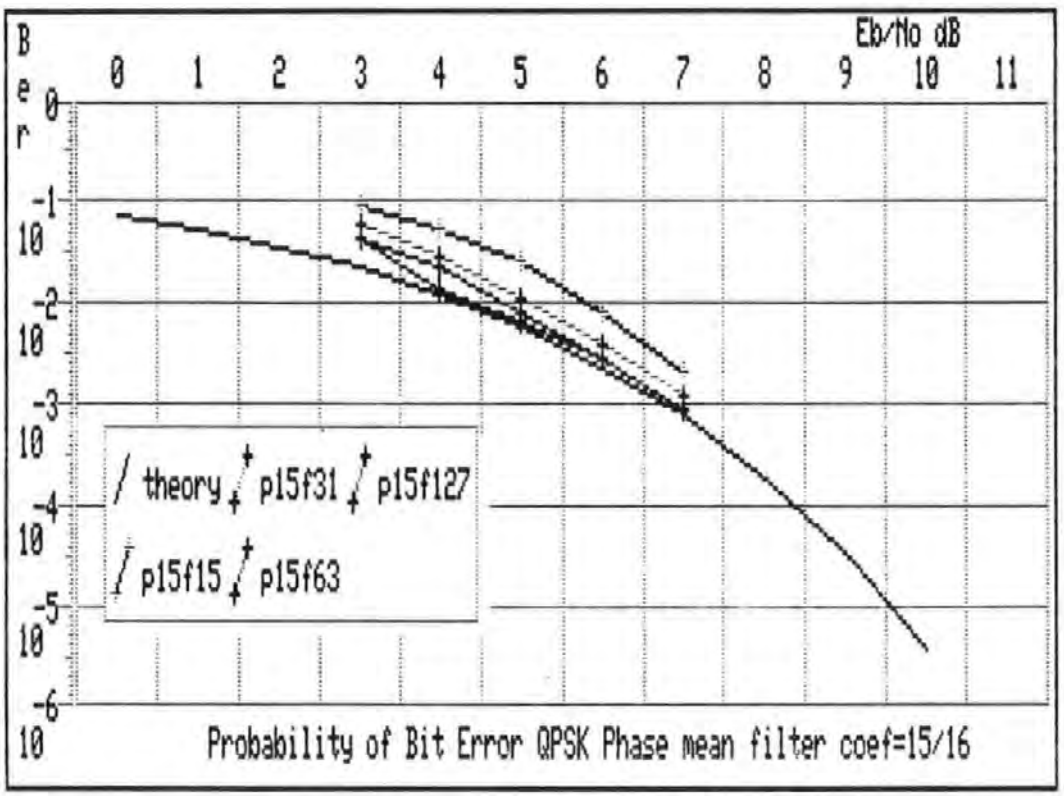


Fig 5.1.2.2.a Simulated QPSK Phase Estimator Performance for $f_{offset} = 0$ Hz

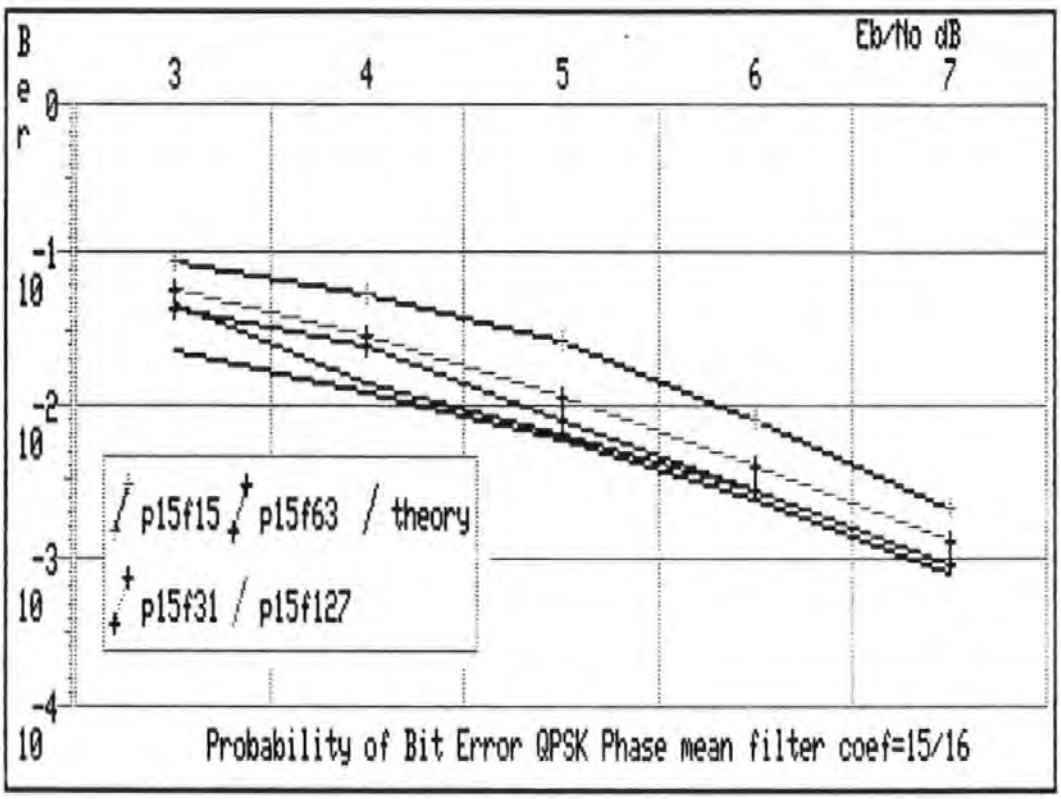


Fig 5.1.2.2.b Simulated QPSK Phase Estimator Performance for $f_{offset} = 0$ Hz

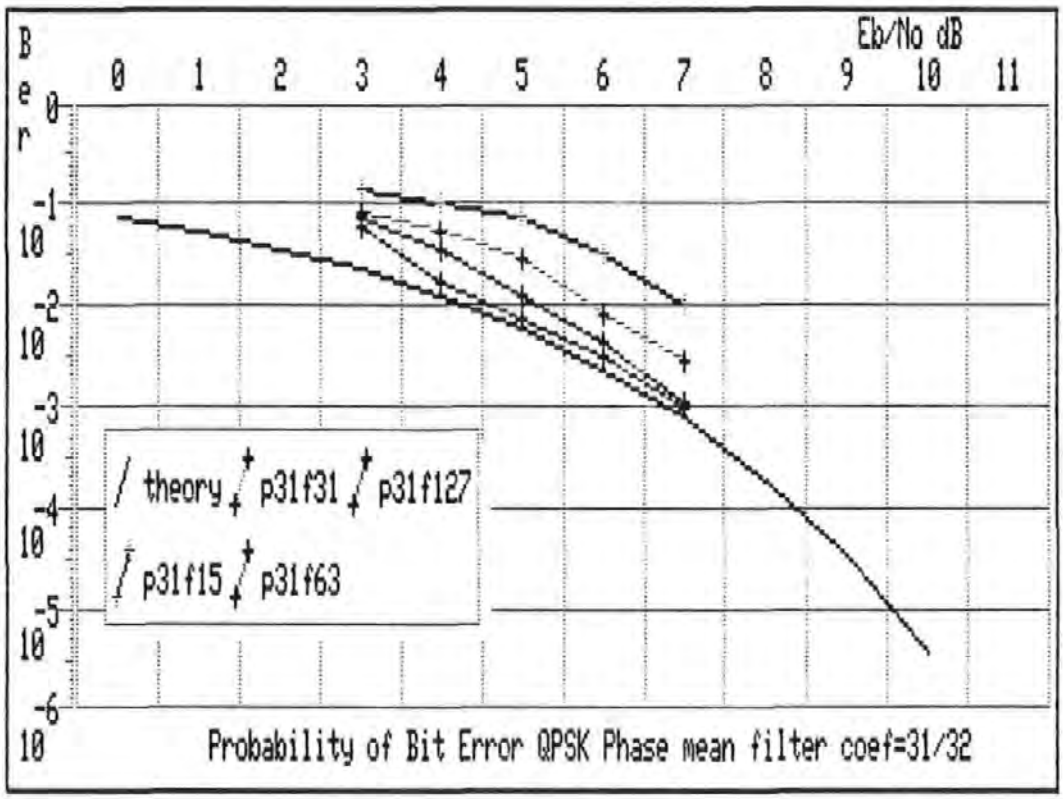


Fig 5.1.2.2.c Simulated QPSK Phase Estimator Performance for $f_{offset} = 0$ Hz

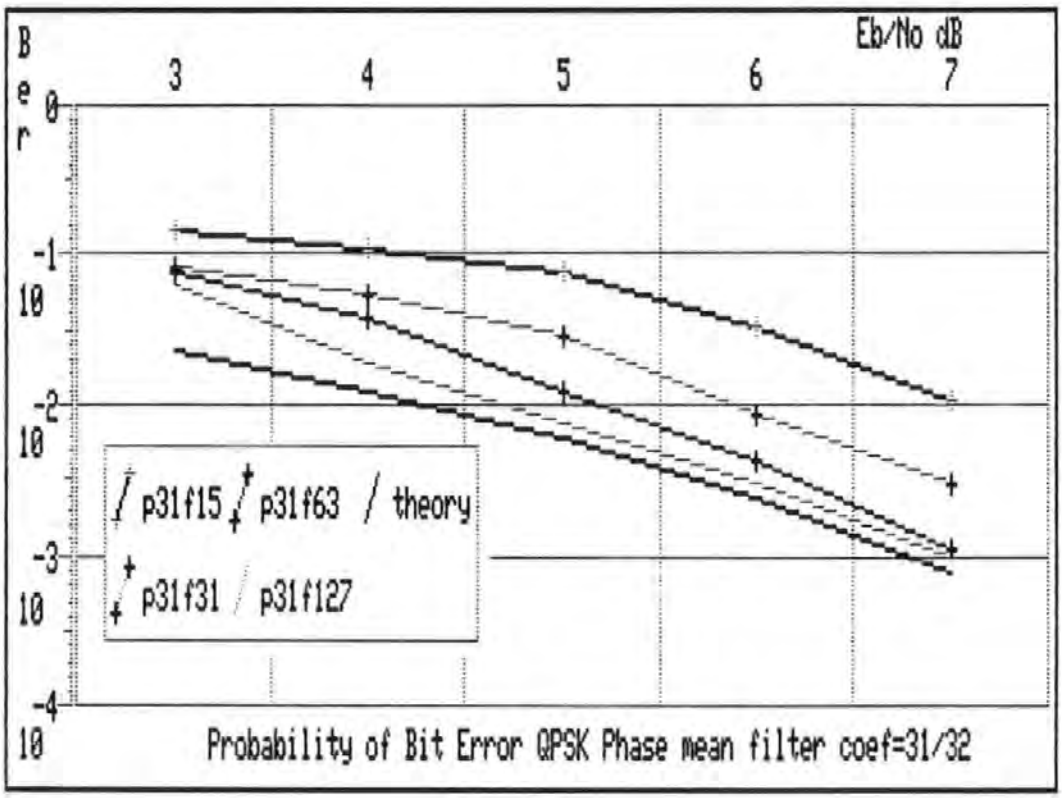


Fig 5.1.2.2.d Simulated QPSK Phase Estimator Performance for $f_{offset} = 0$ Hz

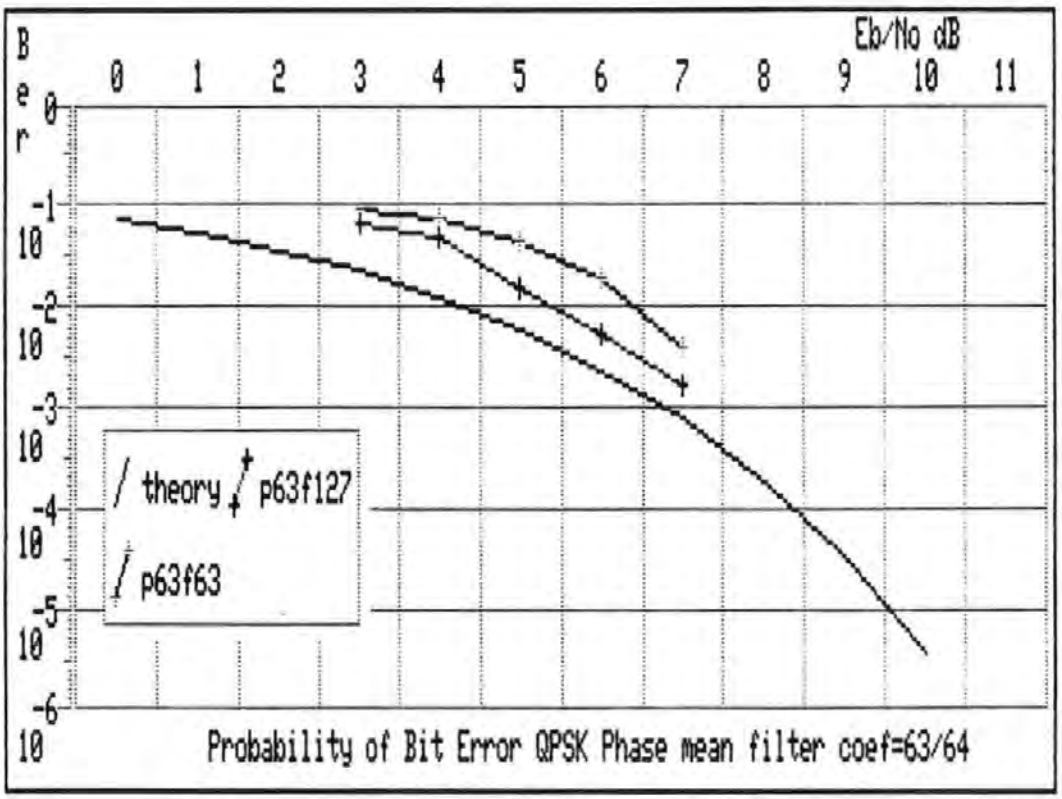


Fig 5.1.2.2.e Simulated QPSK Phase Estimator Performance for $f_{offset} = 0$ Hz

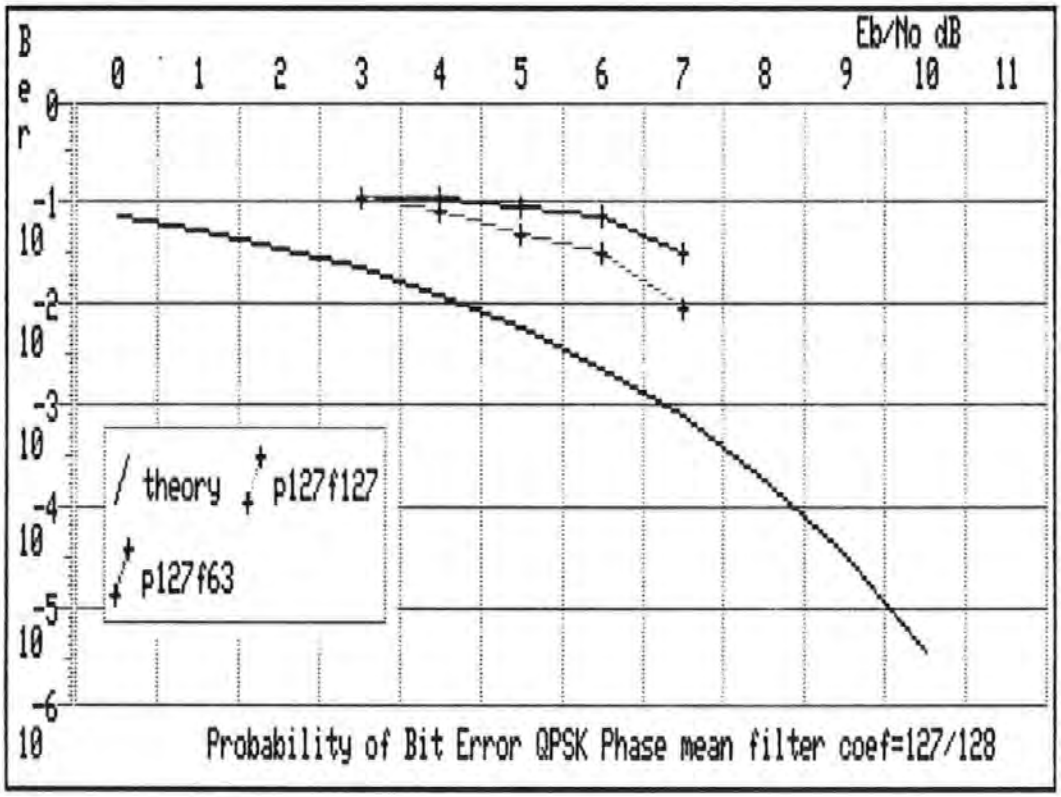


Fig 5.1.2.2.f Simulated QPSK Phase Estimator Performance for $f_{offset} = 0$ Hz

5.1.3. Performance Against Noise with a Frequency Offset.

Initial simulation conditions:-

i) Type = BPSK and QPSK

ii) Symbol rate = 256Ksymbols/sec

iii) Eb/No 3dB to 7 dB

iv) Number of filter feedback bits = 16

v) Frequency offset = 1000 Hz, 2000 Hz, 3000 Hz

vi) Phase Mean filter coefficient 15/16

vii) Frequency filter = 127/128

If we consider fig 5.1.3.a the results for the BPSK simulation performance can be seen. In this figure the performance degradation is minimal even with a 3 Khz frequency offset. If we consider the QPSK case as in fig 5.1.3.b then it can be seen that as the frequency offset increases then the degradation in performance becomes more pronounced. This can be explained by an understanding of the data removal process in both the phase mean and frequency estimation algorithms. To remove the data from the rotating phasor the phase is multiplied by 2 mod 360 for BPSK and 4 mod 360 for QPSK. This results in a folding of the phasor diagram into +/- 90 degrees and +/- 45 degrees respectively. It can easily now be seen that in the QPSK format an instantaneous phase change from nominal of only +/- 45 degree would result in an error, whereas for BPSK an instantaneous change of +/- 90 degrees is required. With the change in format the accuracy of the projecting calculation also effects the overall performance as errors in the estimated phase can be considered to be noise like and hence add to the phase deviation produced by the AWGN in a powerwise manner.The effect of the multiplication process is therefore to enhance the overall phase variance of the processed phasor. Hence as the frequency offset becomes greater and the loop becomes more stressed, then the probability of this error occurring for a QPSK signal arises at a higher Eb/No than for a BPSK signal.

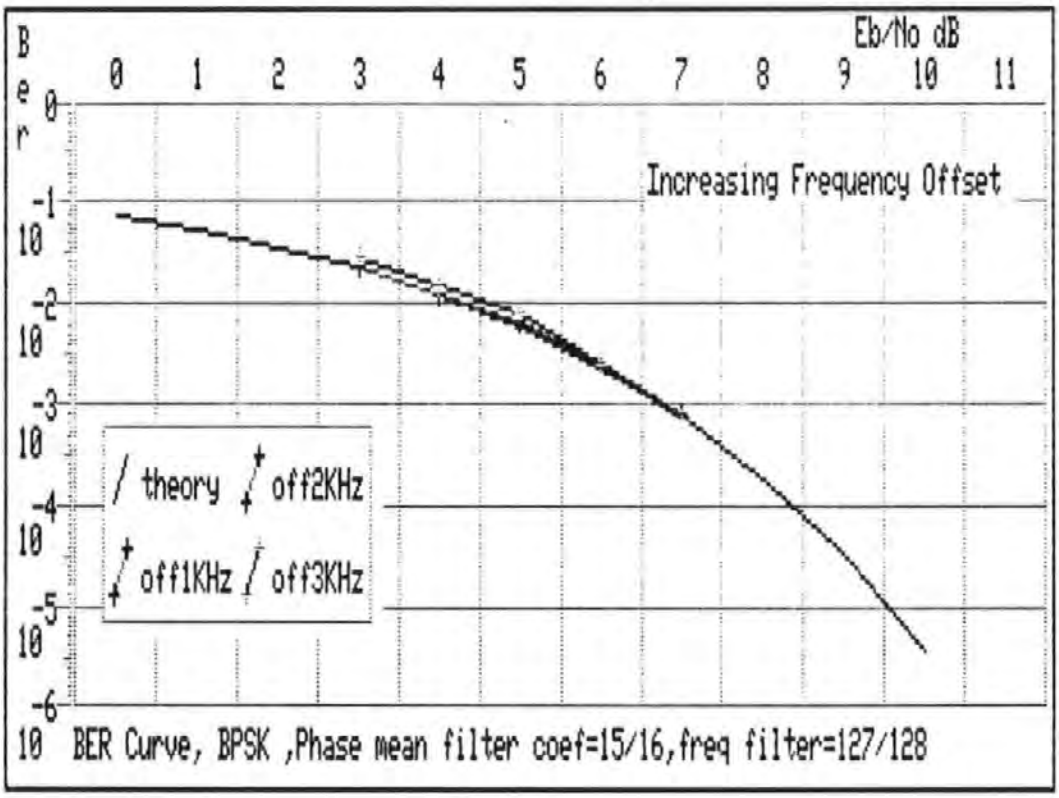


Fig 5.1.3.a Simulated BPSK Phase Estimator Performance for f_{offset} = 1000, 2000, 3000 Hz

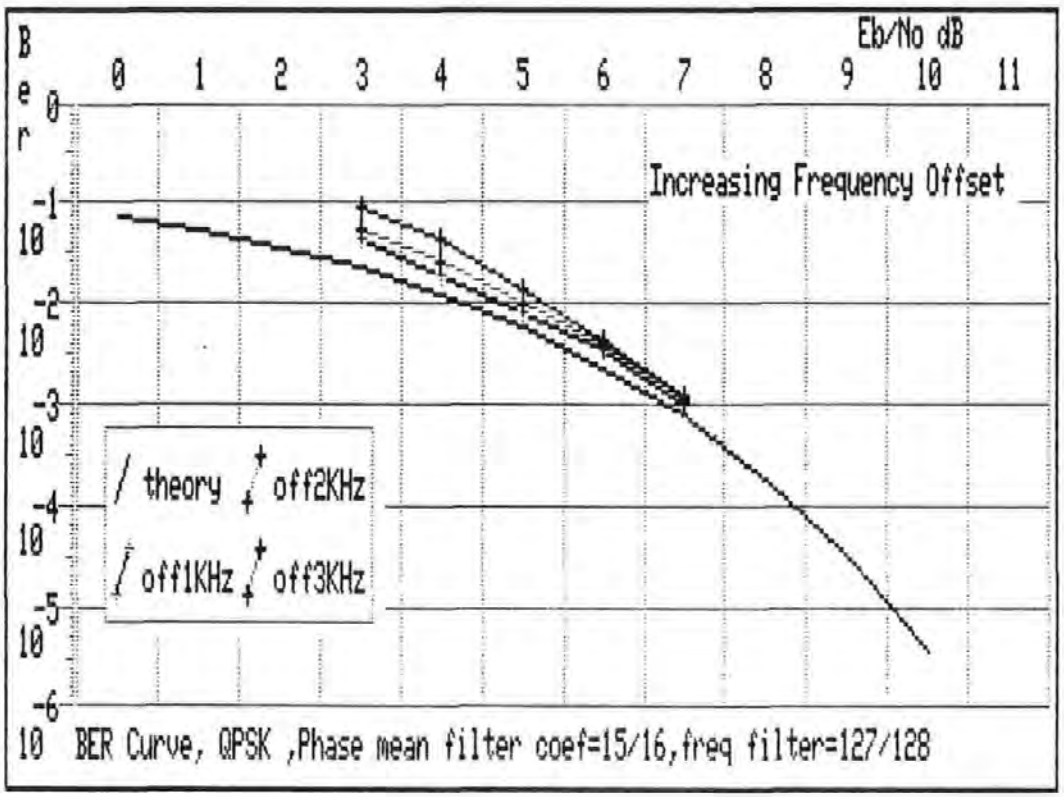


Fig 5.1.3.b Simulated QPSK Phase Estimator Performance for f_{offset} = 1000, 2000, 3000 Hz

5.1.4. Cycle Slip Probability

If we examine the predicted phase value $\hat{\Phi}$ (t) obtained from the simulation, for varying values of Eb/no, we can see that the rms phase error deviation for the stimulated signal has undergone a signal to noise improvement in relation to the received signal. However, a cycle slip will occur if the predicted phase crosses a decision boundary which in the case of QPSK is +/- 45 degrees and +/- 90 degrees for BPSK. The received signal incorporates a phase error deviation produced by the AWG noise (σ) for a given Es/No. From Eq4.4.1.b it can be seen that the phase estimate has equal output phase deviation for both BPSK and QPSK for a given set of coefficients and input noise phase deviation. However since cycle slips occur for an estimator error of greater than 45 degrees in QPSK but 90 degrees in BPSK, then for a given input phase deviation, set by the input Es/No value, the probability of a cycle slip is less for BPSK then for QPSK ie. BPSK operates to a Es/No 3 dB lower than QPSK before cycle slips will occur. By analysis of the plots produced from the simulation it can be seen that the bi-variate distribution fits that produced by the the predicted phase. By using the bi-variate distribution simulated by the Phase Noise simulation program (Noise1-1) and the simulated estimated phase sigma value from the Estimator simulation program (Est1-18) an evaluation of the probability of the phase estimate crossing a decision boundary, the probability of a cycle slip, can be calculated. A table of AWGN Phase deviation can be seen in Table 5.1.4.a for variation in Eb/No. This is then used in the evaluation of cycle slip probability for various frequency offsets from nominal, which increases the stress in the loop, and varying Eb/No, the results of which can be seen in Tables 5.1.4.b for BPSK and Table 5.1.4.c for QPSK.

Es/No dB	sigma degrees	prob>45	prob>90
3	33.6	1.50e-1	2.30e-2
4	29.0	1.08e-1	1.26e-2
5	26.0	7.22e-2	6.00e-3
6	22.8	4.44e-2	2.29e-3
7	19.4	2.55e-2	7.65e-4
8	17.0	1.2e-2	1.86e-4
9	14.9	4.96e-3	3.45e-5
10	13.2	1.6e-3	3.38e-6
11	11.7	3.66e-4	<1e-8
12	10.3	6.34e-5	<1e-8
13	9.2	7.10e-6	<1e-8
14	8.2	2.56e-7	<1e-8
15	7.3	<1e-8	<1e-8
16	6.5	<1e-8	<1e-8
17	5.7	<1e-8	<1e-8
18	5.1	<1e-8	<1e-8
19	4.6	<1e-8	<1e-8

Table 5.1.4.a Phase Deviation (1σ) produced by AWGN at Varying Eb/No.

Eb/No dB	F_{offset} KHz	Estimator Phase dev.	Probability of Cycle Slip/symbol
3	3	15.3	3.4e-5
4	3	10.6	<1e-8
5	3	8.0	<1e-8
6	3	6.2	<1e-8
7	3	4.9	<1e-8
3	2	12.0	<3.3e-6
4	2	9.4	<1e-8
5	2	7.4	<1e-8
6	2	5.7	<1e-8
7	2	4.7	<1e-8
3	1	9.6	<1e-8
4	1	7.6	<1e-8
5	1	6.4	<1e-8
6	1	5.3	<1e-8
7	1	4.6	<1e-8

Table 5.1.4.b Cycle Slip Probability Against Eb/No for BPSK Signals

Eb/No dB	F_{offset} KHz	Estimator Phase dev.	Probability of Cycle Slip/symbol
3	3	17.8	1.2e-2
4	3	13.0	1.6e-3
5	3	7.3	<1e-8
6	3	4.8	<1e-8
7	3	3.7	<1e-8
3	2	15.0	4.9e-3
4	2	10.6	6.3e-5
5	2	6.2	<1e-8
6	2	4.6	<1e-8
7	2	3.6	<1e-8

Table 5.1.4.c Cycle Slip Probability Against Eb/No for QPSK Signals

5.1.5. Quantization Effects due to Feedback Bits in the IIR Filters.

In the Phase mean and Frequency estimate filters the number of Bits used in the feedback path of the IIR filter governs the accuracy of the filter and the minimum residual error that can occur. For example ,consider a IIR filter that is impulsed momentarily. This impulse circulates around the filter, being multiplied by the coefficient and added to the input. Since the input is now zero and the coefficient is less than unity (to remain stable) the impulse declines at a rate set by the coefficient. Due to the finite number of bits, that are used in the feedback path, a point will be reached where the coefficient multiplied by the Z^{-1} value is still equal to the Z^{-1} value i.e.:-

$$| V_q Z^{-1} |_{quant} = | k. V_q Z^{-1} |_{quant} \qquad \text{(Eq5.1.5.a)}$$

This results in the output value of the filter not being able to be reduced any further to zero and this is termed the residual error. This also results in a rounding error in the coefficient multiplier output which can be analysed as noise and is often called 'roundoff noise' <8>. We therefore need to ascertain at what level the number of bits in the filter path effect the performance of the system. This was performed for simulation conditions:-

i) Type = BPSK
ii) Symbol rate = 256Ksymbols/sec
iii) Eb/No 3dB to 7 dB
iv) Number of filter feedback bits = 12,15
v) Frequency offset = 3000 Hz
vi) Phase Mean filter coefficient 15/16
vii) Frequency filter = 127/128

The results of this simulation run can be seen in fig 5.1.5.a. It can clearly be seen that a reduction of bits from 16 bits to 15 bits has a minimal effect of the performance, with only a reduction to 12 bit causing a degradation of approximately 1 dB at an Eb/No of 3dB's. Since the implementation of anything over 8 bits requires the use of two PROMs to perform the multiplication, 16 bits are readily available for feeding the addition circuit. However to create a lookup table with a 16 bit address requires a Eprom size of 64Kbytes per prom. This is now not excessive with currently available PROMs and easily implemented, but due to lack of space on the original 256Ksymbol/sec modem the number of feedback bits was limited to 12 bits for the frequency filter and 11 bits for the phase mean.

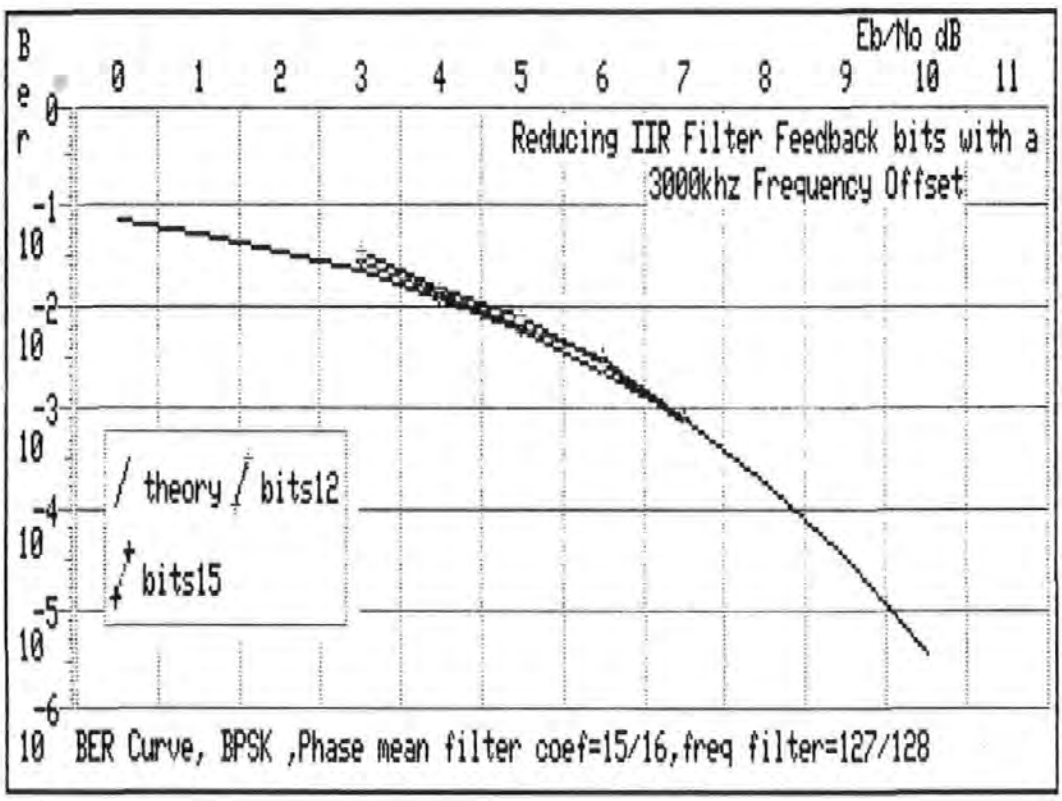


Fig 5.1.5.a Performance Variations due to Feedback Quantization in the Phase Estimator.

5.2. Experimental Performance of Modem System

The complete modem can be seen in fig 5.2.a and consists of a 19 inch, 6U Eurocard rack comprising of 11 boards.

5.2.1. Modulator

The hardware configuration for the modulator can be seen in fig 5.2.1.a. This is a single 6U eurocard 220mm in depth. The modulator takes as its input binary data, at TTL levels, and outputs a spectrally shaped quadrature signal on an IF of 1.024MHz. This is folllowed by an up conversion to 200 MHz where the converted image components are removed by a Surface Acoustic Wave (SAW) filter before converting to a standard IF of 70 MHz. It can therefore be seen that the modulator can be made very compact while still producing the required performance. The spectral shaping and up conversion to the first intermediate frequency are produced by the data stored in the PROM and can hence be easily changed. Firstly, consider the spectral shaping of a signal by a 30% raised cosine filter with a baseband output. By impulsing the modulator with single bits of data it is possible to view the impulse response of the 30% raised cosine filters. This can be seen in fig 5.2.1.b. This is shown for a 256 Ksymbol/sec filter. It can be seen that the response passes through the zero crossing at symbol period intervals. The impulse response lasts for 6 symbol periods with only a very small ripple outside the main response. This results in the frequency response as seen in fig 5.2.1.c. It can be seen that the 6dB point (i.e. half amplitude) occurs at the half symbol rate point, i.e 128 KHz, with the first frequency sidelobe approximately 30dB's down. It is these sidelobes that are caused by the truncation of the impulse response to 6 symbols. The data eye that this filter will produce can be seen in fig 5.2.1.d. .

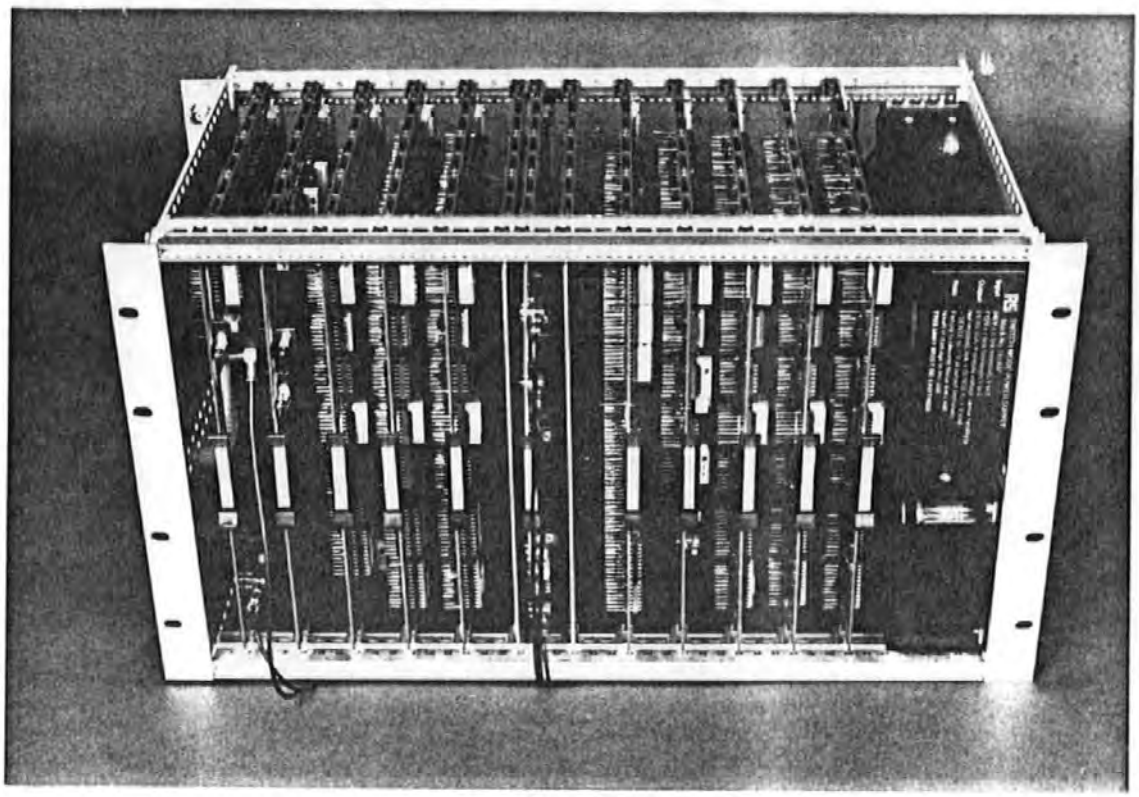


Fig 5.2.a Complete Digital Modem System

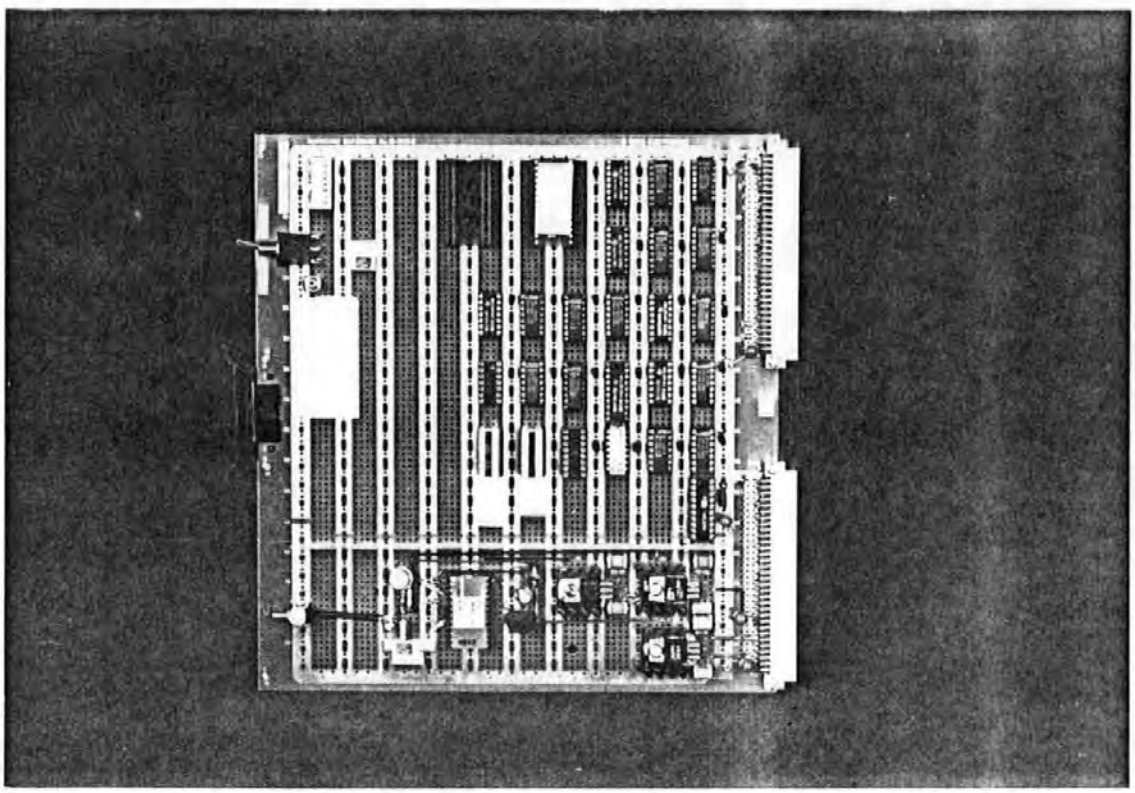


Fig 5.2.1.a Multi-Data Rate Modulator 16 Ksymbol/sec to 256 Ksymbol/sec BPSK and QPSK Formats.

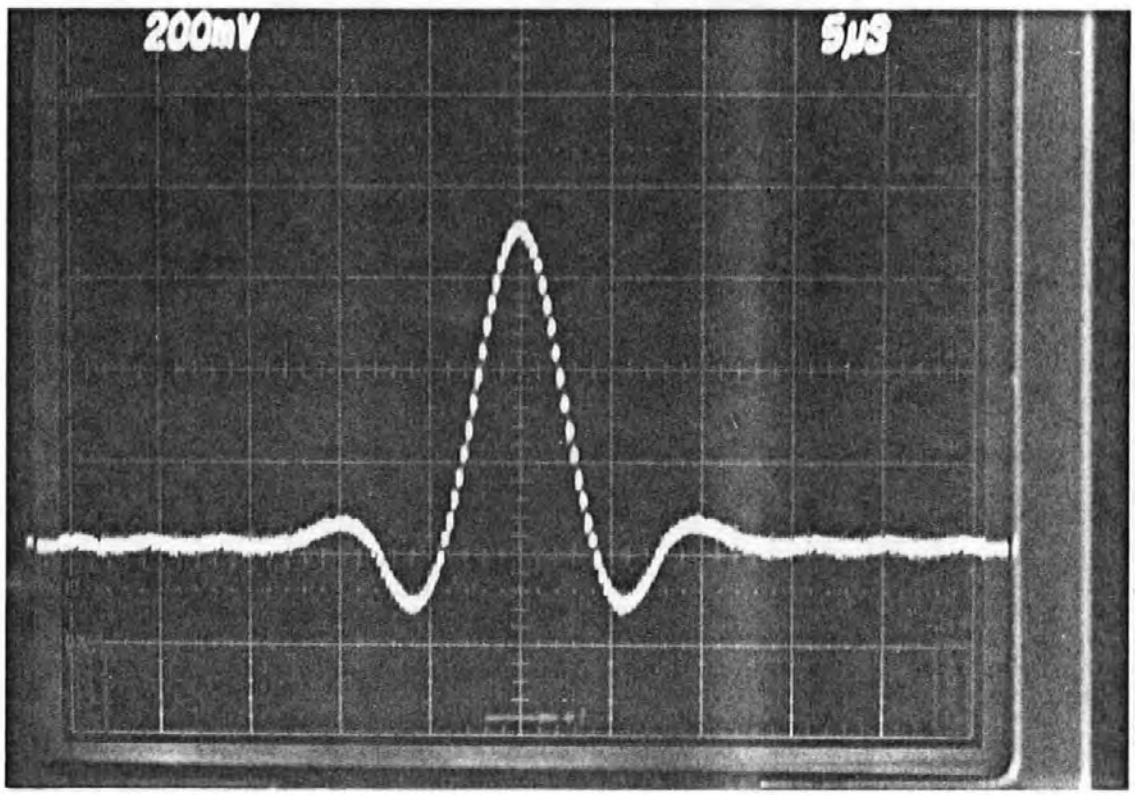


Fig 5.2.1.b 30% Raised Cosine Filter Impulse Response (no Carrier) for 256 Ksymbol/sec Filter.

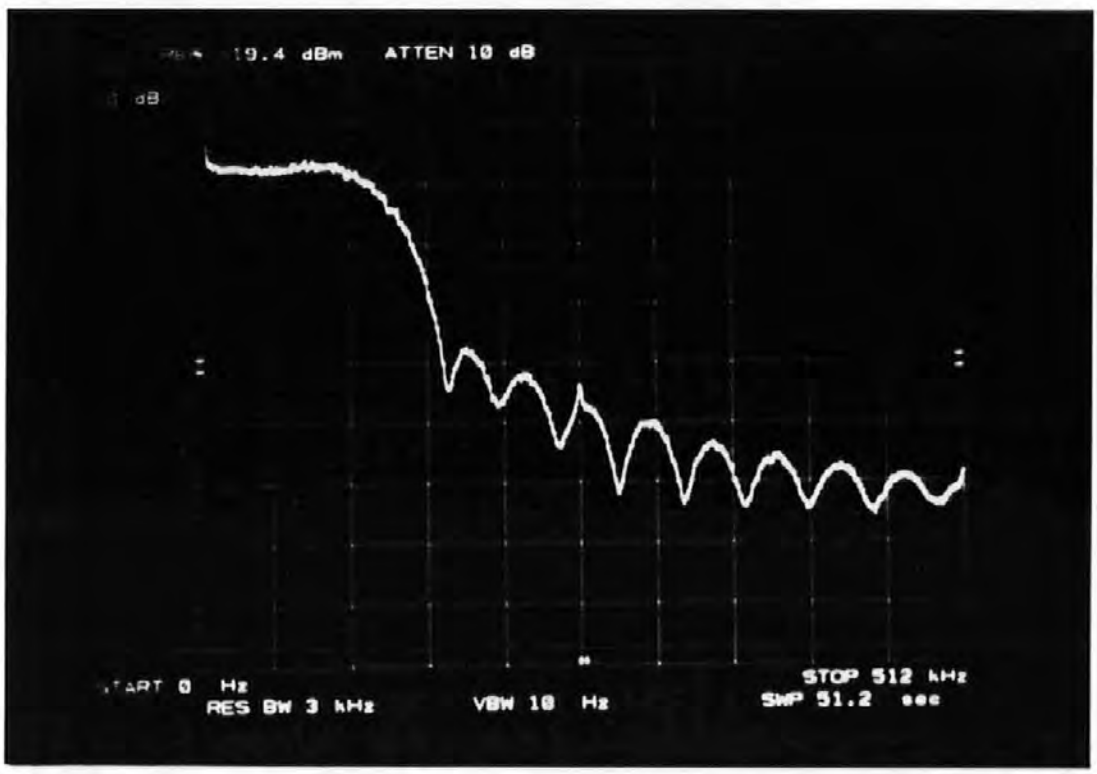


Fig 5.2.1.c 30% Raised Cosine Filter Frequency Response (no Carrier) for 256 Ksymbol/sec Filter.

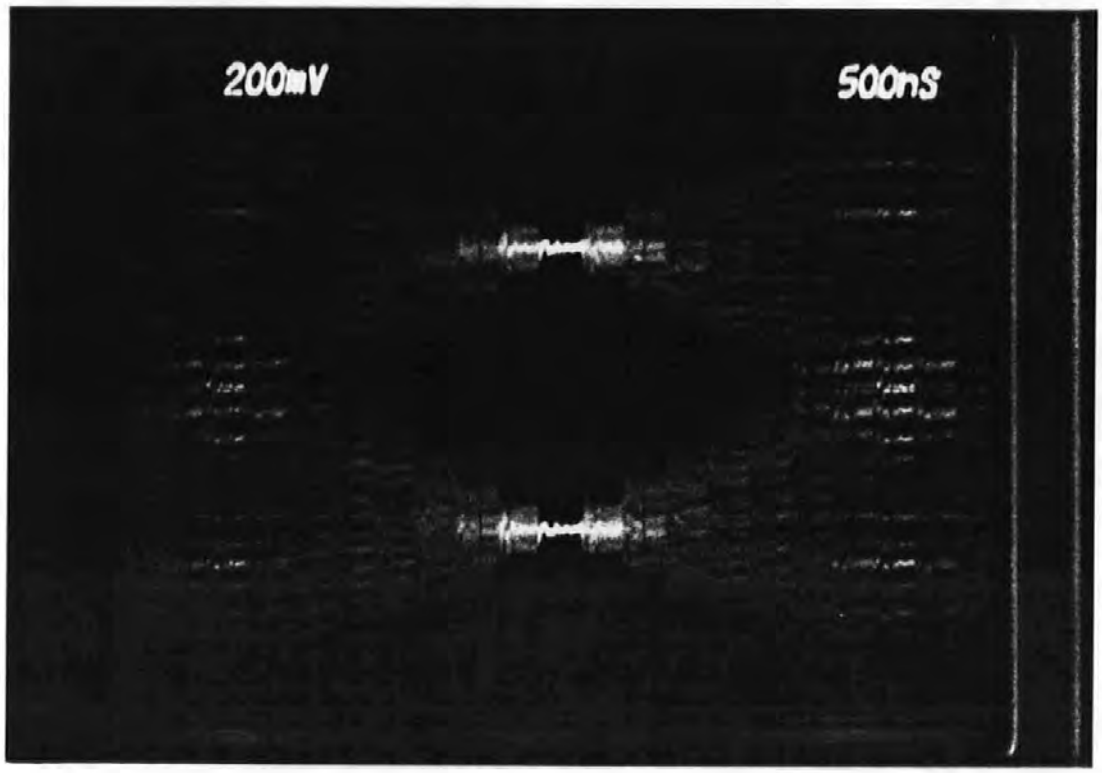


Fig 5.2.1.d Data Eye Generated by 256 Ksymbol/sec Data Stream Passing Through a Digital 30% Raised Cosine Filter

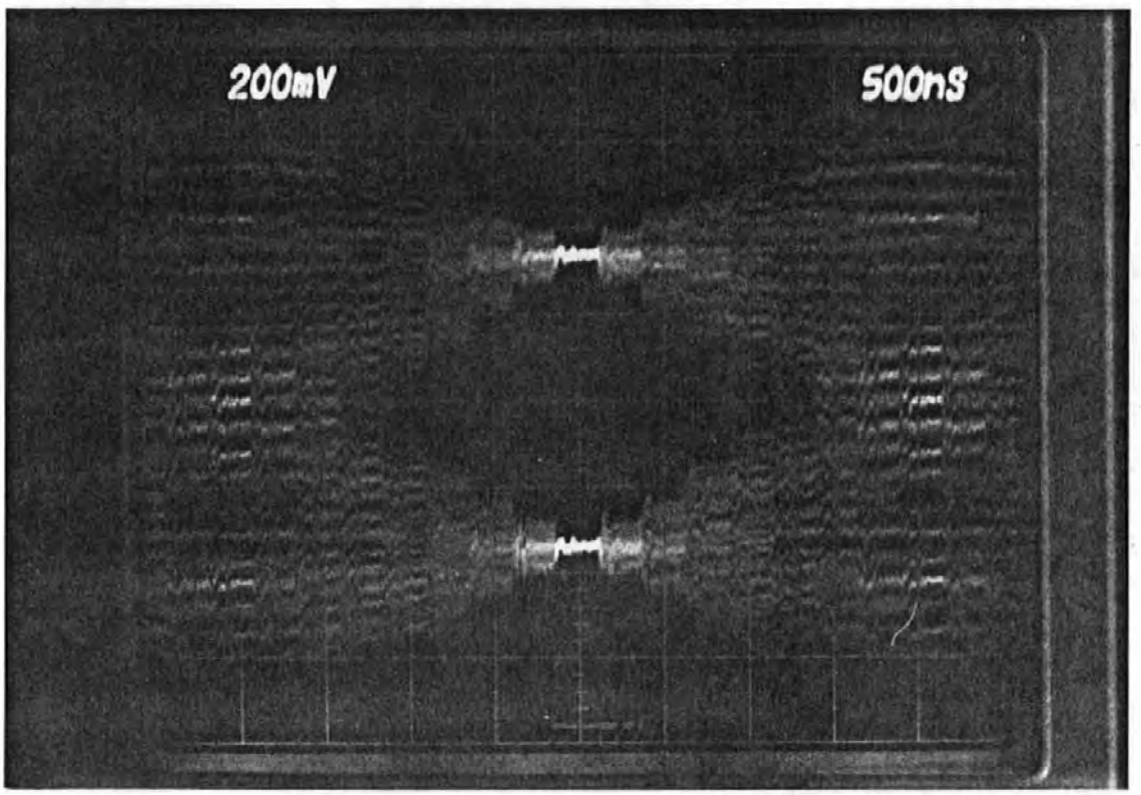


Fig 5.2.1.d Data Eye Generated by 256 Ksymbol/sec Data Stream Passing Through a Digital 30% Raised Cosine Filter

Due to the symmetrical impulse response, with the zero crossings all occurring at the symbol period rate, the data eye is fully open with all transitions passing through two points. To achieve this with an analogue technique would be virtually impossible. Now consider the case where the modulator is producing a root 30% raised cosine response and being modulated onto a 1.024MHz IF. If the modulator is impulsed the result will be a carrier component onto which will be modulated the filter impulse response. Hence if we observe the output of the modulator we should see a series of reversals superimposed with a impulse. In the BPSK format alternate samples,i.e. the quadrature channel samples, are zero and hence a zero component will be observed. This can be seen in fig 5.2.1.e for the impulse response and the data eye in fig 5.2.1.f. The data eye is no longer fully open as some transitions do not now pass through the zero crossing at the symbol periods. However, since the receiver filters will be matched to this response then the open eye can be recovered. The spectral response for BPSK can be seen in fig 5.2.1.g and this conforms to a root 30% raised cosine signal. The frequency sidelobes are approximately 30 dB down on the centre frequency with the half power point i.e. 3dB now occurring at the half symbol rate points i.e. ± 128 KHz. Considering the time and frequency response for the QPSK signal format the results are shown in fig 5.2.1.h,i,j. It can clearly be seen that the time response no longer has a zero component since the quadrature channel now conveys data. The effect of this is an increase of 3 dBs in the spectral response, due to the orthoganality of the signals, but with all other spectral features remaining the same. The spectral response can be compared to the computer plot in fig 5.2.1.k.

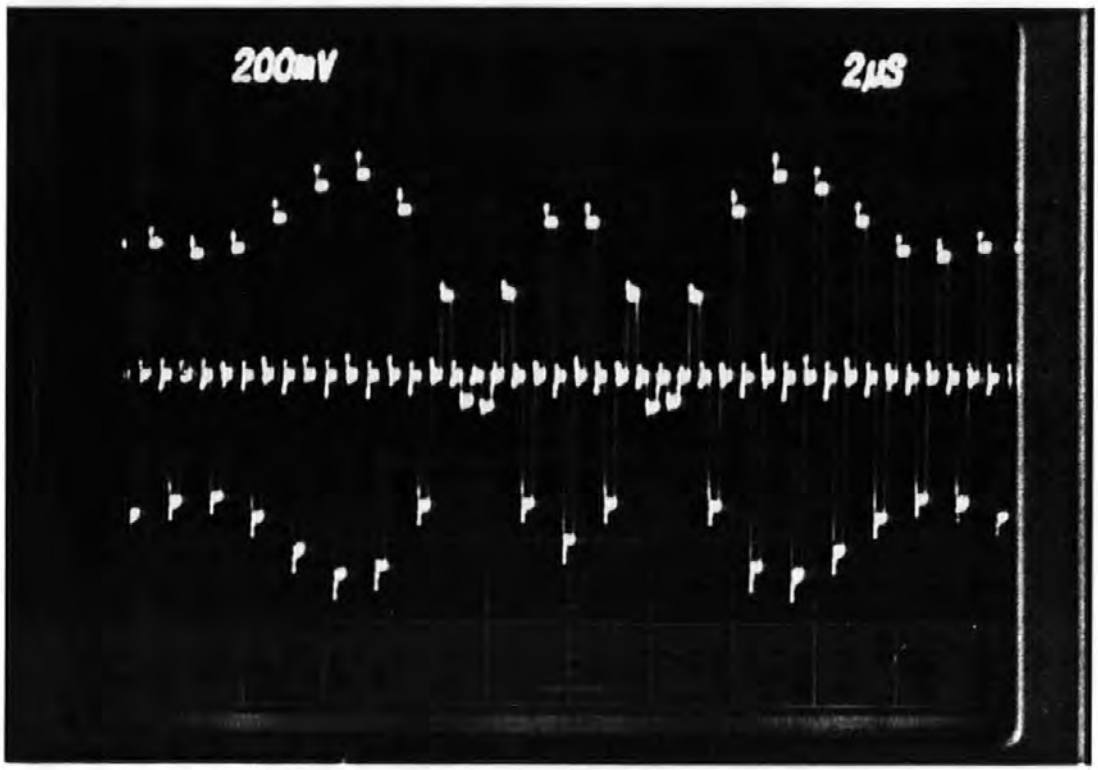


Fig 5.2.1.e BPSK Modulated Root 30% Raised Cosine Filter Impulse Response

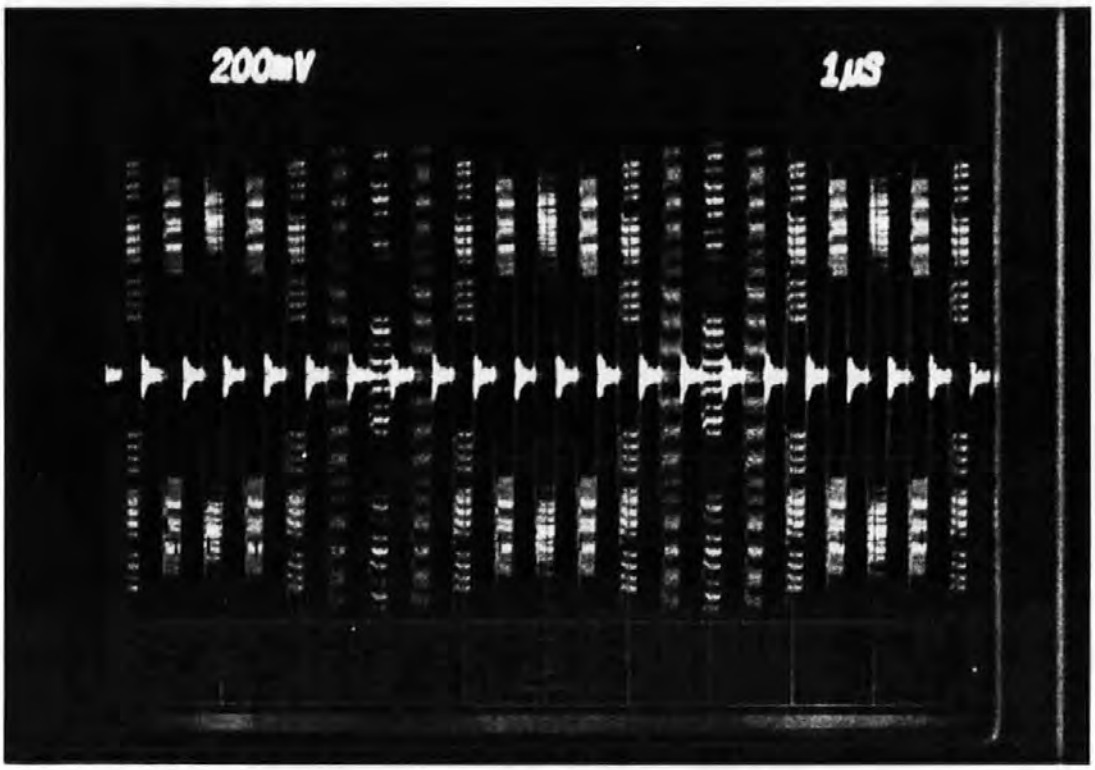


Fig 5.2.1.f Data Eye Generated by BPSK Root 30% Raised Cosine Filter After Modulation

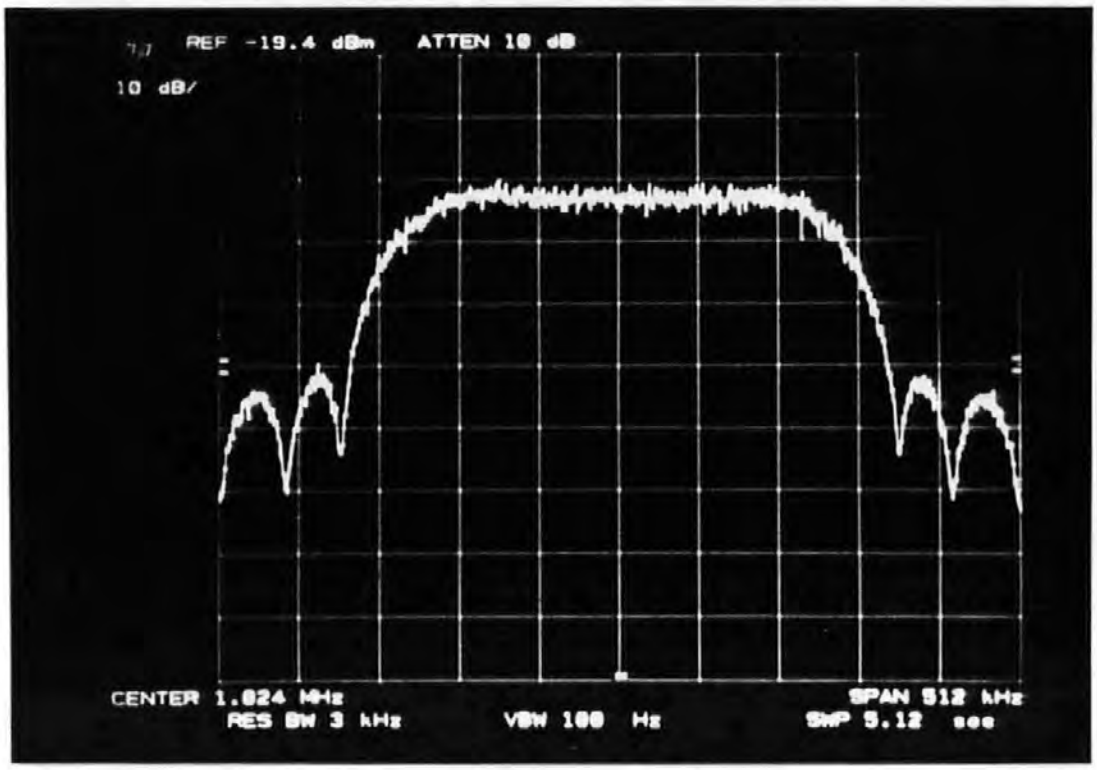


Fig 5.2.1.g BPSK Spectral Response with Root 30% Raised Cosine Filter at 1.024 MHz IF.

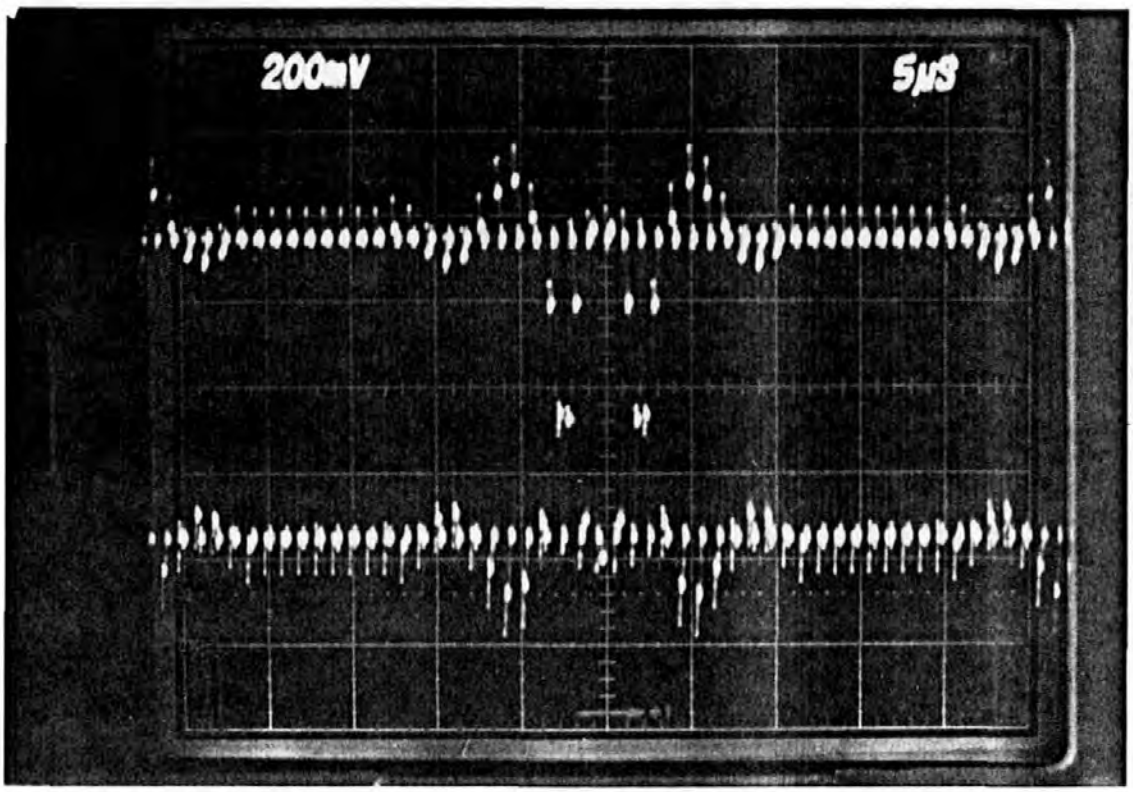


Fig 5.2.1.h QPSK Modulated Root 30% Raised Cosine Filters Impulse Response

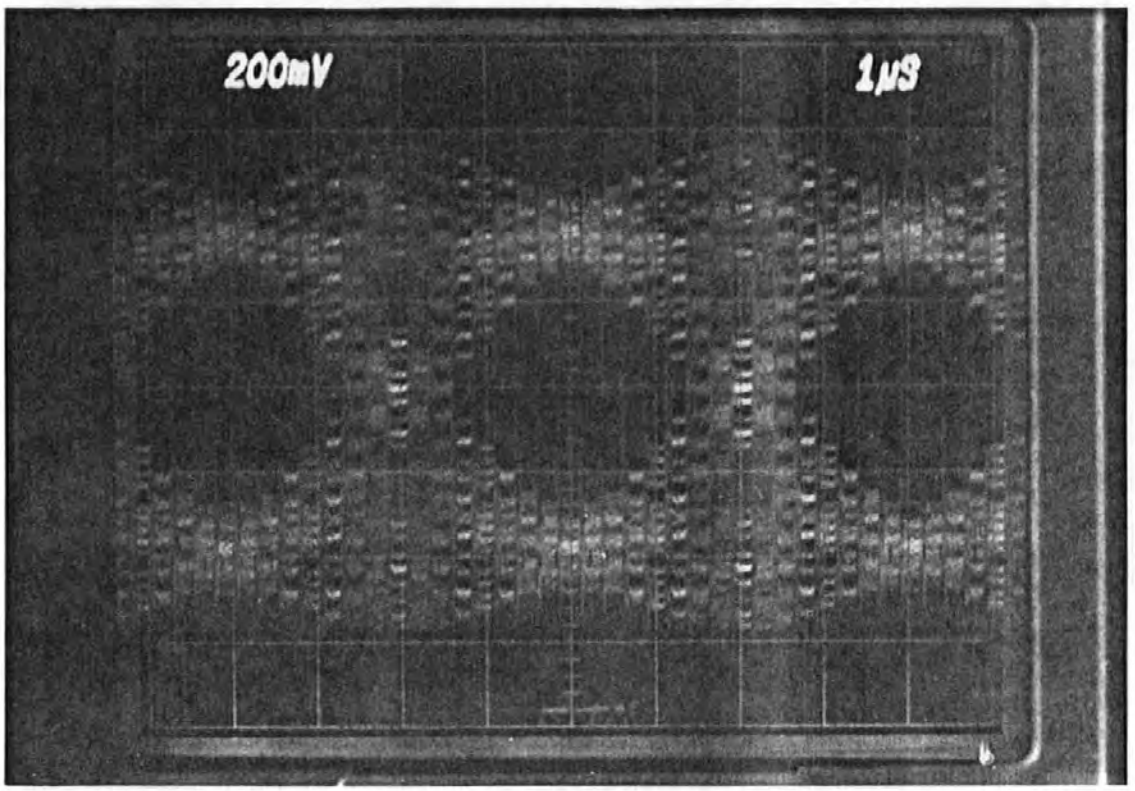


Fig 5.2.1.i Data Eye Generated by QPSK Root 30% Raised Cosine Filters After Modulation

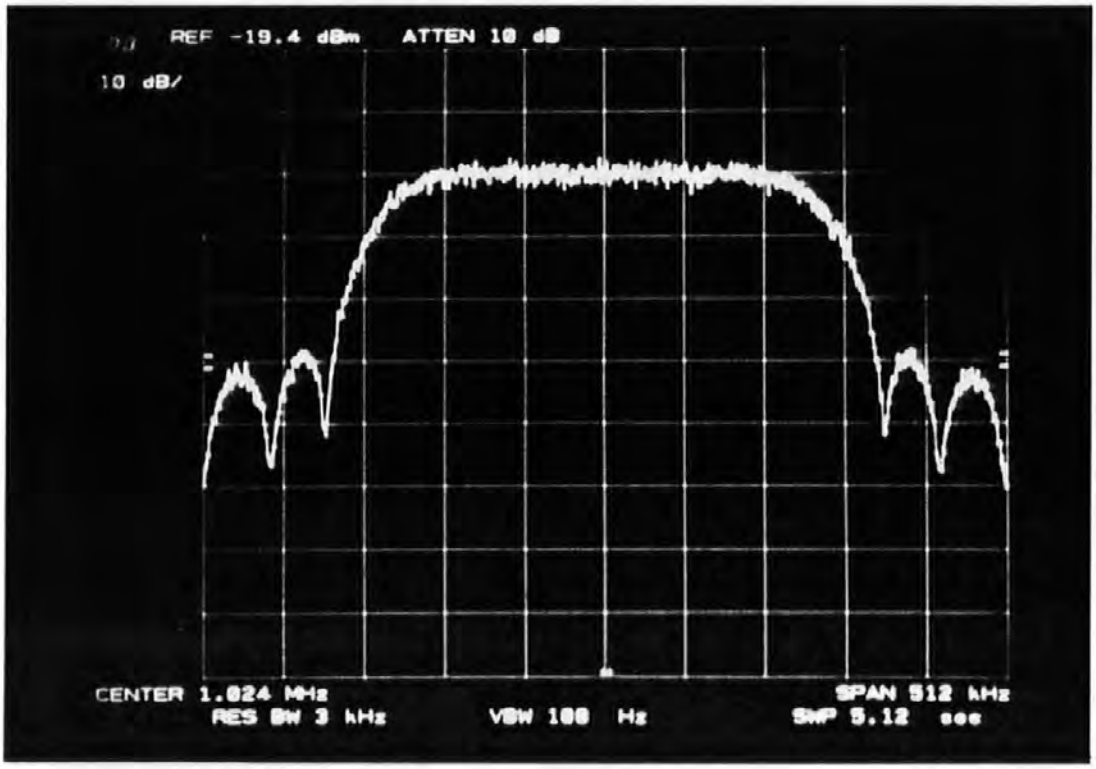


Fig 5.2.1.j QPSK Spectral Response with Root 30% Raised Cosine Filters at 1.024 MHz IF.

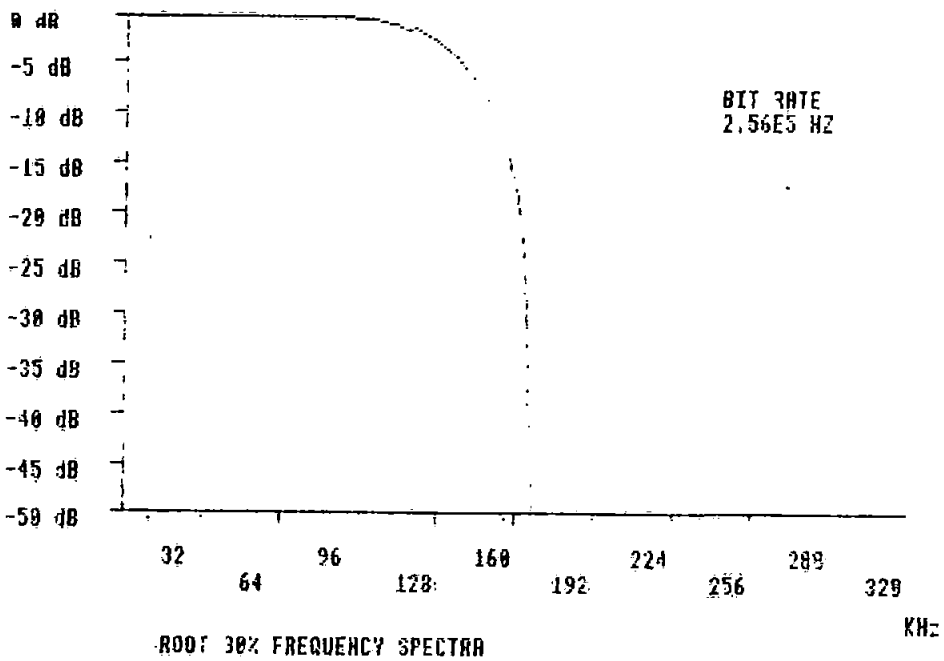


Fig 5.2.1.k Computer Plot of Root 30% Raised Cosine Filter Frequency Response

5.2.2. Demodulator

5.2.2.1. Matched Filter Performance

To evaluate the results of the demodulator we will first evaluate the performance of the quadrature down conversion and spectral shaping to produce the full 30% roll-off raised cosine filter. This is shown for a 256Ksymbol/sec signalling rate with a signal originated by the modulator as a root 30% raised cosine on a 1.024MHz IF. This is received by the demodulator and is down converted and filtered. The Modulator and demodulator clocks are taken as synchronous and no timing recovery is required. The Signal is only passed over the analogue IF interface at 1.024MHz and as such is not effected by any other analogue components. We will first consider the overall impulse response for an impulse in a single channel. The output response of the matched filter can be seen in fig 5.2.2.1.a. It can be seen that the output sample rate is now 4 samples /symbol period and the zero crossings take place at the symbol rate. The Upper trace is the quadrature channel and it can be seen that no quadrature channel interference occurs. The time response for a BPSK data stream can be seen in fig 5.2.2.1.b; it should be noted that at the centre of the data eye that the eye takes on only two major levels with a one quanta variation either side. This would be very difficult to achieve by the use of analogue techniques. The QPSK result can be seen in fig 5.2.2.1.c. Now considering the frequency domain in fig 5.2.2.1.d it can be seen that the spectral shape of the filter output conforms to that of the 30% raised cosine function when compared to the computer plot with stop band frequency components approximately 40dB down. A single filter board can be seen in fig 5.2.2.1.e. The number of filters required per channel is dependent on the symbol rate and hence the system size can be considerable if a large range of data rates is required.

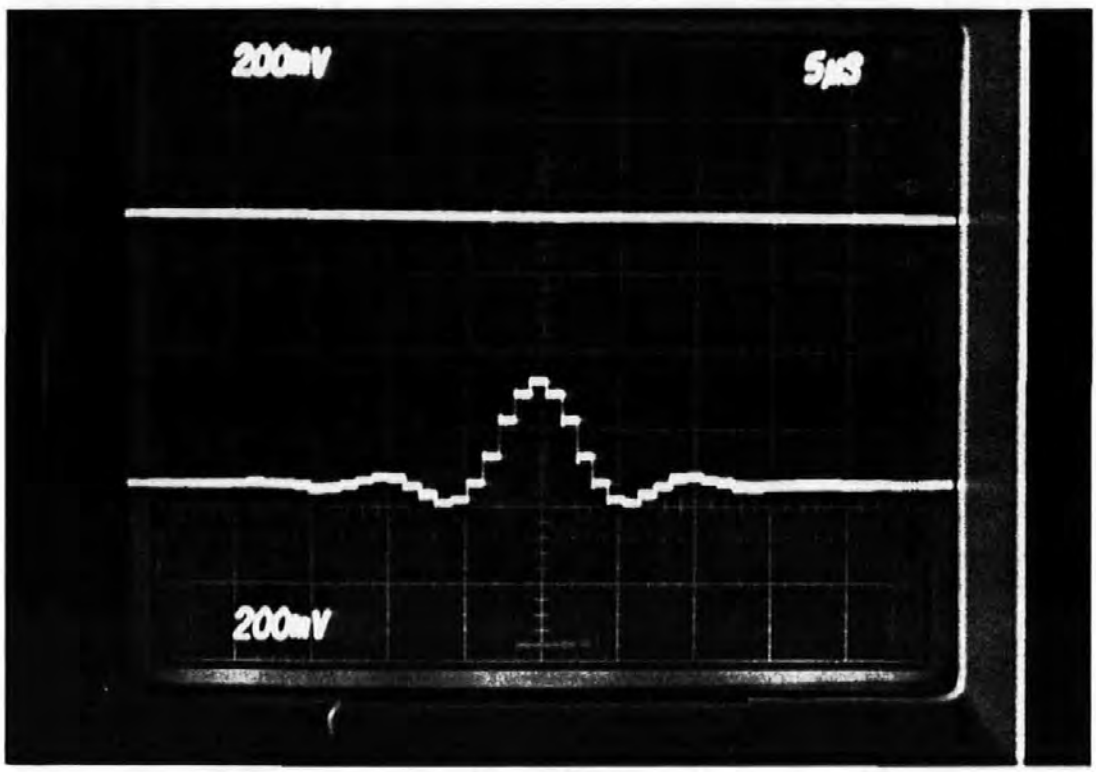


Fig 5.2.2.1.a Matched Filter Output for a Root 30% Raised Cosine Input Impulse

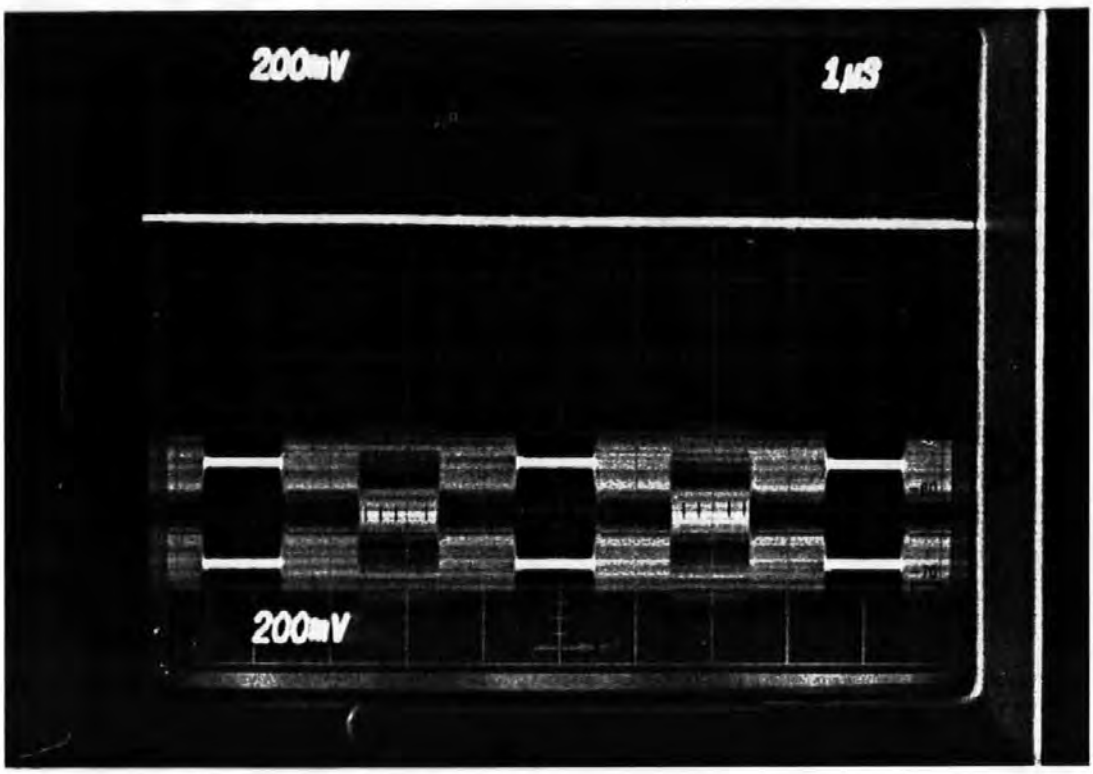


Fig 5.2.2.1.b Matched Filter Output for a Root 30% Raised Cosine Shaped Input BPSK Data Stream

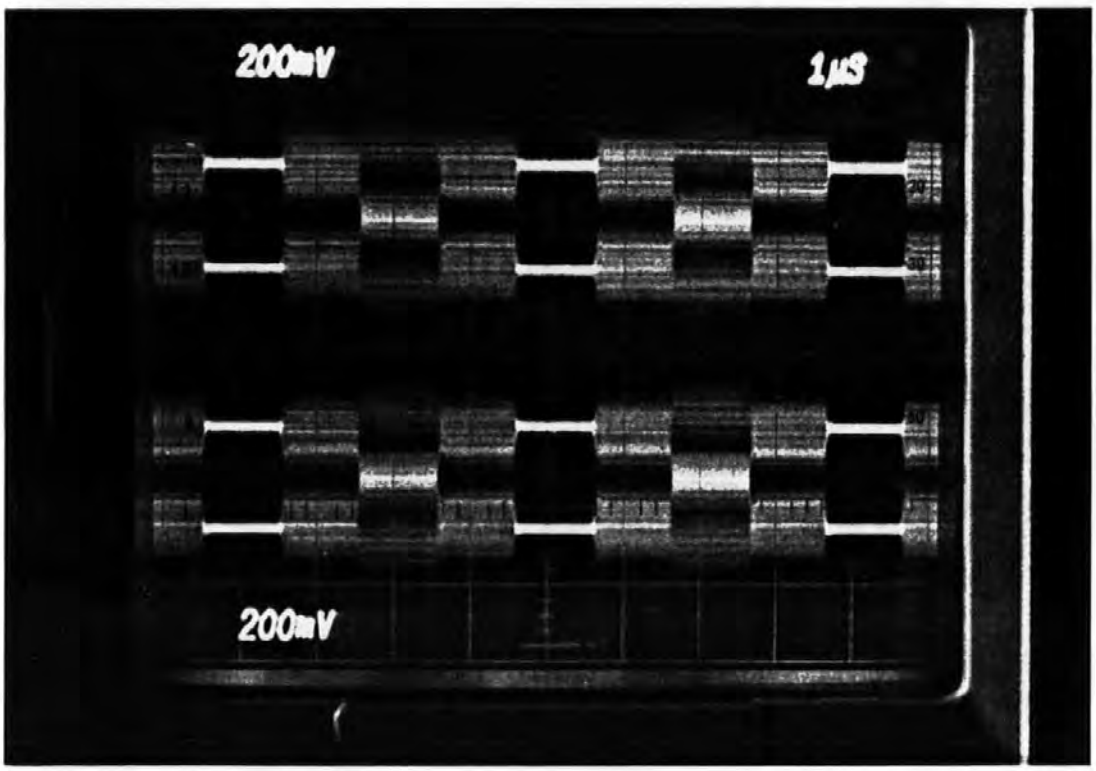


Fig 5.2.2.1.c Matched Filter Output Responses for Root 30% Raised Cosine Shaped QPSK Data Stream Input

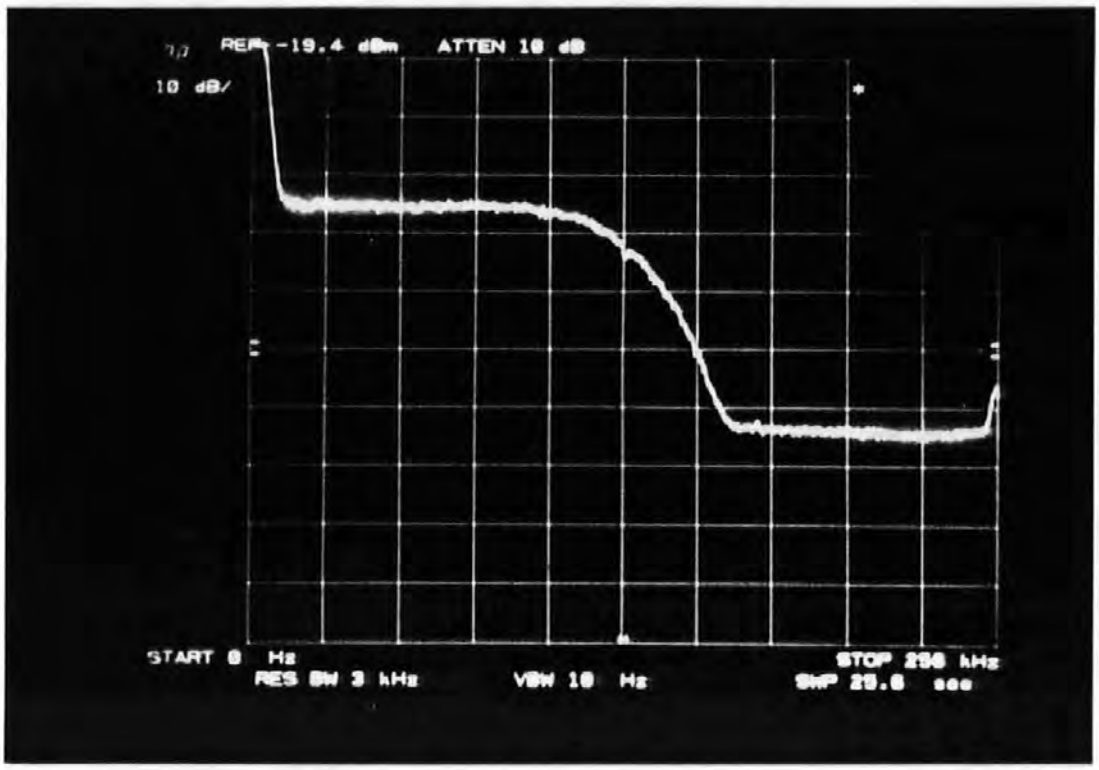


Fig 5.2.2.1.d Matched Filter Output Frequency Spectra Producing an Overall 30% Raised Cosine Response

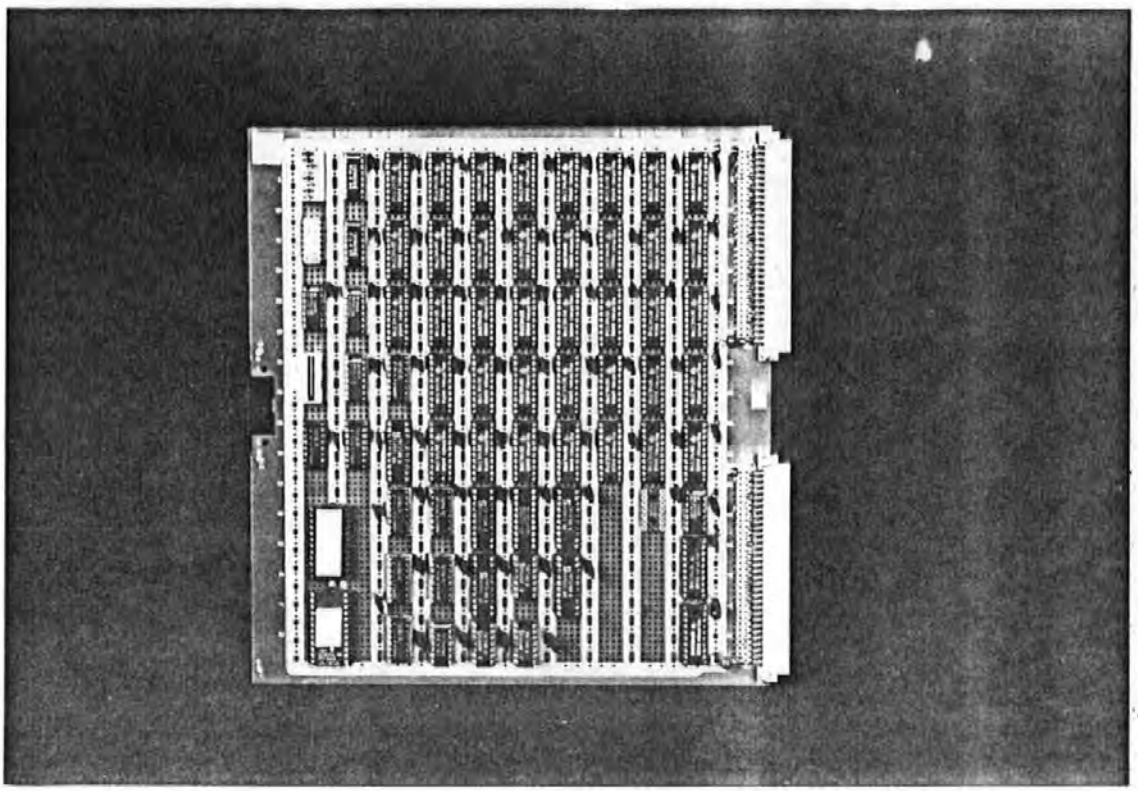


Fig 5.2.2.1.e A Single Digital Filter Board

5.2.2.2. Phase Estimator

The performance of the phase estimator is best considered by an evaluation of the bits error rate of the overall system. However, by the use of Digital to Analogue converters it is possible to monitor the outputs produced at various points throughout the algorithms. A selection of outputs from the phase estimator circuitry can be seen in fig 5.2.2.2 a to c. Performance estimations are very difficult to evaluate in this situation as the performance relies on the variance of the phase estimate and hence the variance would have to be measured to give an accurate performance indication and this could only be achieved by off line monitoring and post processing, for which the equipment was not available and hence no attempt is made at this point. Let us consider the output from the $\Gamma^{-1}(.)$ PROM with a synchronously demodulated signal such that there is no frequency, phase or symbol timing error. For the BPSK case the phase output will be two levels representing 180 degree phase differences. This can be seen in fig 5.2.2.2.a. The slew from one level to another is due to the switching time of the Digital to Analogue Converter (DAC) being used for this output. For the QPSK signal the output will be four phase levels 90 degrees apart. The analogue representation can be seen in fig 5.2.2.2.b. However, it can be seen that the 5 levels are present. This is due to the output being on the 0,90,180,270,360 axis with 0 and just less than 360 degrees although next to each other on the unit circle are being represented by 00 and FF on the digital output which is at the two extremes of the DAC ranges. If a carrier signal of approximately 1.024MHz is fed into the demodulator, with no modulating data, it is possible for the phase estimator to lock onto this carrier and display the phase varying output. This can be seen in fig 5.2.2.2.c. the bottom trace being the output from the $\Gamma^{-1}(.)$ PROM and the upper trace being the phase estimate

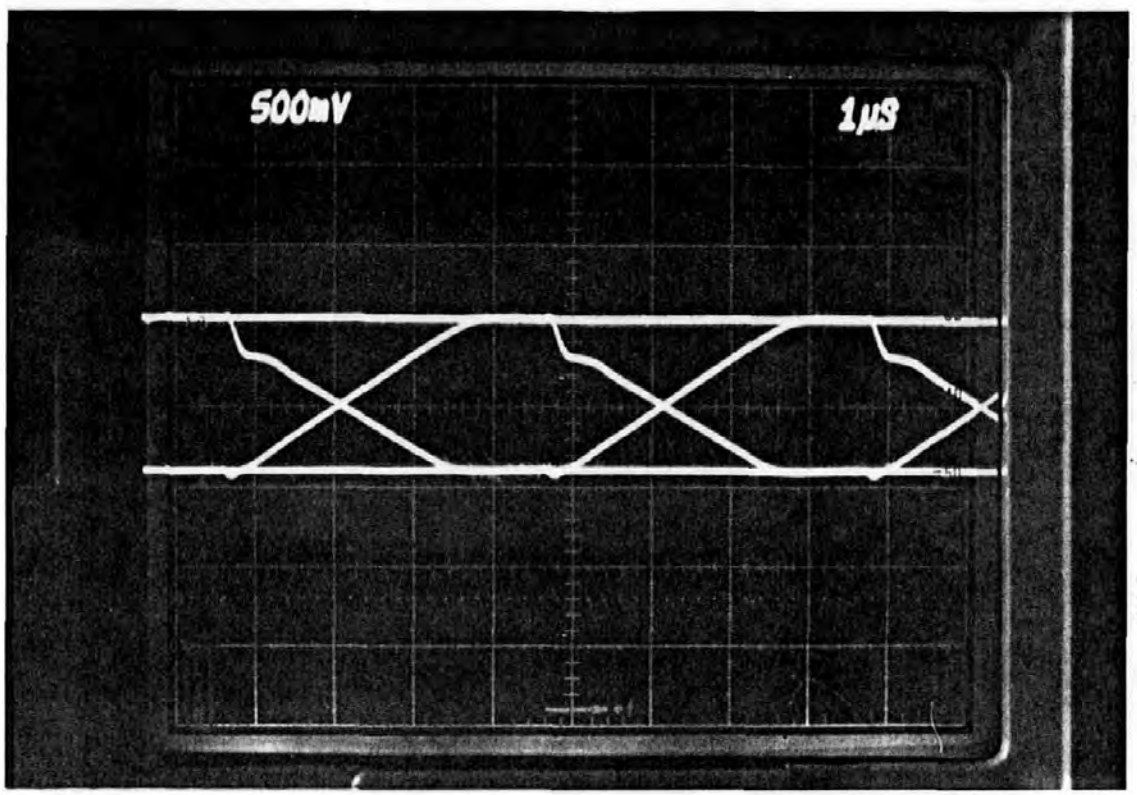


Fig 5.2.2.2.a Synchronous BPSK Phase Output from Decode PROM Representing 0,180 Degrees

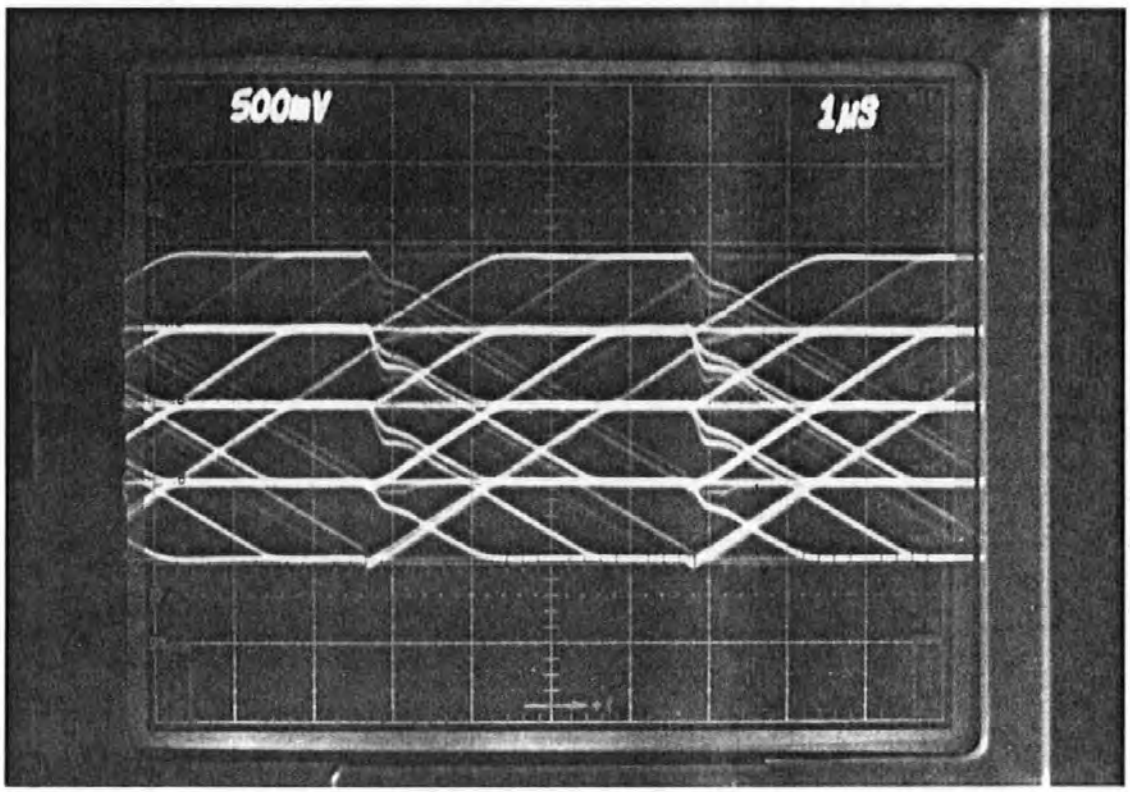


Fig 5.2.2.2.b Synchronous QPSK Phase Output from Decode PROM Representing 0,90,180,-90 Degrees

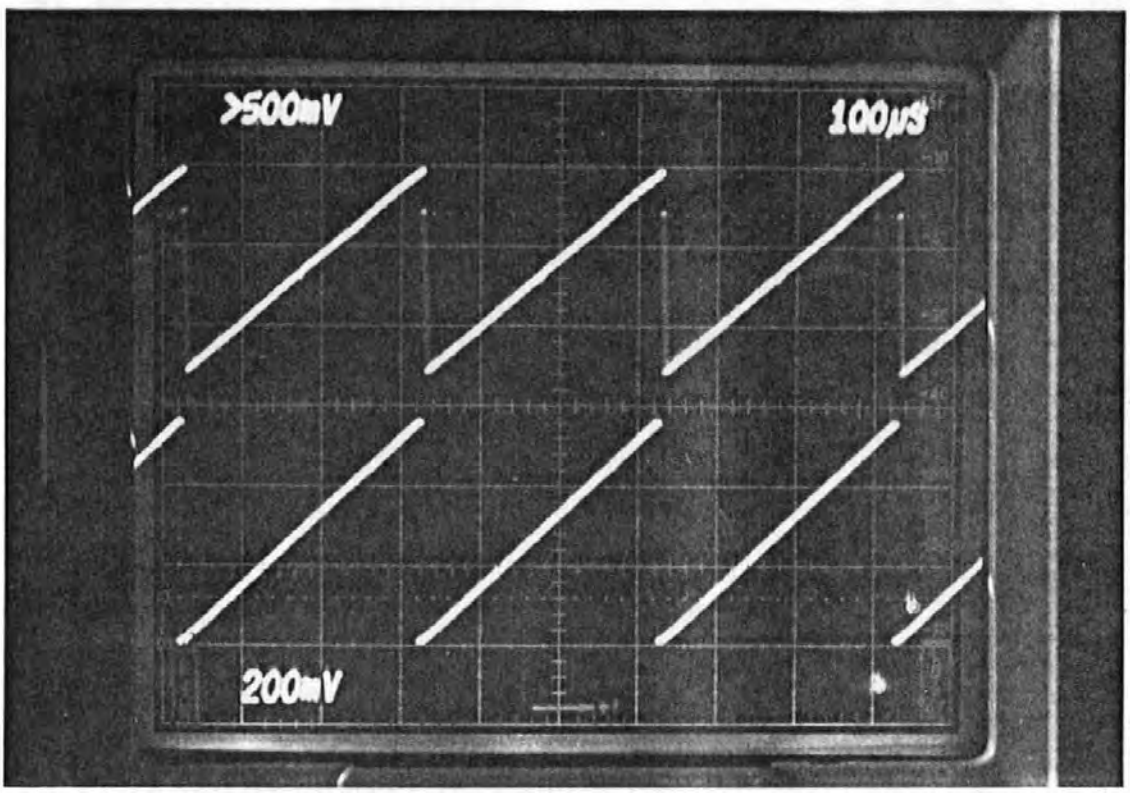


Fig 5.2.2.2.c Phase Estimator (top trace) Locked to a Rotating Phasor Produced by Phase Decoding the Received Carrier Signal by the $\Gamma^{-1}(.)$ Decode PROM Output

from the phase estimator algorithm. In this case the Phase estimator has locked with zero phase error but this could be at 0, 180 degrees for BPSK or 0, 90, 180, 270, for QPSK. The period of the sawtooth waveform is the time required for the phase to change by 360 degrees and hence the slope of the waveform i.e. the rate of change of phase, is the frequency offset from the effective system 'local oscillator'. The Phase Estimator hardware can be seen in fig 5.2.2.2.d.

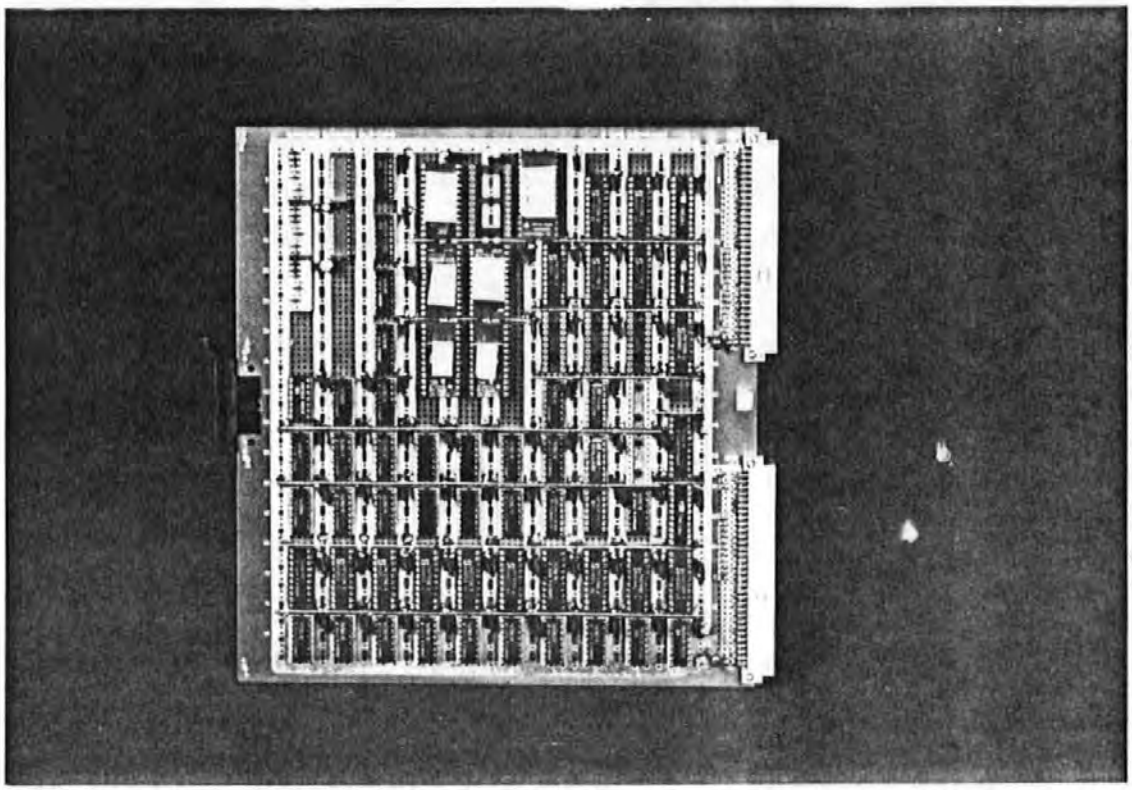


Fig 5.2.2.2.d Phase Estimator Hardware

5.3 Trials Results

Satellite trials where carried out on the digital modem system on the 22-23 May 1986 at Goonhilly Downs, Cornwall, Uk with the help and collaboration of British Telecomm International. Due to the limitation of the number of bits in the frequency filter causing performance degradations the test was limited to using only the phase mean estimate. This could only be achieved by keeping the frequency offset from nominal at the digital IF to a minimum. The Test setup fo the signal path can be seen in fig 5.3.a. The satellite used was Intelsat 5A flight 11 at 27.5 degrees West. The signal could also be routed via a 'Test Translator' for tests when the satellite was unavailable and for more controlled experiments. The signal was provided to the up-conversion equipment at a 70 Mhz IF frequency and obtain from the down conversion equipment at a 70 MHZ IF. Signal to noise ratio was obtained by removing the modulating signal from the transmitter and measuring the carrier power using an HP 8566A spectrum analyser. The noise power was also calculated in a 1 Hz bandwidth by the HP 8566A by measuring the noise power away from the carrier. Corroborating measurements where also made by the use of a power Meter and a Calibrated Filter. This value of Carrier to Noise in 1Hz was then converted to Eb/No by the Formula:-

$$Eb/No = C/No - 10\log_{10} \text{data rate} \quad dB \qquad \text{(Eq5.3a)}$$

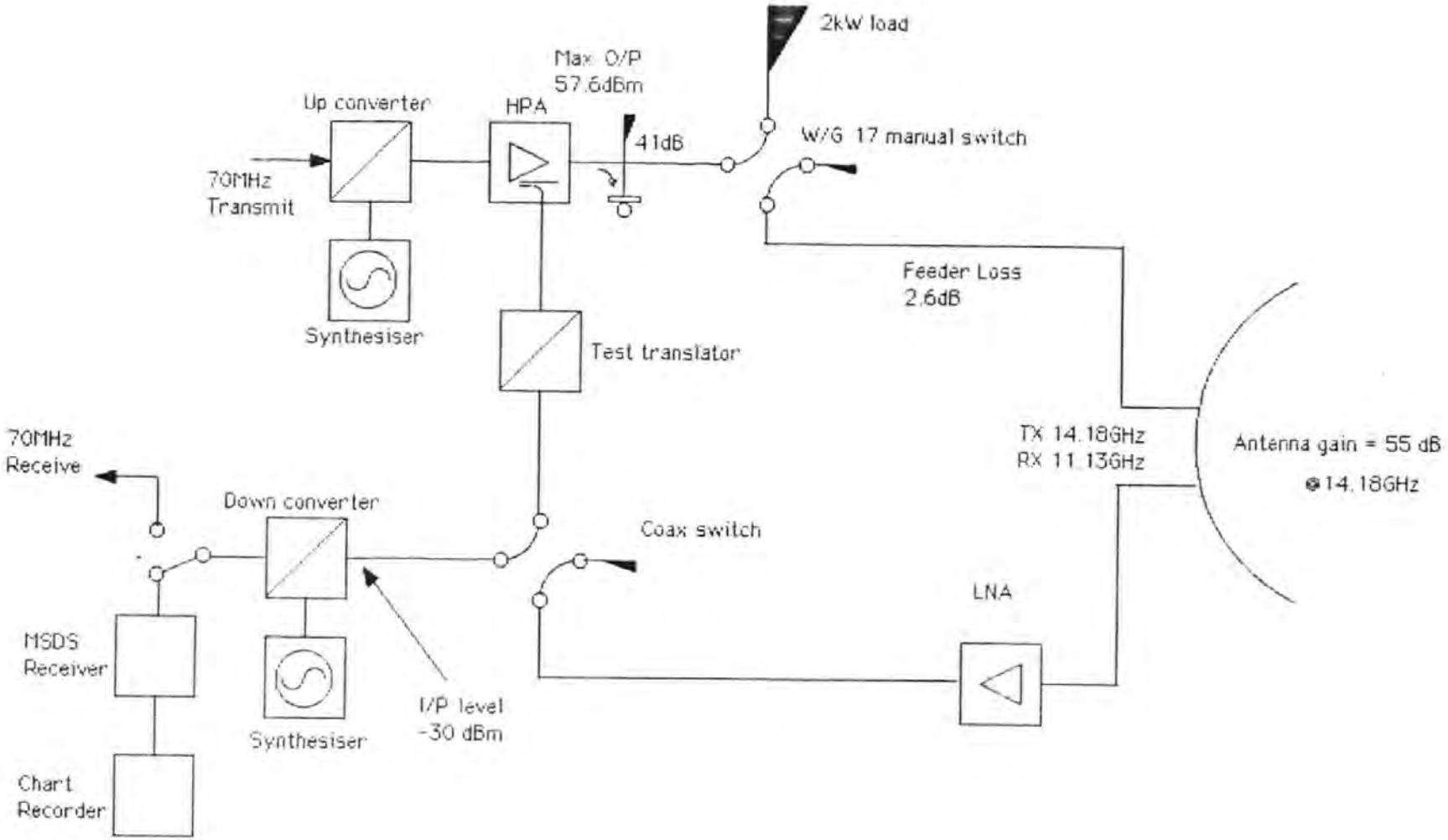


Fig 5.3.a Signal Path Arrangement for Satellite Trials

5.3.1 Experiments Carried Out Over a Simulated Satellite Link

To enable the performance of the modem to be evaluated, and not the degradations caused by the satellite or up/down link equipment, tests where performed taking measurements at various stages in the system to ascertain the optimum method for the satellite trials tests. Considering fig 5.3.1.b and c a series of test were performed for BPSK and QPSK using the test translator to simulate the satellite link. Measurements where taken with the system in:-

i) A back to back mode via the 70 MHz IF interface.
ii) Back to Back via the Test translator.
ii) Back to back via the test translator and the HPA at 8dB backoff and 1 dB backoff.

If we first consider BPSK it can be seen in fig 5.3.1.b, that the modem via the 70MHZ interface has a degradation from theoretical of 0.7 dB over the range of Eb/No from 0 dB to 11 dB. This degradation is within the system as the degradation becomes less at low Eb/No. This occurs as the system becomes of less significance in relation to the additive received noise and hence the effect of the internal system noise is reduced at low SNR.

Introduction of the test translator results in a further degradation of 0.1 dB ,fig 5.3.1.c and the HPA with 8dB backoff a further 0.2 dB, fig 5.3.1.d. With the HPA at 1 dB backoff the degradation from theoretical is now 1.8dB, fig 5.3.1.e. This is due to the satellite TWT (test Translator) now becoming non-linear and introducing AM-PM and PM-PM products, ie any amplitude modulation present on the signal is converted to phase modulation and any phase modulation is enhanced.

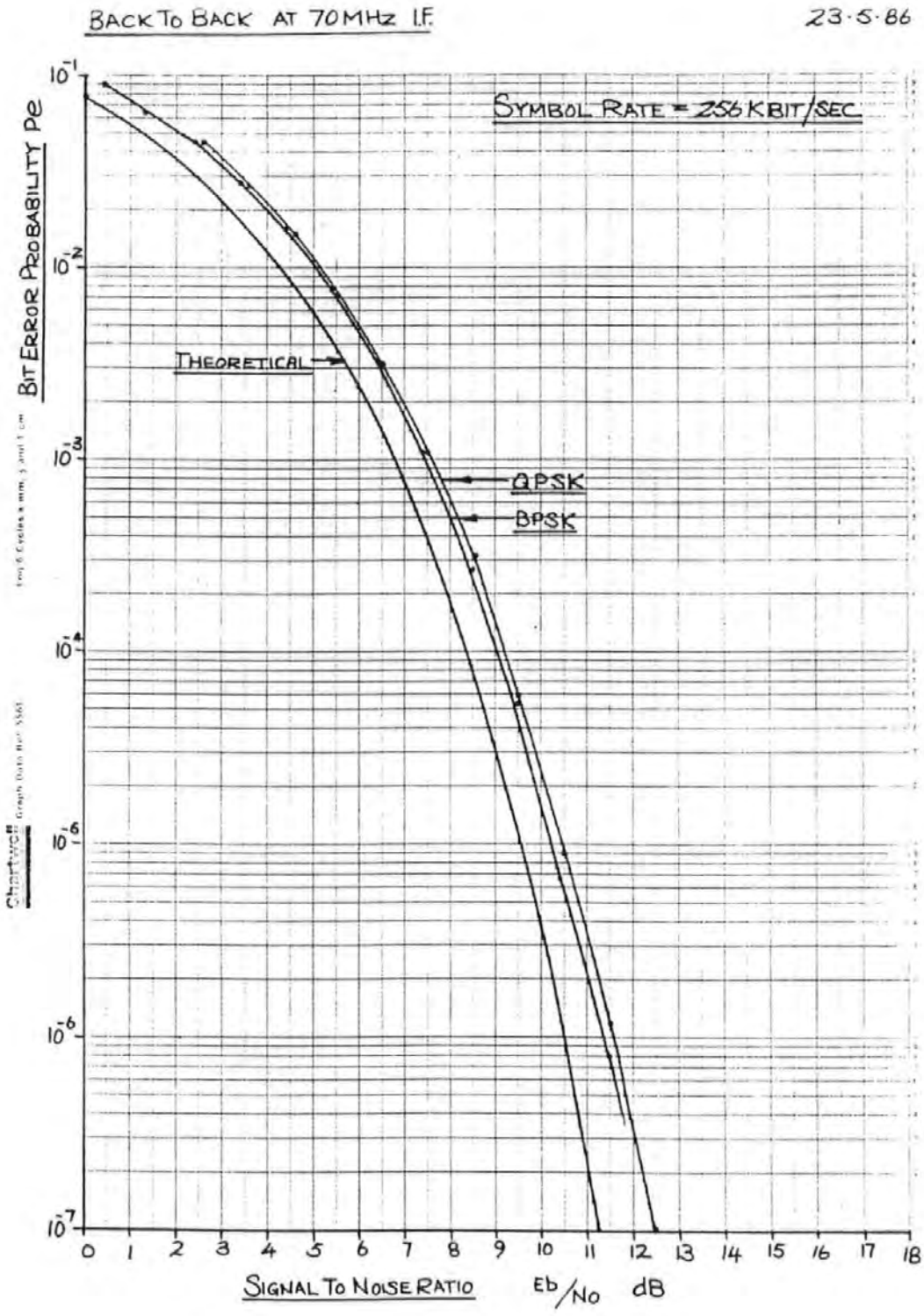


Fig 5.3.1.b 70MHz Interface Loop Back Test Results for BPSK and QPSK at 256 Ksymbol/sec

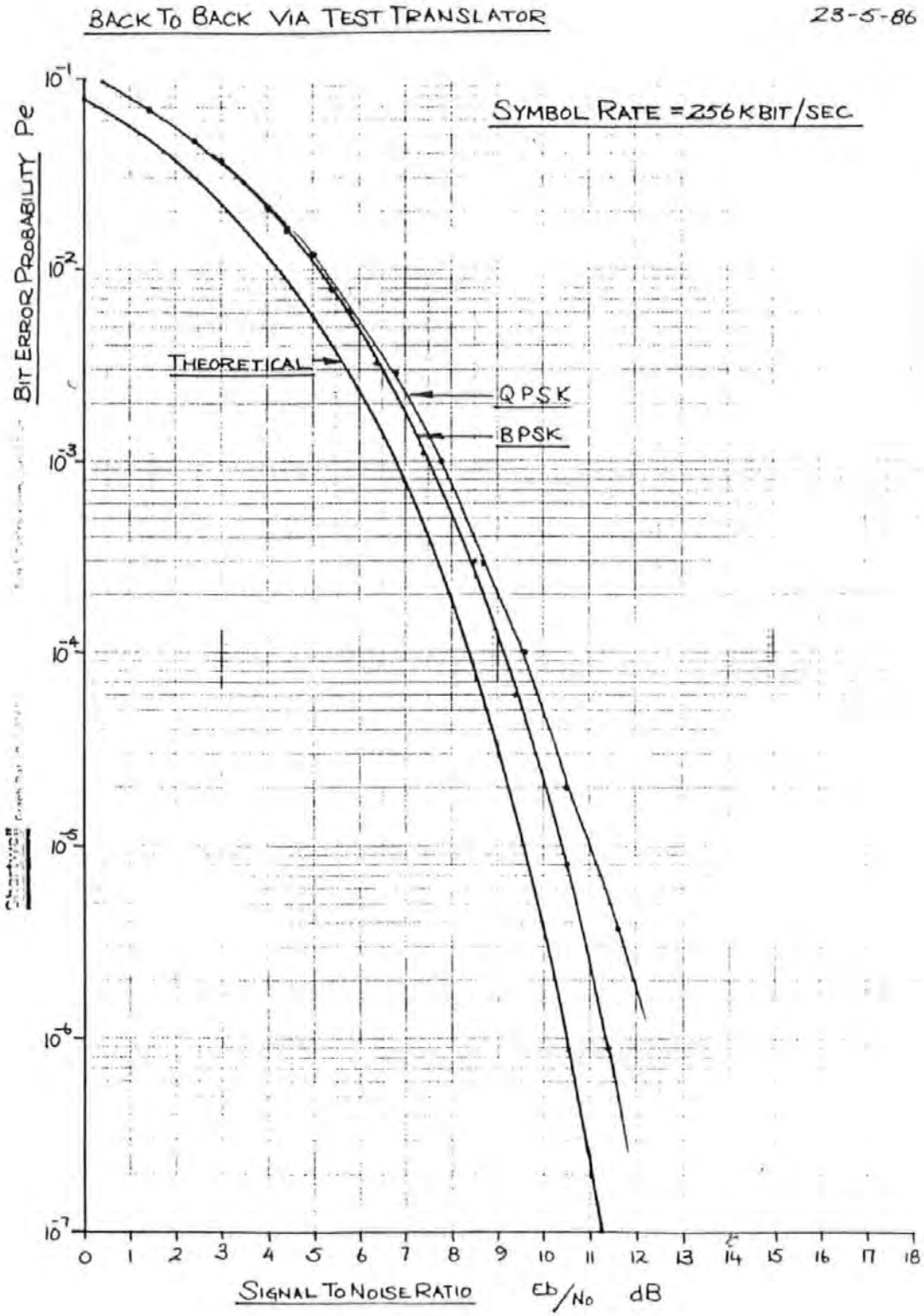


Fig 5.3.1.c Test Results for BPSK and QPSK at 256 Ksymbol/sec via Test Translator at 14GHz

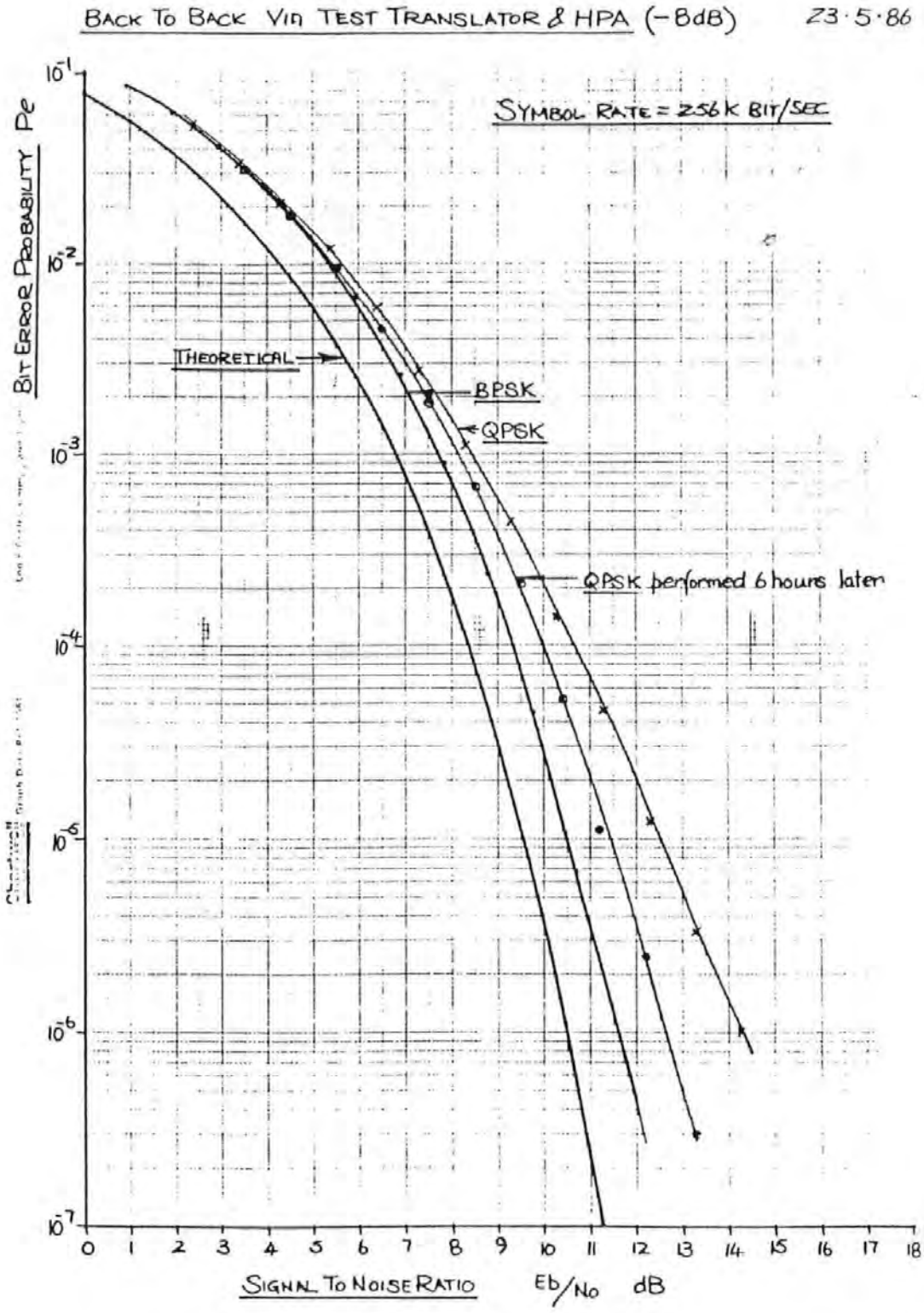


Fig 5.3.1.d Test Results for BPSK and QPSK at 256 Ksymbol/sec via Test Translator at 14GHz and HPA at -8dB Backoff

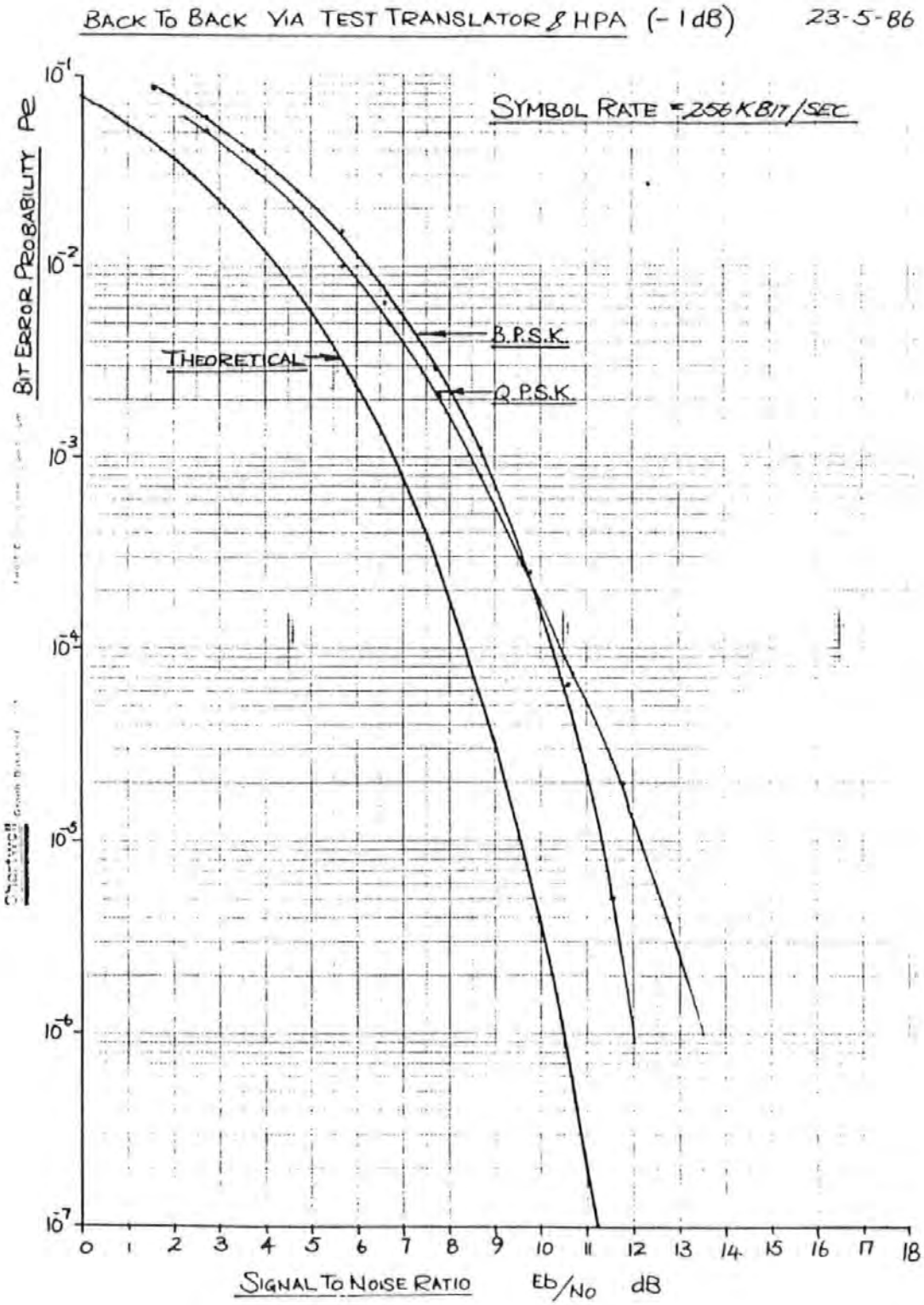


Fig 5.3.1.e Test Results for BPSK and QPSK at 256 Ksymbol/sec via Test Translator at 14GHz and HPA at -1dB Backoff

This is therefore no longer a measure of the modem performance but is now dependant on the Non-linearity of the TWT. This area is therefore to be avoided for modem evaluation tests but this may not be the case in the operational environment. The results from the QPSK tests where similar, however the degradation from theoretical is now between 0.8dB and 1dB over the range 3 dB to 12dB with greater degradation for the effects of the test translator and the HPA. This increased degradation is due to the reduced decision boundary which now occurs at +/- 45 degrees from the transmitted phase position compared with +/- 90 degrees for BPSK and is hence more susceptible to phase noise . This also results in cycle slips occurring at a much higher value than in BPSK and hence the lowest measured Eb/No is 3 dB's greater than for BPSK as discussed previously in section 5.1.4 and confirms the results of the simulation. Considering fig 5.3.1.f and g the relative degradation for each stage in the transmission and reception sequence, over a simulated satellite link, can be seen for BPSK and QPSK respectively. The worst case degradation between BPSK and QPSK can be found for the 70 MHz interface, ie the Modem degradation from fig 5.3.1.b and is 0.2 dB at 11.5 dB Eb/No on top of a 0.7 dB degradation from theoretical.

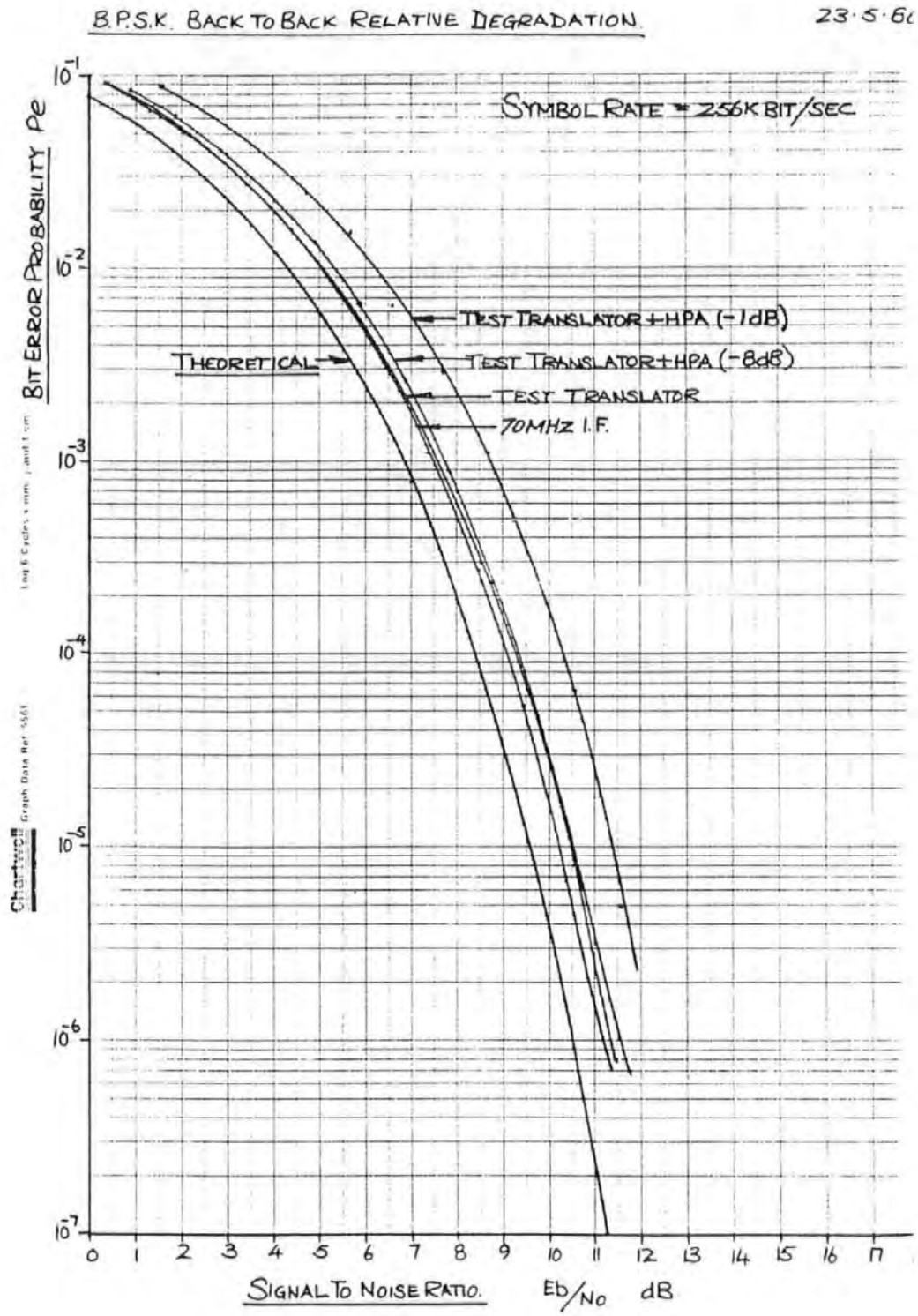


Fig 5.3.1.f BPSK Relative Degradations Throughout the Simulated Satellite Link

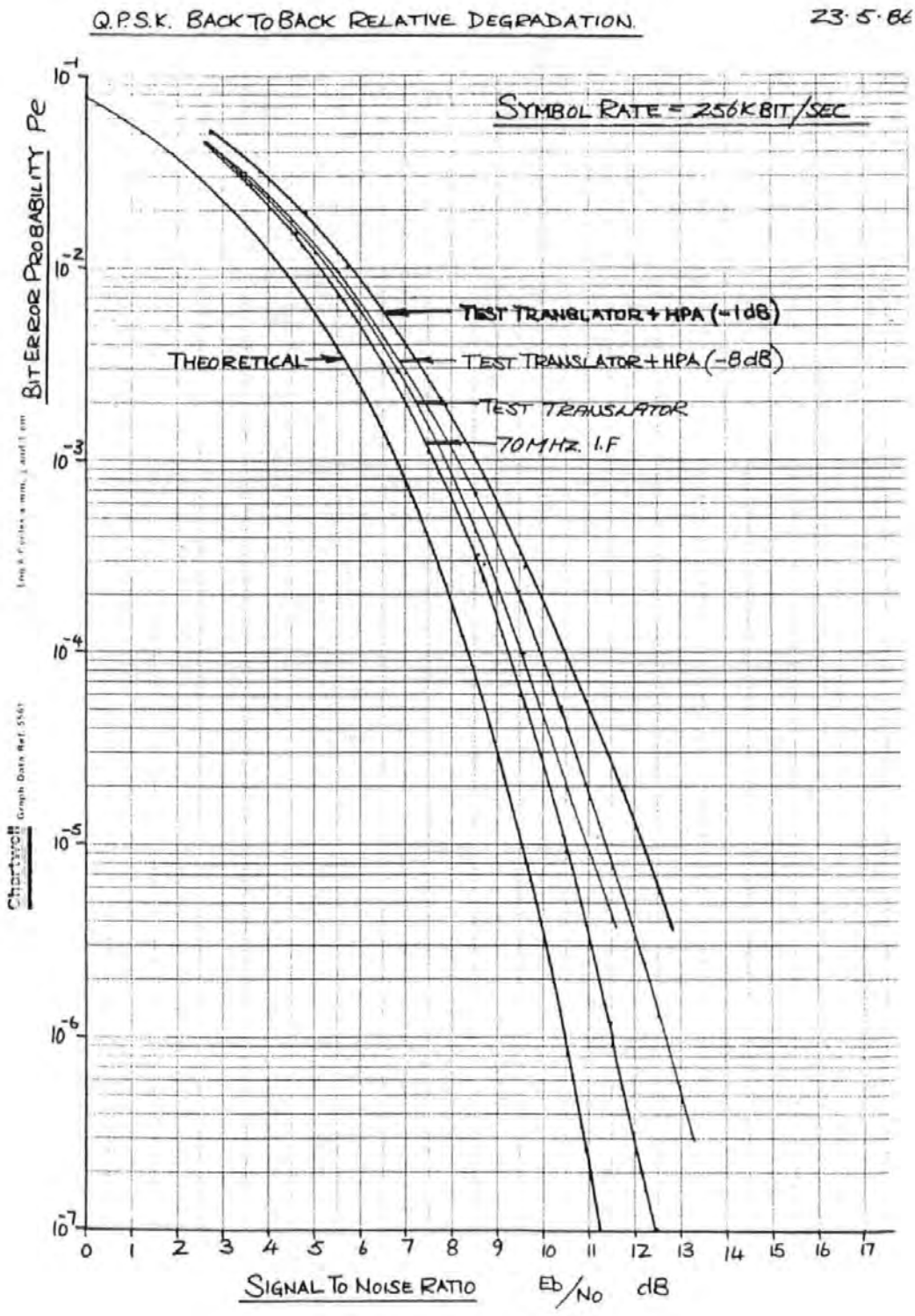


Fig 5.3.1.g QPSK Relative Degradations Throughout the Simulated Satellite Link

5.3.2 Experiments Over an Operational Satellite Link

To investigate the effect of noise over a satellite link the signal to noise ratio i.e. Eb/No must be varied. This can be achieved by 2 methods.

i) Varying the transmitted power level and keeping the satellite system noise floor constant.
ii) Keeping the the transmitted power constant and increasing the additive noise.

Method 1 has the advantage of simplicity but this relies on the linearity of the satellite HPA and that the HPA does not approach saturation. The results from this method are shown in fig 5.3.2.a for BPSK and QPSK. It can be seen that the degradation from theoretical is no longer as constant as in the test translator results although the degradations has not increased by a great amount in either case.

Method 2 is slightly more difficult than method 1 as it required a White Gaussian Noise source to be added to the received signal before demodulation. The Noise source was provided by BTI and added to the the received signal by the use of a passive combiner. The satellite power was maintained at 20 dB backoff to reduce the effects of saturation.The results for BPSK and QPSK can be seen in fig 5.3.2.b. It can be seen that the BPSK results are similar to those obtained at 70 MHz with a degradation from theoretical of 0.8 dB across the same Eb/No range. The QPSK result has a sight divergence from theoretical with a maximum degradation of 1.2 dB. The degradation is again thought to be phase noise related. Comparisons of the satellite degradation against 70 MHz IF can be seen in fig 5.3.2.c and d for BPSK and QPSK respectively.

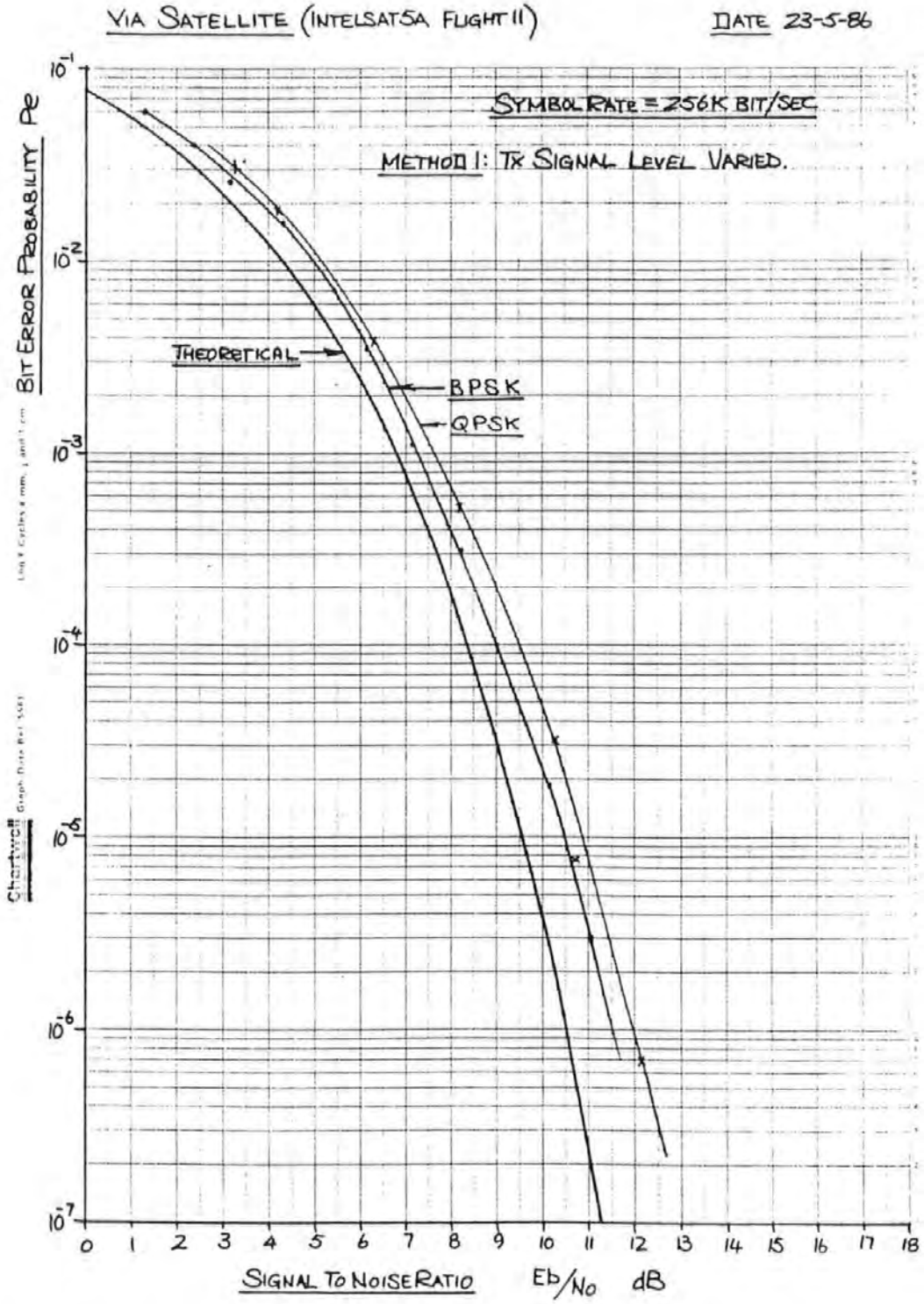


Fig 5.3.2.a BPSK and QPSK BER Curves For Satellite Link Using Method 1

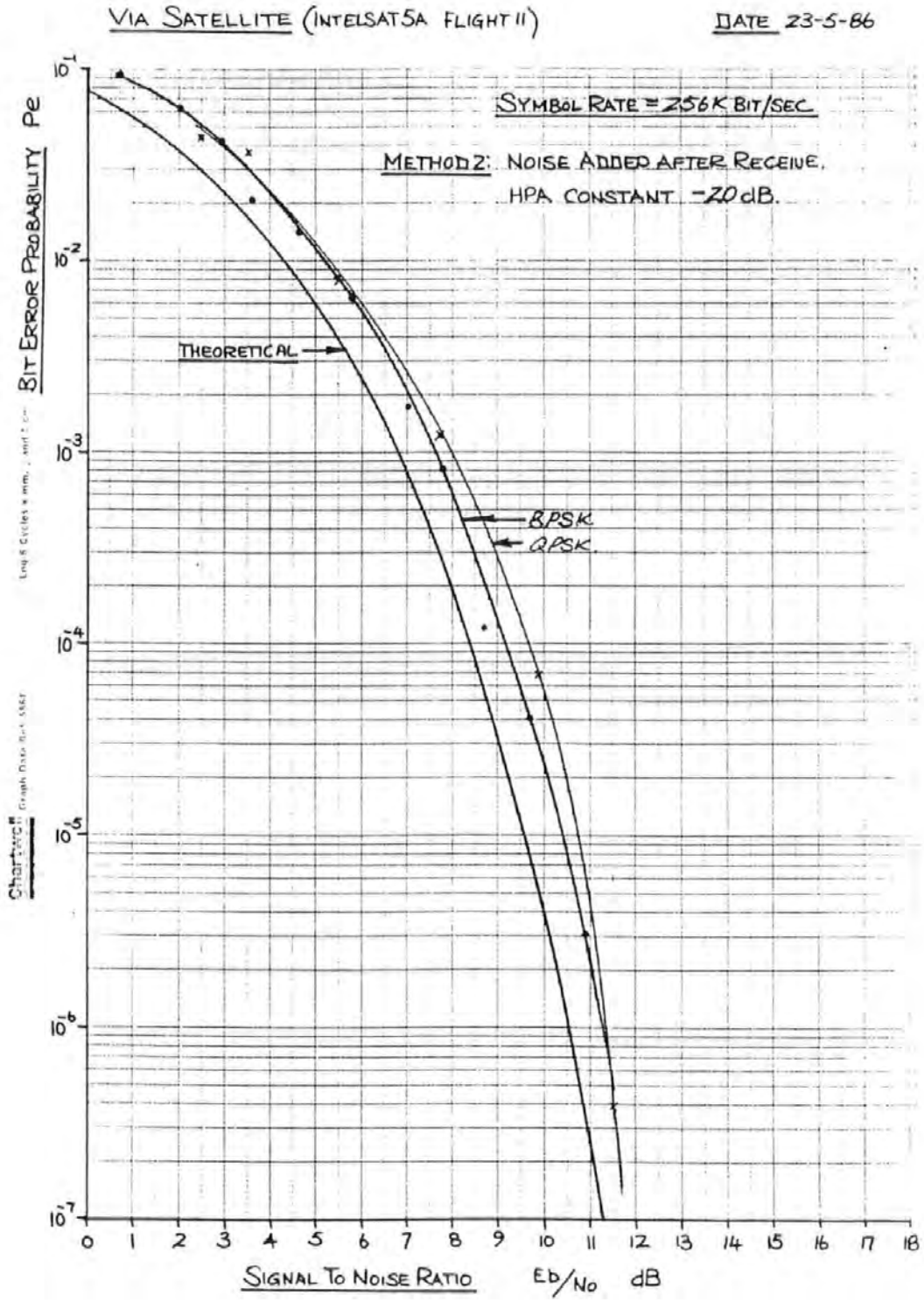


Fig 5.3.2.b BPSK and QPSK BER Curves For Satellite Link Using Method 2

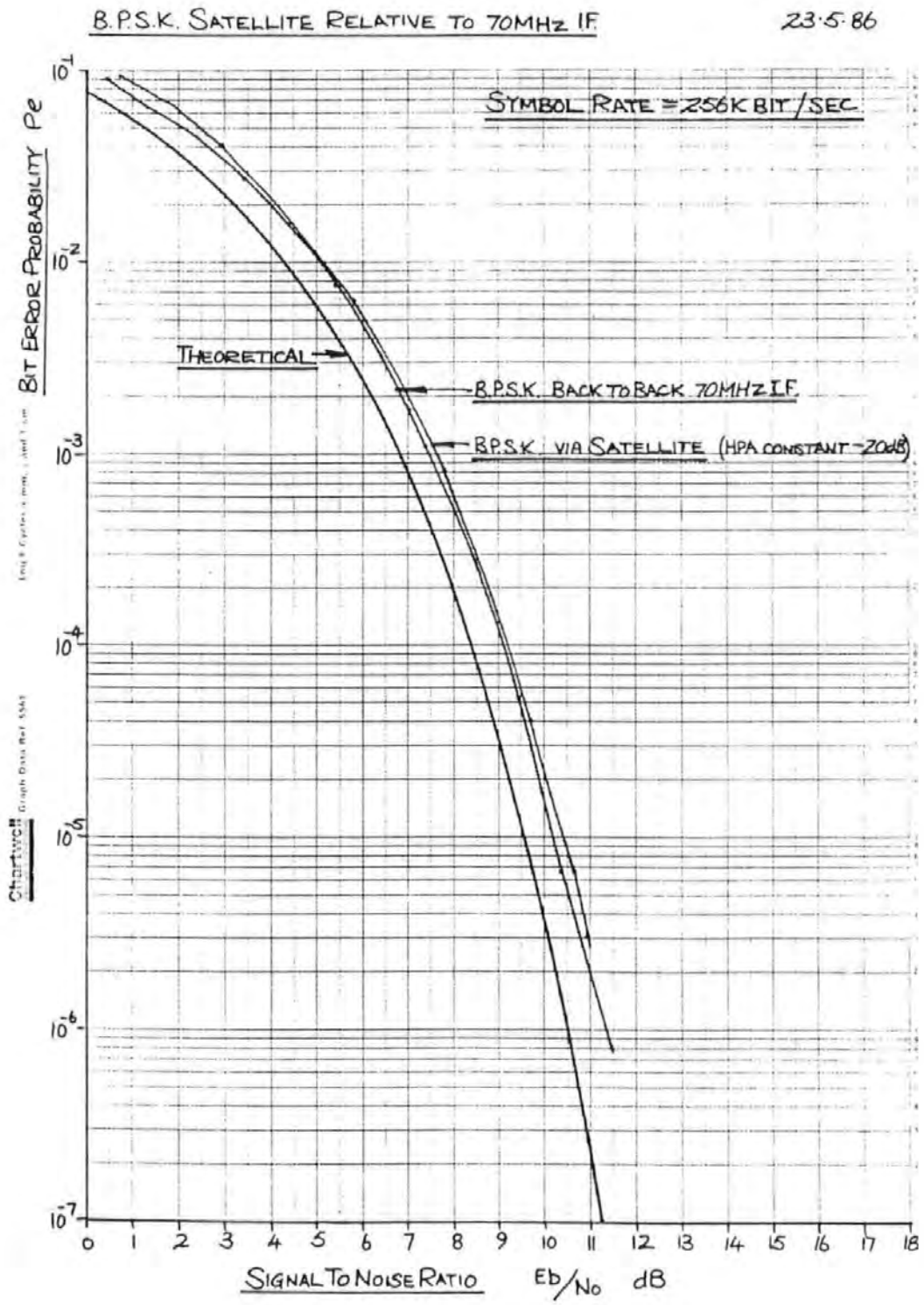


Fig 5.3.2.c Comparison of BPSK BER Curves for the 70 MHz Interface and Via the Satellite Link

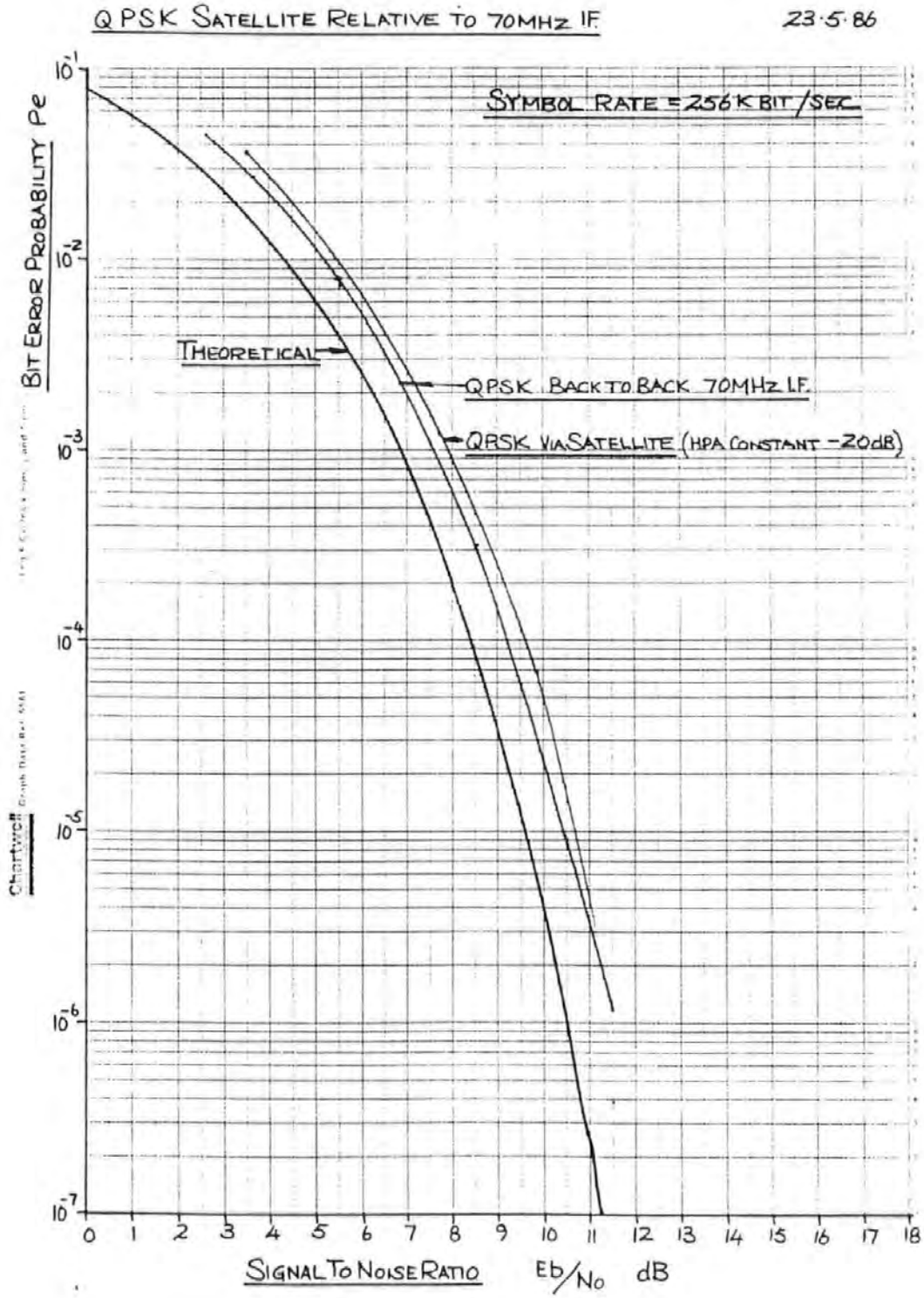


Fig 5.3.2.d Comparison of QPSK BER Curves for the 70 MHz Interface and Via the Satellite Link

6. Conclusions

The previous sections have documented the investigation into the possibility of a digital Modulator and Demodulator for any signal that can be modelled using the quadrature model. In section 2 it was suggested that this model could be used as the basis for an implementation in digital hardware and resulted in the design of a Multi-data rate Phase Shift keying Modem capable of producing a BPSK and QPSK modulated signal. During this investigation an algorithm was hypothesized for the use of a Phase estimator to predict the received signal carrier phase instead of using a phase lock loop system which produced major acquisition and stability problems due to the effects of the long time delays used in the digital filters. This Phase estimator algorithm was first simulated by the use of an IBM AT computer with the algorithm implemented in Pascal. The results from this simulation indicated the need for a frequency and Phase correction algorithm and that the algorithm would operate over a maximum of 4KHZ acquisition range at the symbol rate of 256 Ksymbol/sec. A reduction in symbol rate will also reduce the acquisition range on a pro-rata basis. The accuracy of the filters used in the algorithm where also accessed with an indication that a minimum feed-back path of 15 bits was required to implement the filters without any major degradation in the algorithm implementation.

It has also been shown that the degradation becomes less when the frequency estimate is averaged over more samples. Acquisition time could therefore be traded against performance since the greater the averaging time the greater the accuracy of the frequency estimate.

The Phase estimator then has a minimum acquisition time of:-

$$\text{Min acq. time} = \frac{1}{1-K_p} + \frac{1}{1-K_f} \text{ symbols}$$

where K_x is the feedback coefficient of the phase filter and the frequency filter.

The effects of AWGN on cycle slip performance against frequency offset from nominal, for varying Eb/No also suggest that the more stressed the loop becomes due to frequency offset the higher the probability of cycle slip. This is due to the effects of calculating a phase mean estimate of a variable phase and noise component and then projecting this value forward in time to compensate for the frequency offset or rate of Change of Phase per symbol. This projection becomes more prone to error as the frequency offset increases. It is therefore desirable to try to minimise the frequency offset by some other method before phase tracking is used.

The digital implementation of the modem as also been shown to produce filter implementations of a very high accuracy for filter responses that are non causal and hence produce minimal degradation in received data eyes. The filters where implemented as Root 30% Raised Cosine Filters at both the transmitter and receiver to produce an overall channel response of a 30% Raised Cosine Filter. The complete digital implementation was evaluated in trials at British Telecomm International Earth Station At Goonhilly, Cornwall, over a satellite link and was shown to have a degradation from theoretical of 0.7dB for BPSK and 0.8 dBs

for QPSK with a cycle slip performance threshold of 0 dB Eb/No for BPSK and 3 dB for QPSK .The results of the simulation and experiments of the digital multi-data rate modem demonstrate that the use of digital techniques in the modulation and demodulation of a signal is possible and that high accuracy and small degradations from theoretical can be achieved. Although the implementation resulted in a large number of circuit boards and components the use of digital signal processing has some major advantages over that of analogue systems.

Firstly the individual logic elements may be combined by the use of Very large Scale Integration (VLSI) to reduce the number and size of logic elements used. This will also reduce the production costs.

Secondly, by using digital implementations once the correct system design is achieved , any reproductions of that system can be produced with the minimum of set up as opposed to analogue circuits which require individually setting up.

Thirdly the multi-data rate system has been designed to enable a degree of flexibility to be established when using a satellite communication system. The modem can be configured to allow for the use of a single data rate or of a series of data rates dependent on power and availability of transponder bandwidth or as a customer desires. This can be achieved by the addition of digital filter boards as required but with no other major changes to the system. One anomaly however is that for the lower the data rate required the greater the complexity of the receiver. Fig 6.a shows the complete data modem as configured for 256 Ksymbols/sec.

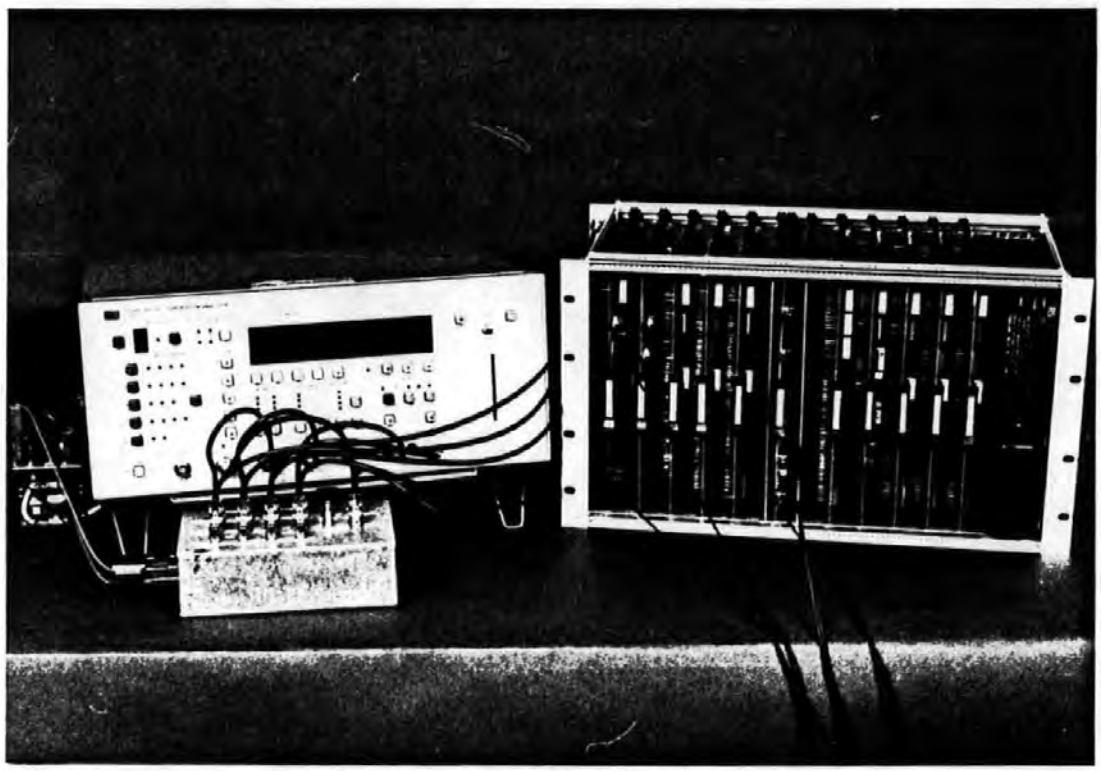


Fig 6.a The Complete Data Modem Configured for 256 Ksymbols/sec.

6.1. Further Investigations

As with all new concepts this dissertation has only scratched the surface of a large area for investigation. Although the digital modem system experimentally developed was successful in that it operated close to theory and for low Eb/No values the system design may not have been optimum from a cost point of view. With the Phase estimator the algorithm used was found to be very successful but no attempt has been made to find the optimum algorithm or the optimum hardware solution for this algorithm. However a few suggestions for areas of further investigation into the algorithms design concepts can be discussed.

6.1.1. Digital Implementation

The investigation has centred around the development of algorithms to implement a Multi-data rate BPSK/QPSK modem for data rates in the range 16Ksymbol/sec to 256 Ksymbols/sec. This range can be extended with present day technology to a symbol rate of 4.096 Msymbols/sec by an increase in the parallelism used in the Modulator and demodulator. The limiting factor is the speed in which multiplications and additions can be performed. This is limited in this design by the access times of the bipolar proms used in the digital demodulator filters. The current design uses a serial multiply and addition for all coefficients. This could be reduced to each prom performing a single multiplication with additions pipelined. This could result in a throughput rate of 33MHz. Another improvement in throughput rate can be achieved by only calculating the sample at the centre of the data eye for the final output of the matched filters and hence the throughput rate becomes equal to the symbol rate with the sampling rate equal to four times the symbol rate. This is approaching the state of the art in digital TTL compatible logic at the present time.

6.1.2. Phase Estimator

Further investigations can be performed to discover the optimum solution for the phase filter. It is presently implemented as a single pole IIR filter with History modification. Could the filter be implemented as an integrator, or as a FIR filter and would the performance increase warrant any extra complexity required?

The current arrangement of the phase estimator has a certain amount of feedback in the system design. This results in the usual manifestations of acquisition and cycle slip. i.e. If the system looses lock due to noise then the resultant cycle slip and acquisition time are dependant on the loop bandwidth. This can be reduced to dependance on the phase filter only by the implementation of a feed forward system as shown in fig 6.1.2.a. In this implementation the frequency estimate is used to correct the phase to within the range of the phase estimator before the phase is filtered and subtracted from the phase plus data signal. This algorithm relies on the use of circular filters and time delays to match the phase estimator output with the received carrier phase. Another system that could be implemented even with the long time delays of the multi-data rate filters requires the use of a frequency locked loop so that a frequency estimation signal is derived from the $\Gamma^{-1}(.)$ PROM after data removal and differentiating and fed back to the ' local oscillator' to correct for any frequency error. The time constant of the loop can be adjusted by changing the multiplying coefficient K_i to achieve a stable system. This method is shown in fig 6.1.2.b. Since the phase estimator is still used to follow phase and coherently demodulate the signal the frequency estimate only has to appear stable enough such that a phase noise degradation is not superimposed on

the incoming signal. Hence the delay around the loop can be much larger than is possible with a Phase locked loop system.

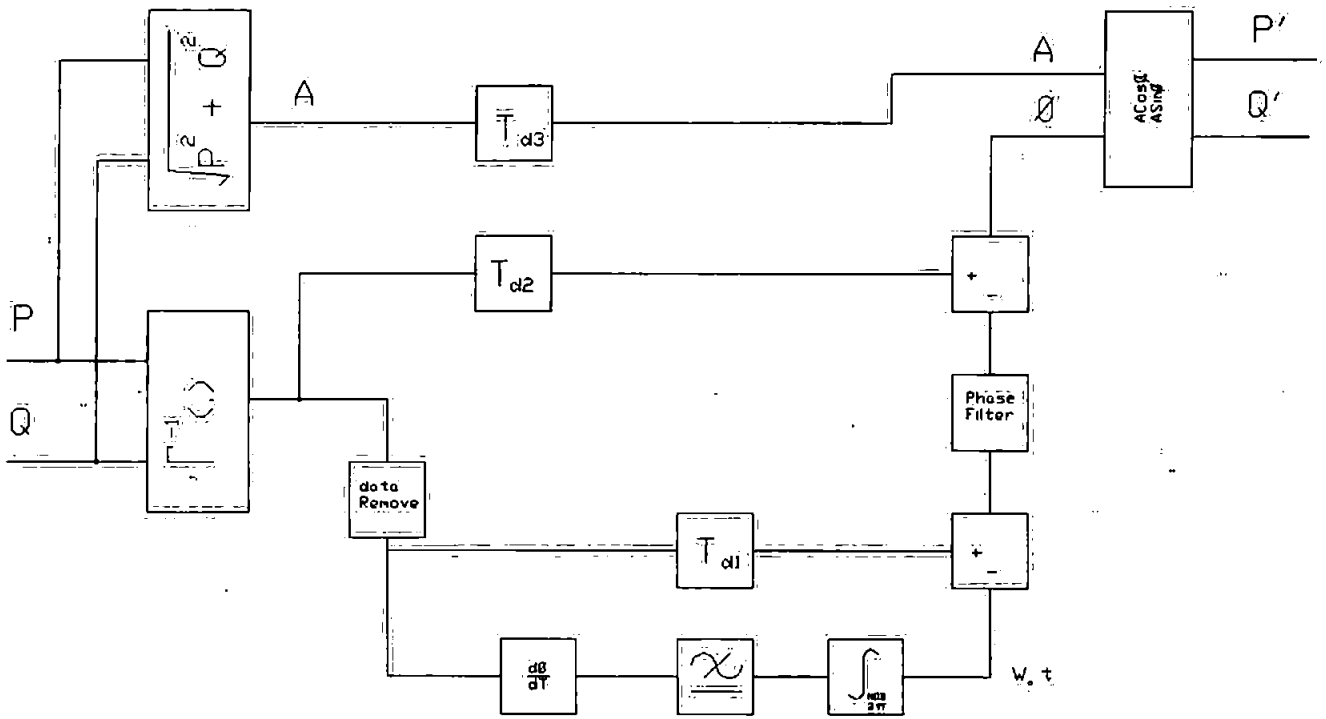


Fig 6.1.2.a Feed Forward Phase Estimator Algorithm

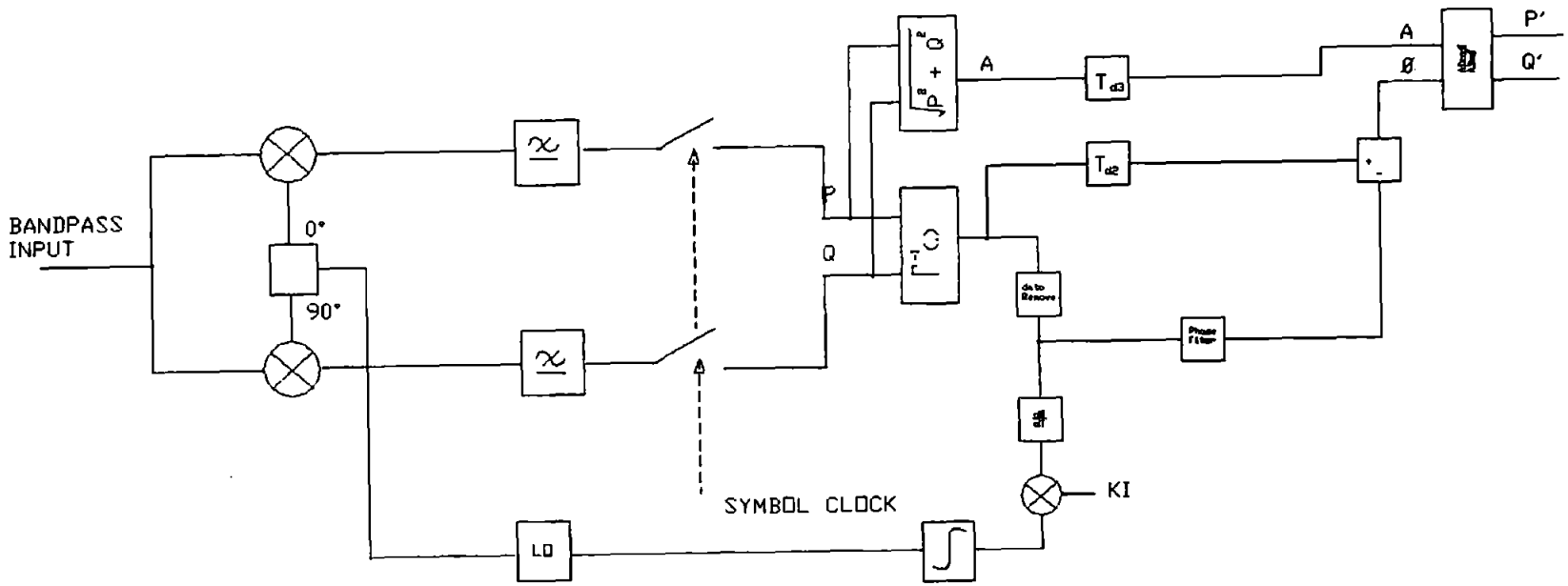


Fig 6.1.2.b Frequency Feedback Loop with Feed-Forward Phase Processing, an Optimum Solution?

Acknowledgements

I would like to thank the following people, who have helped in the research and presentation of this thesis.

Roger Allen - Telemetrix Research Ltd.,

David Sauders - Telemetrix Research Ltd.,

John Everett - British National Space Centre.

Dr Robert Stevens - British National Space Centre.

British Telecomm International Goonhilly Downs, Cornwall.

Prof. Martin Tomlinson - Plymouth Polytechnic.

Prof. Des Mapps - Plymouth Polytechnic.

and especially

Paul Smithson - Plymouth Polytechnic.

Bibliography

<1> Nyquist,H.," Certain Topics in Telegraph Transmission Theory", Trans AIEE, Vol 47, pp 617-644 , Feb 1928.

<2> Shannon, C.E.," A Mathematical Theory of Communication ", Bell Syst. Tech. J., Vol 27, July 1948.

<3> Bhargava,V.K., D.Haccoun, R.Matyas, P.Nuspl, Digital Communications By Satellite, Wiley 1981.

<4> Various, IEEE J. on Selected Areas in Communications, Vol SAC-1, No 1, Jan 1983.

<5> Stremler, F.G.,Introduction to Communication Systems, Addison Wesly Jan 1977.

<6> Couch II, L.W, Digital and Analogue Communication Systems, Macmillan 1983.

<7> Lucky,R.W., J.Salz and E.J. Weldon, Principles of Data Communications, McGraw-Hill, 1968.

<8> Rabiner,L.R and B.Gold, Theory and Application of Digital Signal Processing, Prentice-Hall, 1975.

<9> Feher,K., Digital Communications,Prentce-Hall,1981

<10> Lindsey,W.C. and M.K.Simon, Telecommunications Systems Engineering, Prentice-Hall,1973

<11> Imbeaux, J.C.," Performance of the Delay-Line Multiplier Circuit for Clock and Carrier Synchronization in Digital Satellite Communications ", IEEE J. on Selected Areas of Communications, Vol SAC-1, No 1, Jan 1983.

<12> Gardener,F.M., PhaseLock Techniques, Wiley,1979.

<13> Proakis,J.G.,Digital Communications, McGraw-Hill, 1983.

<14> Palmer, L.C., S.A.Rhodes, and S.H.Lebowitz, "Synchronization of QPSK Transmission via Communications Satellites ", IEEE Trans on Comms,Vol Com-28, No 8, Aug 1980.

<15> Modestino,J.W. and K.R.Matis," Interactive Simulation of Digital Communicatins Systems" IEEE J.Selected Areas of Comms, Vol.SAC-2, Jan 1984.

<16> Palmer,L.C. "Computer Modelling and Simulation of Communications Satellite Channels ", IEEE J. Selected Areas of Comms, Vol. SAC-2, Jan 1984.

<17> Braun W.R. and T.M.McKenzie, " Class: A Comprehensive Satellite Link Simulation Package ", IEEE J. Selected Areas of Comms,Vol. SAC-2, Jan 1984.

<18> Jeruchim, M.C., " Techniques for Estimating the Bit Error Rate in the Simulation of Digital Communication Systems", IEEE J. Selected Areas of Comms,Vol. SAC-2, Jan 1984.

<19> Tomlinson, M. and J.R.Bramwell," Data Communication Method and Apparatus ", British Patent Application, Jun 1984

<20> Golomb,S.W, et al ," Digital Communications with Space Applications",Prentice-Hall, 1964.

<21> Tomlinson,M. and Bramwell, J.R." Real Time Digital Signal Processing in Satellite Modems", Proc. Comms86 1986, IEE,London.

<22> Gardner,F.M.,"Carrier and Clock Synchronization for TDMA Digital Communications", European Space Agency Tech. Memo ESA TM-169(ESTEC), pp232-233 Dec 1976.

<23> Stiffler,J.J." Theory of Synchronous Communications", Prentice -Hall, 1971.

<24> Costas ,J.p."Synchronous Communications", Proc IRE vol 44, pp1713-1718 Dec 1956.

<25> Develet, J.A." The Infuence of Time Delay on Second -Order Phase-Lock Loop Acquisition Range" ,Proc.Int Telemetering Conf ., vol 1, sept 23-27, 1963 ,pp432-437,IEE,London.

<26> Viterbi,A.J and Viterbi, A.M " Non Linear Estimation of PSK-Modulated Carrier Phase with Application to Burst Digital Transmission", IEEE Trans on Information Theory, Vol IT-29, no 4, July 1983.

<27> McVerry, F. "Performance of a Fast Carrier Recovery Scheme for Burst-Format DQPSK transmission over Satellite Channels", 6th Conference on Digital Communications, 1985

<28> Jaffe,R. and Rechtin,E. ," Design and Performance of Phase-lock Circuits Capable of Near Optium Performance Over a Wide range of Input Signal and Noise Levels", IRE Trans Inform. Theory, Vol IT-1,pp66-76, Mar 1955.

<29> Meyer,S.L., "Data Analysis for Scientists and Engineers" p74, McGraw-Hill.

<30> Heller,J.A. and Jacobs,I.M.,"Viterbi Decoding for Satellite ans Space Communications", IEEE Trans. on Communications Technology, Vol COM-19, pp835-848, Oct 71.

<31> "Pro Pascal Users Manual", version iid 1.2, May 1984.
Prospero Software.

Proceeding of Communication 86 Conference, NEC, Birmingham,UK.
May 1986, IEE Press ,London.

REAL TIME DIGITAL SIGNAL PROCESSING IN SATELLITE MODEMS.

M.Tomlinson and J.R.Bramwell

Department of Communications Engineering, Plymouth Polytechnic.

Abstract
This paper contains a brief overview of the ongoing research carried out at Plymouth Polytechnic and Telemetrix PLC, sponsored by the Department of Trade and Industry, into the use of digital signal processing (DSP) to provide accurate modelling of the channel transmit and receive filters, up and down conversion, carrier and symbol timing recovery. Due to the digital nature of the modem, changes in configuration can easily be performed by micro-processor resulting in a User friendly Multi-Data rate modem operating over a range of 16kbits/sec to 4.096Mbits/sec. This, coupled with virtually zero ISI and optimal bandwidth shaping filters, produces a high performance system capable of operating close to theoretical predictions.

Introduction.

The use of the quadrature model in the field of communications has enabled many forms of modulation to be developed, one of which is Quadrature Phase Shift Keying, QPSK. By the use of this model it is possible to convey twice as much data over a bandwidth limited channel but at the same symbol rate as Binary Phase Shift Keying. However with the Quadrature model the bandwidth and shape of the modulated signal is determined by the baseband filtering, if any. If Non-Return to Zero (NRZ) pulses are used i.e. no baseband filtering the bandwidth required would be considerable. Bandlimiting these pulses with little care results in the introduction of Inter-Symbol Interference (ISI) and hence a degradation of the Bit Error Rate results.
However, Nyquist <1> proposed that ISI could be reduced by the use of a filter that conforms to a skew symmetric transfer function. One such family that conforms to this specification is that of the Raised Cosine function. This enables the 3dB bandwidth to be contained to half the symbol rate frequency ,fo. Furthermore there will be no spectral components beyond the frequency fo+fx, where fx is a percentage of fo. This results in the specification of a filter as a percentage roll-off factor, where 100% leads to spectral components out to the symbol rate, 2fo, and 0% has no spectral component above fo and is often termed a Brick Wall filter (fig 1). The penalty for this reduction in bandwidth is an increase in oscillatory tails of the impulse response. This results in a need for more accurate symbol timing recovery for zero ISI, and the need for non-causal filters. Frequently only 100% Raised Cosine filters are used due to the difficulty in producing symmetrical impulse responses accurately enough to give zero ISI from analogue components. It has been proposed by the authors <7> that DSP can be used to increase performance in modem systems. This paper shows that the use of DSP in a real time environment is a realistic proposal and can achieve any reasonable form of filter function that may be required to produce close to zero ISI at the receiver while confining the bandwidth. An added advantage of the digital approach is the easy switching of bandwidths enabling a Multi-Data Rate modem to be produced.

The Modulator, A Brief Overview.

Consider a previously scrambled,hence pseudo-random data stream being presented to a modulator where by it will be converted into a QPSK signal format and transmitted to up-conversion circuitry on a 1.024MHz First IF carrier. The conventional approach would be as describe by Feher <2>,Couch <3>,Proakis <4> and many more, with spectral shaping either taking place in the baseband channels or at the carrier frequency using analogue methods. The basic diagram can be seen in fig 2. The incoming data sequence is multiplexed into two data sequence at half the incoming data rate. These two parallel data sequences or NRZ pulses can then be filtered for efficient spectral shaping before modulating onto a IF carrier in quadrature.

Digital Transmit Filters.

The roll-off factor for the overall channel spectrum is chosen to be 0.3 or 30%. This figure is a compromise between channel bandwidth requirements and jitter tolerances on the symbol recovery circuitry giving rise to ISI. Higher roll-off factors i.e. 50%, produce wider bandwidths but a higher jitter can be tolerated before ISI degrades the data eye. It was decided to implement Root 30% Raised Cosine Spectral responses in the both the transmitter and receiver to conform both to Nyquist Criteria and Match Filter criteria as proposed by Lucky, Salz, and Weldon, <5>.In the transmitter the incoming data pulses are of the NRZ format and hence only ascribe to two values +1 and -1.The convolution process is therefore limited to <6>:

$$y(kT)=\sum_{n=-\infty}^{\infty} h(nT).x((k-n)T) \qquad (1)$$

where x(iT)= +1 or -1. In a digital system the impulse response of the filter must be truncated to allow for an economical implementation. The consequence of this is an increase in stopband side lobe power and hence a compromise must be reached. In this case the truncation was chosen to be at +/- 3 symbol periods and results in a stopband 30 dB down at the 30% point. Hence the convolution can be approximated by:

$$y(kT)=\sum_{n=0}^{5} h(nT).x((k-n)T) \qquad (2)$$

hence only a limited number of coefficient multiplications and additions need to take place, but there is still another saving that can be made. Consider the data sequence not as a series of NRZ pulses but as a series of pseudo-random impulses with a flat normalized spectral response. The output spectrum from any filter is therefore defined by the filter spectrum only and hence its impulse response. The digital filter now consist of the convolution of two impulse responses. This can be considered as in fig 3 for a filter impulse response which is only significant over 6 symbol periods. In the centre of the sequence the tails from all the filter responses converge. For this region of interest it is possible to define the intermediate results for any input sequence of 6 symbols i.e. calculate the convolution results and store them in memory. If the sampling rate of the filter impulse response has been correctly chosen with a 6 symbol input word it is possible to store all combinations of results in a PROM. Due to the symmetry between the two quadrature channels it is also possible to use the same information for both channels, if not the same PROM. If the sample clock is therefore set at 4.096MHz then a 16Kbyte Prom would be required for a 16Ksymbol/sec symbol rate. By the use of extra PROMS or intermediate sampling more data rates could easily be accommodated.

Digital Up-conversion to First IF.

Consider a sequence of 4.096MHz samples output from the above digital filter PROM fig 4a and we require a carrier frequency of 1.024MHz. The sampled In-Phase carrier waveform can be consider to be as in fig 4b. If these two waveforms are multiplied together the result is as in fig 4c i.e. a 4.096MHz sample stream with alternate samples equal to zero. If we consider another sample stream multiplied by the quadrature carrier to fig 4b the resultant is as in fig 4d and it can be clearly seen that again a 4.096MHz sample stream results with alternate samples equal to zero however for corresponding positions in the In-phase channel when the quadrature channel is zero the In-phase is non-zero and vice versa. The output quadrature carrier IF therefore consists of the In-phase and quadrature channels added together which in this implementation is simply a multiplexing of the two channels. By correct choice of the phase reference between the two channels the carrier can be implemented within the PROM and the complete modulator circuit becomes that of fig 5. For a BPSK signal format the incoming data is only diverted to the In-phase channel and the quadrature channel output is set to zero. The use of other data rates can be accommodated by scaling the clocking frequencies appropriately and hence the IF output frequency.

Multi-rate Digital Demodulator.

From the previous discussions of the modulator it is readily apparent that the demodulator may be based on the same concept i.e. the digital demodulator consist of an input signal on a minimum carrier frequency of 1.024MHz. This is digitally converted to baseband in quadrature and then digitally filtered and sample rate reduced to enable Root 30% raised cosine filters to be implemented practically while covering a span of symbol rate from 16Ksymbols/sec to 256Ksymbols/sec. The output sample rate from these filters is four times the symbol rate. From these four samples per symbol one of the samples must be extracted as the peak sample instant corresponding to zero ISI (when in conjunction with a matched transmitter filter) and the center of the data eye. Carrier phase reference with the 1MHz IF must also be achieved to enable data to be decoded from the received Phase and Quadrature samples. To enable both of these processes to occur the symbol timing must be first recovered and then a carrier reference can be established.

Digital Filter Implementation.

The Root 30% Raised Cosine Filter that is required for the receiver must be linear phase to minimize group delay variations and must hence be of a Finite Impulse Response Design (FIR). The format used in the design filters was as shown in fig 6 and can be termed a Direct Form FIR Filter with Serial Addition. The convolution technique relies on the coefficients of multiplication i.e. the desired impulse response being multiplied by the incoming samples, the rate at which this convolution takes place results in the scaling of the frequency response obtained from the filter. It is therefore possible to change the scaling of the frequency response by changing the processing rate of the filter.

Practical Implementation.

The initial bandlimiting of the received signal to prevent aliasing prior to digitizing, is obtained by the use of a SAW filter for phase linearity, producing minimal group delay variations. The incoming signal, in the range 52-88MHZ, is first up-converted to 200MHz, the frequency being mainly dependent on the availablity and cost of SAW filters and then bandlimited to a 700 KHz bandwidth. The bandlimited signal is then down converted to a 1.024MHz minimum carrier frequency before being digitally sampled. This sampling takes place at a 4.096MHz rate to fulfil sampling theorem requirements and allow synchronization with the symbol clock. In practice it was found that two or three times the carrier frequency did not fulfil requirements as the spectrum will extend to 1.8 MHz with the particular SAW filter used. This sampled signal is then digital mixed to baseband before filtering. This reduces the requirement for closely phase equalized low pass filters before the sampling process can take place. The method used for down-conversion to baseband is obtained by a similar argument as in the modulator. Assume that the incoming signal is on a carrier locked to a quarter of the sampling frequency, down converting with a 'local oscillator' synchronous with the sampling frequency produces alternate zero value samples in the Phase and quadrature baseband channels. This can be modelled by the multiplication of sample pairs by 1.024MHz local oscillator i.e. +1 and -1 which is implemented by complementing samples. Demultiplexing of these samples into the phase and quadrature channels, with the addition of a zero value sample in the opposite channel while one channel is being enabled, results in two baseband 4MHz sample streams. The addition of the alternate zero term into a channel when the data sample is diverted into the other channel allows sampling to occur at half

rate than would normally be required for the 4 MHz sample rates per channel. It is necessary to maintain the sample rate in each channel to enable the switching of data rates over the given range and reduces the effect of the sinc function imposed on the frequency spectrum by the rectangular sample pulses.

Multi-Symbol Rate Receive Filter.

Consider the 16Ksymbol/sec case: the incoming sample stream is highly over sampled but cannot be reduced in sample rate due to aliasing constraints imposed by the SAW filter. One alternative is to change SAW filters. This can be very expensive for a large number of symbol rates. Another alternative was proposed and implemented. The digital filters reduce the bandwidth of the 4MHz sample stream such that sample rate reduction can take place.i.e.. Sub-sampling of the output data. This is the approach taken which results in the use of switchable Primary filters. Each one containing the same impulse response coefficients but produce different bandwidth scaling due to the reduced input sample rates and hence processing rate. The lower the data rate the greater the number of primary filters that must be used to reduce the bandwidth to a managable level for pulse shaping.The final filter is designed to equalize the frequency response of the primary filters and provide the final shaping to obtain the overall root 30% Raised Cosine Filter. The final output rate is four times that of the symbol rate to allow a recognizable data eye to be monitored.

Carrier Reference Acquisition and Data Decoding.

The receiver is required to make an optimum decision on the received signal at the centre of the data impulse only and hence as long as the decimation in time keeps the correct samples no degradation occurs. Accurate symbol timing is required and can be obtained from the phase and quadrature channel or from the input carrier IF signal. However this system requires that the sampling at the A/D input must be synchronous with the symbol timing signal and the input carrier frequency if an optimum decision is to be achieved.One method to allow both the symbol timing and the carrier frequency to be synchronized relies on phase locking the carrier with the system clocks synchronized to the symbol timing component. Another method using a digital approach allows the incoming carrier frequency to be close to that of the digital system down-conversion frequency but not locked to it in any form.If the frequency offset is not to great this signal can be processed by the filters to produce minimal degradation in the impulse response . If the incoming carrier frequency is not synchronous then quadrature channel interference results. Due to the symmetry of the incoming signal the channel outputs become:-

P channel

$$r(t) = P.\cos(\emptyset(nT)-\theta) - Q.\sin(\emptyset(nT)-\theta) \quad (3)$$

Q channel

$$r(t) = P.\sin(\emptyset(nT)-\theta) + Q.\cos(\emptyset(nT)-\theta) \quad (4)$$

where $\emptyset(nT)$ is the varying phase component due to the frequency offset of the carrier, θ is the initial phase offset of the system clocks with P and Q representing the channel information. If this symmetry did not exist then the cross channel components would be required to resolve the channel signals. From the final sub-sampled filter outputs the phase and quadrature samples are presented as addresses to the Decode PROM which is pre-programmed to translate them to a phase angle representation by a Tan -1 (.) function extended over a full 360 degrees. This output can be viewed as a rotating phasor, $\emptyset'(nT) = \emptyset(nT)-\theta$, or as a slope representing the rate of change of phase modulo 360 degrees, superimposed with the transmitted data information. From this information it is now possible to remove the data from the phasor and by the use of an Estimator algorithm the rotating phasor position can be estimated for the next incoming phasor plus data sample, fig 7. After subtraction of the phase estimate the data is removed to give an error value. Addition of the phase estimate $\emptyset p(nT)$ to this error value leaves the instantaneous phase offset from the presumed reference phase plus any phase noise. The Phase noise is of a bi-variate Gaussian distribution produced by the effect of the quadrature noise components fig 8. This signal is now filtered to reduce the noise variance on the signal leaving the instantaneous mean phase position. If the rate of change of phase is too great the the mean phase lags the incoming phase due to the delays introduced by the filter. This eventually results in the mean phase output aliasing, not tracking the incoming phase rotation and the system falls out of lock. This can be extended by incorporating into the estimator frequency information that can also be extracted from the Tan-1 PROM and projecting the instantaneous mean phase position to compensate for the high rate of change of phase i.e. frequency offset. If the amplitude information of the filter outputs is also extracted it is now possible to remove the rotating phase vector from the data phase information.This converts from a phasor amplitude representation back to a complex representation, and regenerate the correct P and Q channel data information with $\emptyset'(nT)$ = 0,90,180, 270 degrees, the usual 4 phase ambiguity for QPSK. This data is then available for soft decision decoding. A hard decision data symbol can also be extracted from the estimator circuit but due to the 4 phase ambiguity a set of 8 decoding maps are required (for QPSK) to obtain the correct demodulated data output. The error rate performance of this system has been tested in a back to back mode fig 9, without cycle slipping, over an operating frequency range of 12 KHz. (Cycle slipping is very important for data modems operating with FEC at low Eb/No values). Due to the digital nature of this system bandwidths can be easily adjusted to produce any trade-offs that are required to optimize the system for a particular operating point. Another advantage of this Estimator is that it acquires 'lock' within a very short time period of 32 symbol periods or less depending on the filtering requirements used. This system has been tested by simulation, back to back testing and by satellite trials.

Conclusions

The use of DSP in the satellite Data modem area can be used to improve the accuracy with which filters can be constructed to enable the use of zero ISI filter functions. In addition to the reduction of bandwidth many more advantages and savings can be found in:

1) System setup, lack of filter equalization by hand or machine, once designed accurate reproduction can be simply achieved.
2) The use of VLSI can reduce the modem costs.
3) Due to the use of Digital quadrature up and down conversion and a synchronous system both quadrature channels can be easily matched and phase quadrature maintained across the passband.
4) The complete system can maintain linear phase.
5) Data rates can be made switchable for the range 16kbits/sec to 256kbits/sec, for both QPSK and BPSK. A version that can operate from 50 bit/sec up to 4.096Mbits/sec QPSK is under development. These versions are being implemented using similar techniques but requires more parallel processing to take place at the higher rates.

Along with the advantages produced by the digital filtering the use of Phase estimation techniques to allow following of any phase reference errors between the carrier and the system local oscillator produces an increase in frequency range over which the system can operate compared with a more conventional approach. This also reduces the acquisition time of the system and is also flexible enough to allow optimum operating conditions to be set . It is envisaged that the modem will be micro-processor controlled to increase user friendliness and provide flexibility. VLSI custom chips will allow manufacturing problems to be eased, and costs reduced.

Acknowledgements.

The authors would like to thank Paul Smithson, Plymouth Polytechnic, for his considerable efforts in the design and construction of this modem. Thanks are also due to Roger Allan and Dave Saunders of Telemetrix Research for their support and co-operation in this joint project. The work has been guided and supported by John Everett of RSRE and the authors acknowledge his considerable contribution to the project.

References.

(1) Nyquist,H.," Certain Topics in Telegraph Transmission Theory ", Trans AIEE, Vol 47, pp 617-644 ,Feb 1928.
(2) Feher,K., Digital Communications,Prentice-Hall,1981
(3) Couch II, L.W, Digital and Analogue Communication Systems, Macmillan 1983.
(4) Proakis,J.G.,Digital Communications, McGraw-Hill, 1983.
(5) Lucky,R.W., J.Salz and E.J. Weldon, Principles of Data Communications, McGraw-Hill, 1968.
(6) Rabiner,L.R and B.Gold, Theory and Application of Digital Signal Processing, Prentice-Hall, 1975.
(7) Tomlinson,M and Bramwell,J.R. "Data Communications method and Apparatus", British Patent Application No. 8420380.

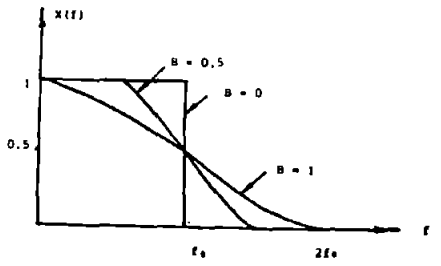


Fig 1 Frequency Response Shaping for Raised Cosine Filters.

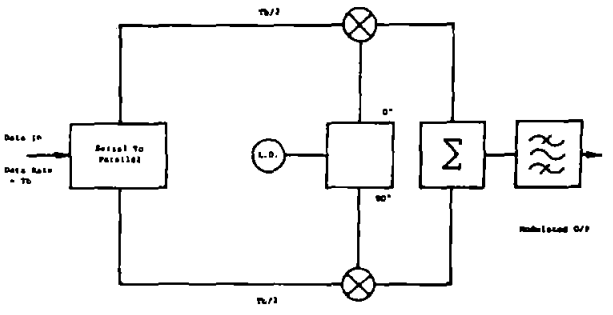


Fig 2 Basic QPSK Modulator for NRZ pulse Shaping

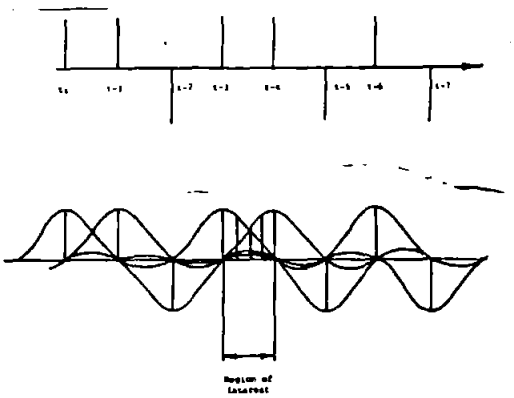


Fig 3 Impulse Response Summation

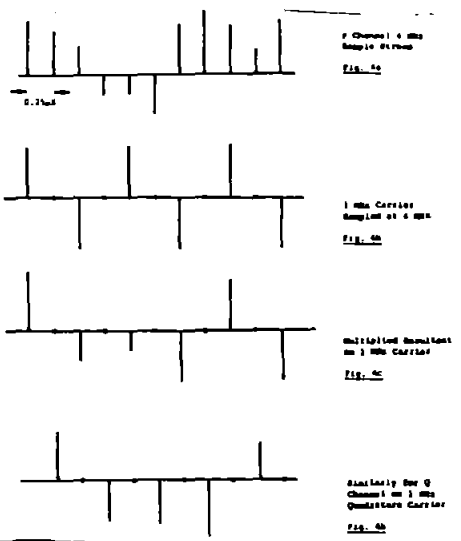


Fig 4 Frequency Conversion of Impulse Stream to Allow Channel Multiplexing

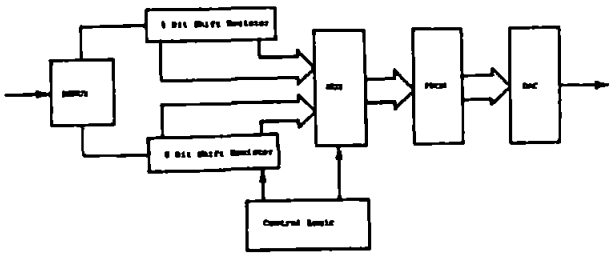


Fig 5 Digital QPSK/BPSK Modulator with Root Raised Cosine Filtering

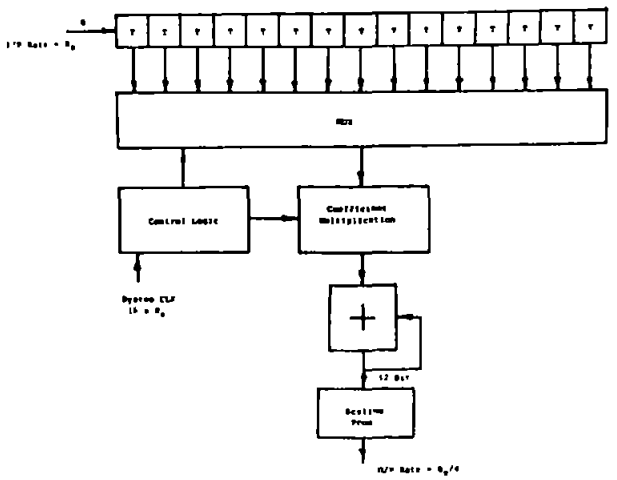


Fig 6 Basic Digital Receive Filter Architecture

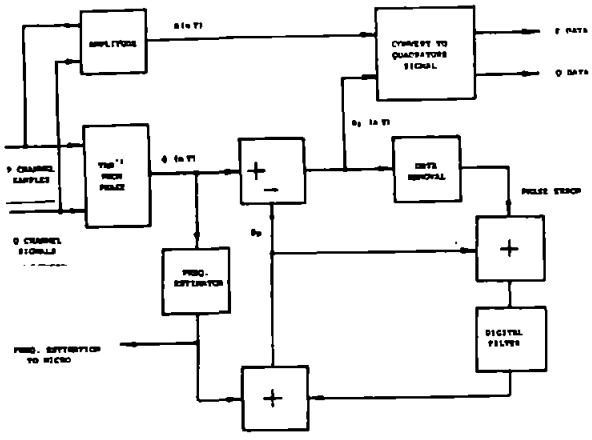


Fig 7 Simplified Phase Estimator Architecture

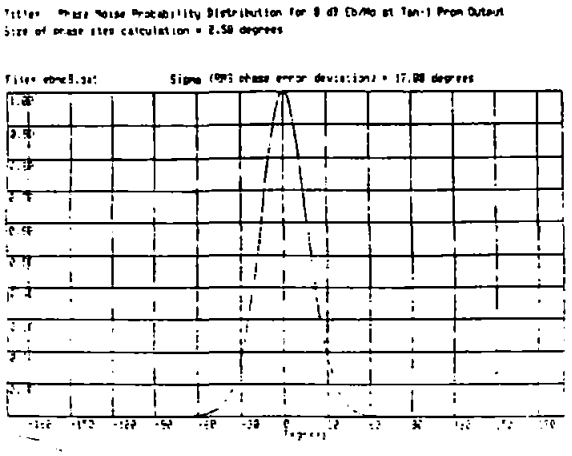


Fig 8 Phase Noise Probability Distribution from Tan-1 Prom

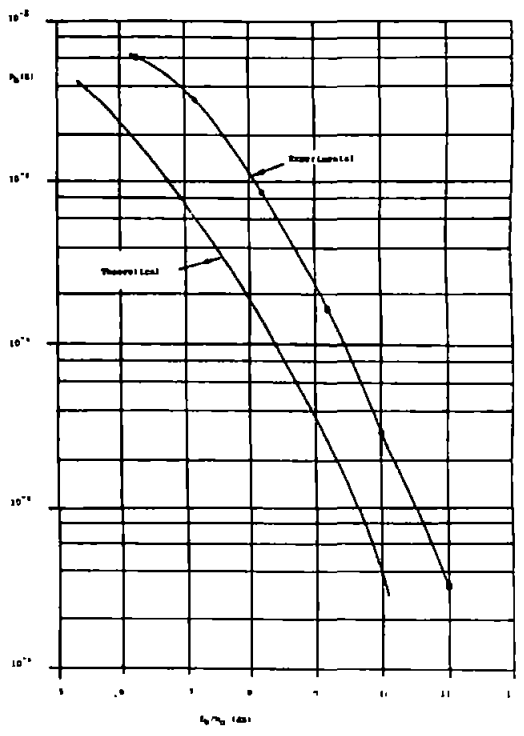


Fig 9 Bit Error rate Curve for BPSK at 256Kbit/sec

Appendix A

Extract from specification of Multi-Data rate Modem produced by the DTI (later to become the British National Space Centre) June 1984.

Electrical

Modulation	BPSK and QPSK (capability for Absolute and Differential Decoding).
Data Rates	32Kbits/sec to 512Kbits/sec QPSK. 16Kbits/sec to 256Kbits/sec BPSK.
Mode of operation	Continuous.
Channel Spacing	1.41 x data rate
Channel Filter Response	The overall transmit and receive filter should be that of a 30% Raised Cosine Filter amplitude equalized for reasonably flat group delay. The Transmit channel filters shall be such as to produce a transmit energy mask as shown in figure A.1: the receive channel filter shall have the characteristics as shown in figure A.2. Group delay per symbol period for the received filter shall be as shown in figure A.3.

Cycle Skipping Performance	When operating at 6.6 dB Eb/No the observed cycle slip rate shall not exceed 1 slip per minute.
Bit Timing Integrity	Bit Error Rate due to clock slip events must be negligible compared to BER due to received noise.
Back to Back Performance	Eb/No for a given BER to be within 1.3 dB of Theoretical assuming perfect Nyquist filtering (for BER 10^{-3} to 10^{-10}).
Transmit Frequency	In the Range 52 to 88 MHz.
Receive Frequency	In the Range 52 to 88 MHz
Frequency uncertainty	± 25 KHz
Acquisition Time	< 1sec
Lock Acquisition	Must acquire lock for Eb/No >4 dB.

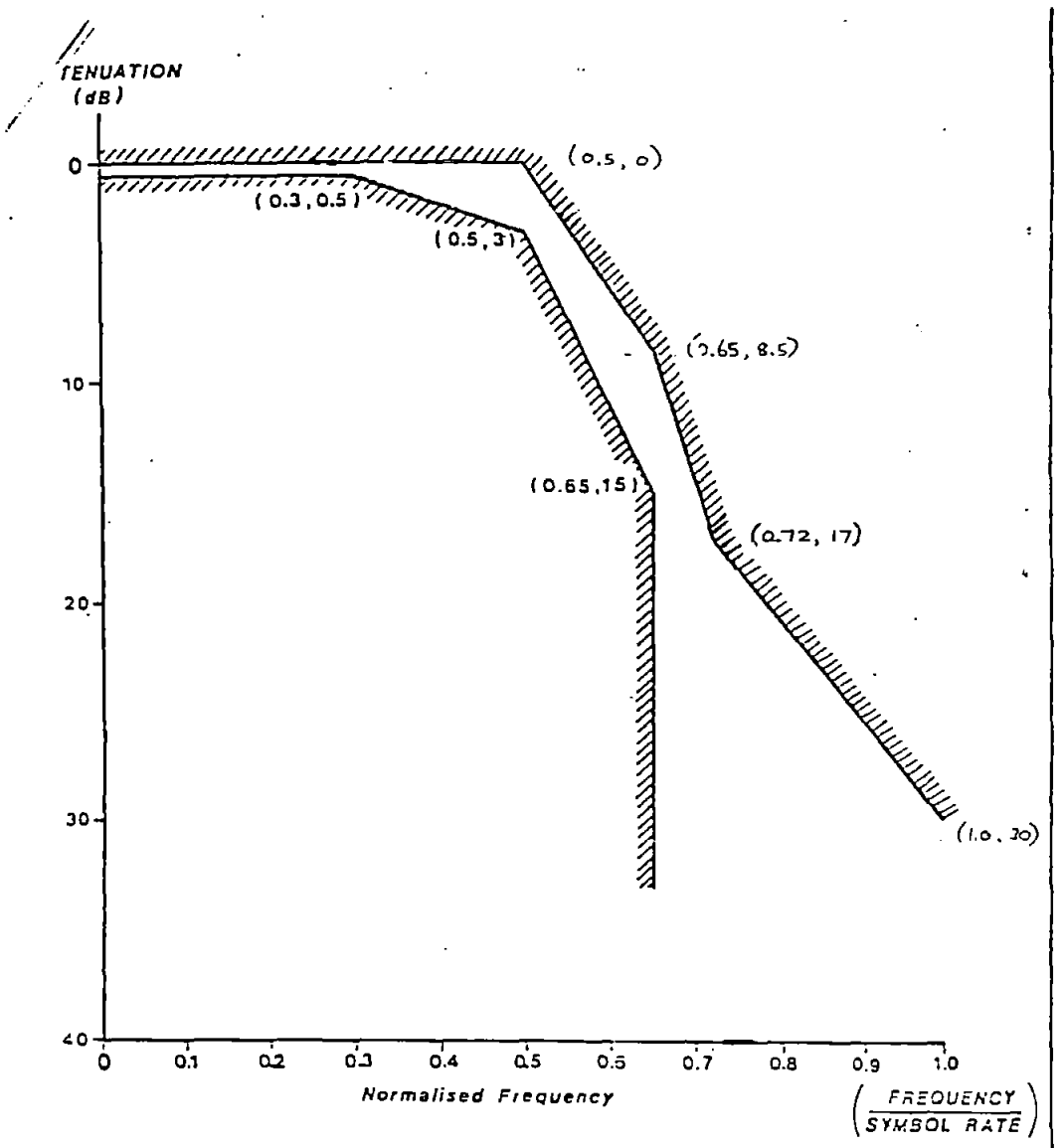


Fig A.1 Transmit Filter Mask.

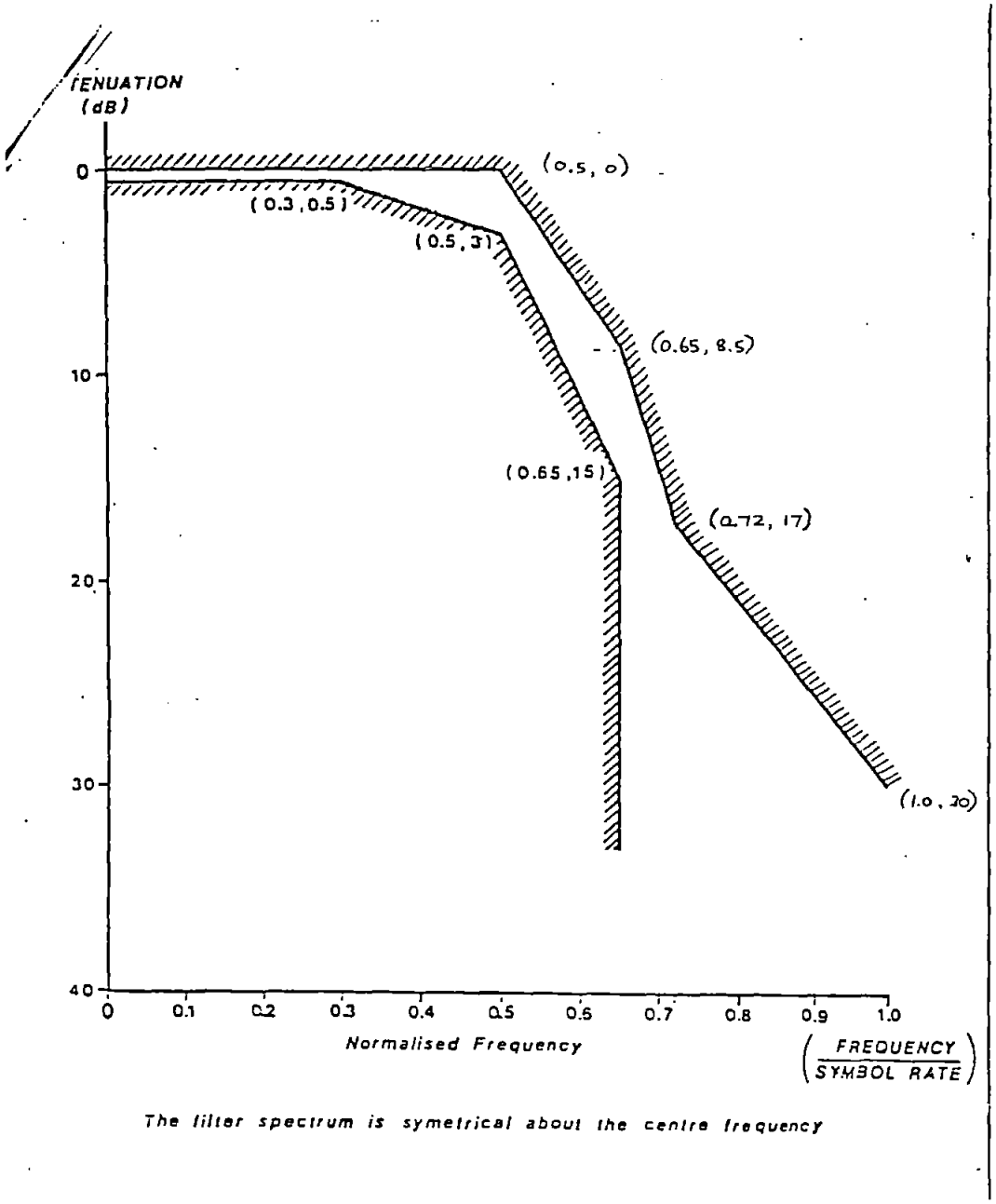


Fig A.2 Received Filter Mask.

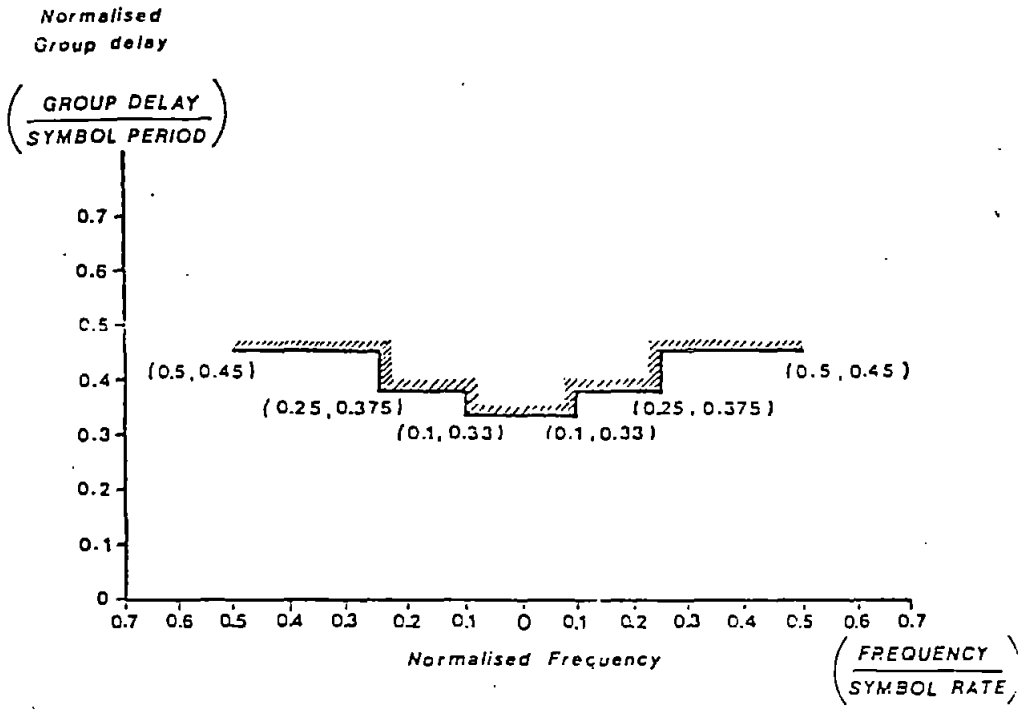


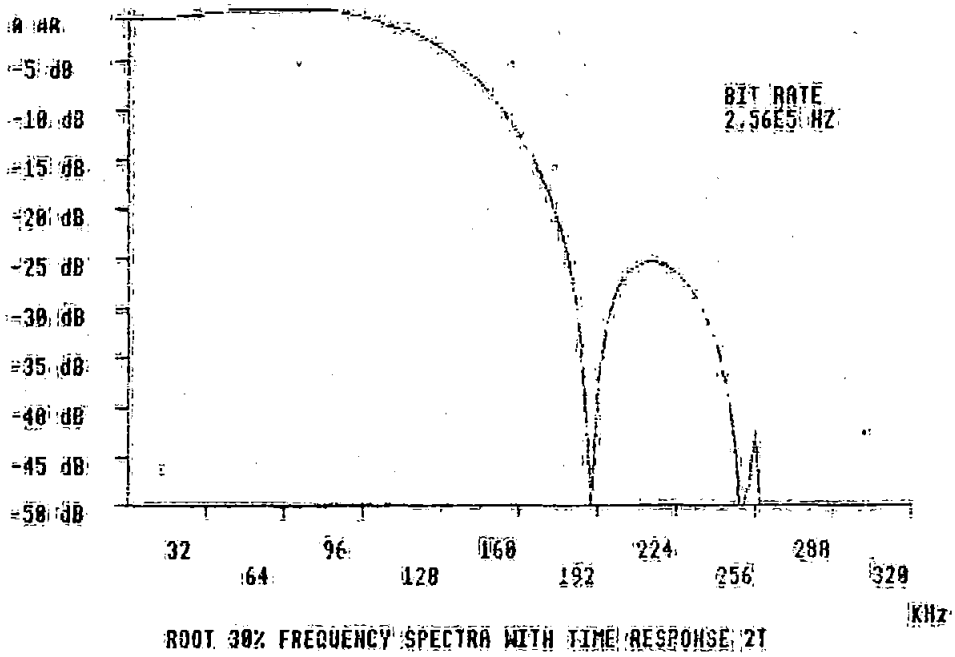
Fig A.3 Received Filter Group Delay per Symbol Period

Appendix B

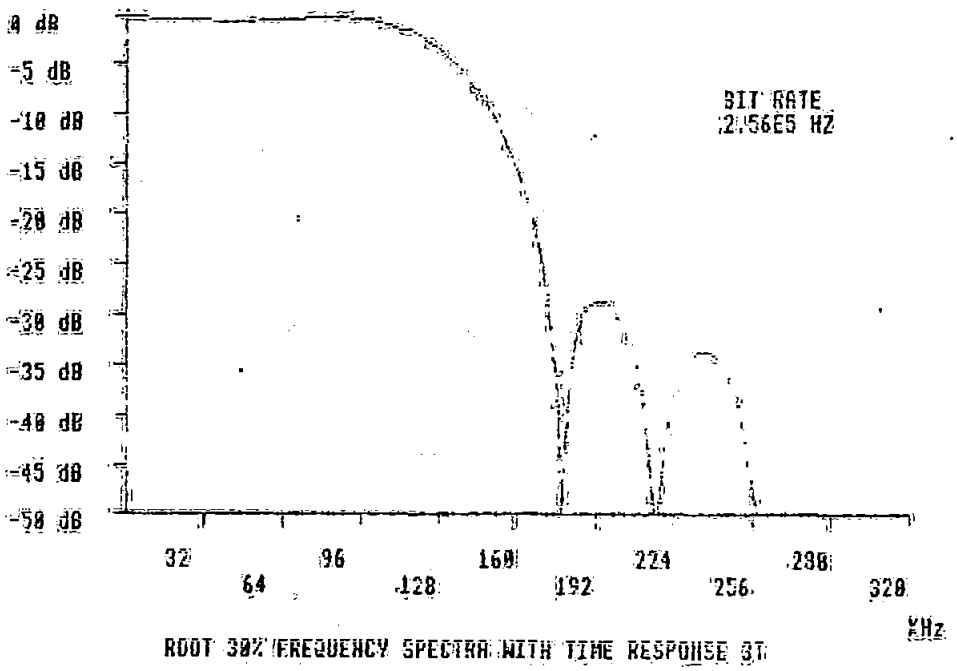
The Truncation of Root 30% Raised Cosine Impulse Response and it Effect in the Frequency Domain

In the design of the Raised Cosine filters a number of factors can be traded to obtain the most elegant and convenient solution. These factors include the choice of roll-off factor, truncation length for the impulse response, size of filters required in terms of delay sections or Proms etc. Careful choice is required. For example in the transmitter the length of the impulse response is constrained by the size and access times and economics of available Proms. The initial specification called for Root 30% raised cosine filters at the transmitter and receiver with a 30dB stopband at 1.4 times the symbol rate hence the final filter arrangement must conform to these conditions. For the required number of data rates to be implemented with a 4.096MHz sampling frequency the size of prom required is 16Kbytes. This can be economically implemented in 8Kbyte 2764 EPROMs but this constrains the impulse response length to 6 symbol periods (double sided), but is this enough to fulfil the design specification?

If an analysis is undertaken on a root 30% Raised Cosine impulse response it can be seen in fig B.1 that if the response is reduced to 2 symbol periods (single sided) the the frequency response obtained will have sidelobes to a level of -26dB. If the truncation is 3 symbol periods the sidelobe level is -29dB and if 4 symbol period the level is -32dB. The 3 symbol period truncation was chosen to be close match to the desired response for the research model without incurring the extra cost of the 4 symbol system.

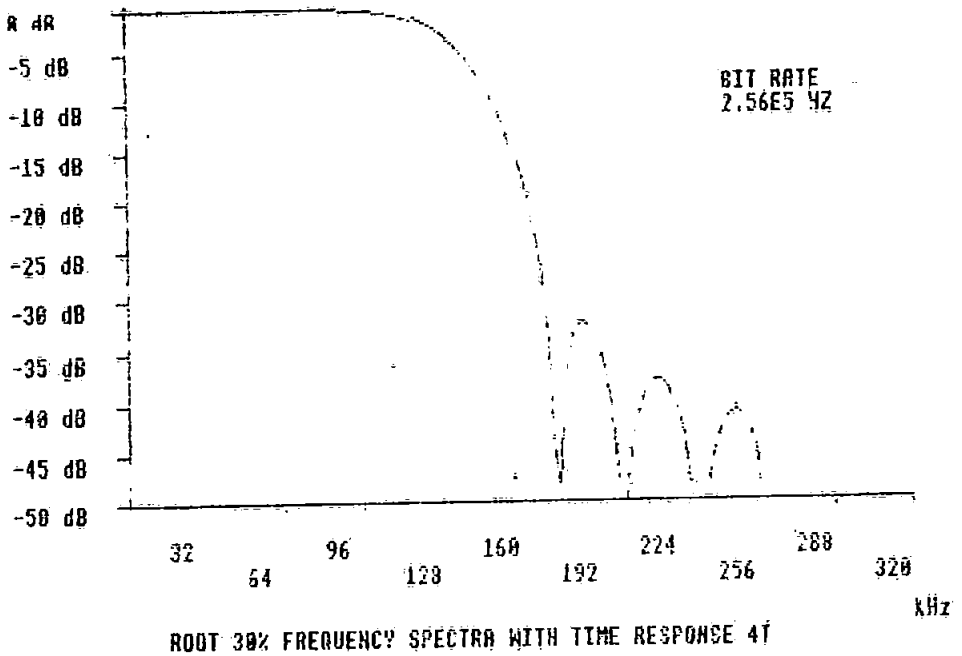


ROOT 30% FREQUENCY SPECTRA WITH TIME RESPONSE 2T

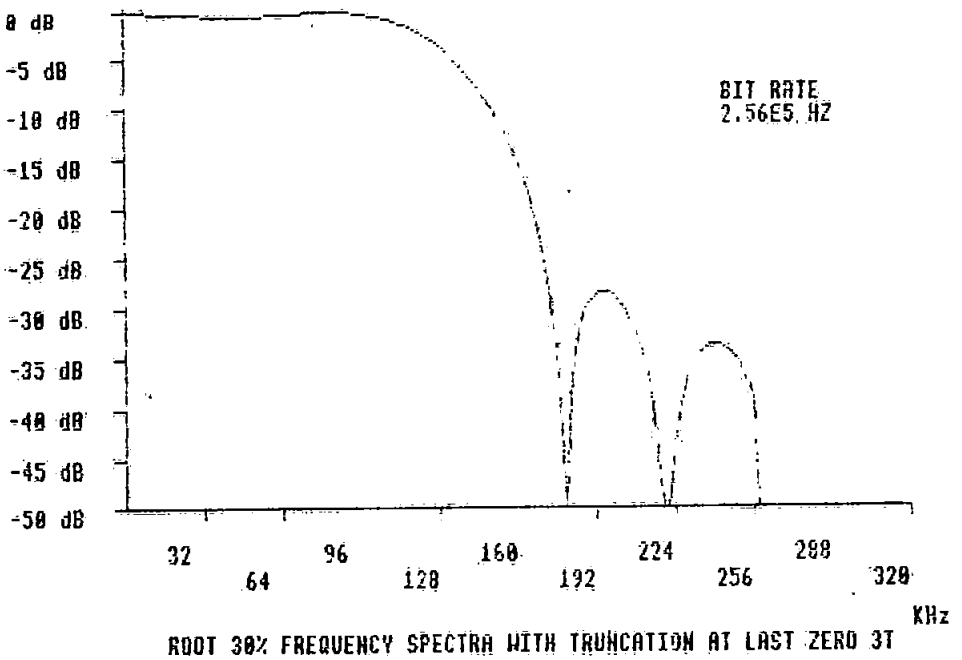


ROOT 30% FREQUENCY SPECTRA WITH TIME RESPONSE 3T

Fig B.1 Variations in Sidelobe Performance due to Impulse Response Truncation.



ROOT 30% FREQUENCY SPECTRA WITH TIME RESPONSE 4T



ROOT 30% FREQUENCY SPECTRA WITH TRUNCATION AT LAST ZERO 3T

Fig B.1 Variations in Sidelobe Performance due to Impulse Response Truncation.

Appendix C

Infinite Impulse Response Single Pole Filter

The single pole filter can be defined by the equation:-

$$V_o = V_{in} + K.V_o.Z^{-1} \qquad \text{(eq C.1)}$$

where V_o = output ,V_{in} = input, K= feedback coefficient, Z is the Z transform.

Therefore

$$\frac{V_o}{V_{in}} = \frac{1}{1 - K.Z^{-1}} \qquad \text{(eq C.2)}$$

now $Z^{-1} = e^{-jwT}$

and eq C.2 becomes

$$\frac{V_o}{V_{in}} = \frac{1}{1 - K.\text{Cos } wT + jK.\text{Sin } wT} \qquad \text{(eq C.3)}$$

Hence the magnitude response is given by:-

$$\left| \frac{V_o}{V_{in}} \right|^2 = \frac{1}{1 - 2.K.\text{Cos } wT + K^2} \qquad \text{(eq C.4)}$$

and the phase difference output to input θ_{lag} is given by:-

$$\theta_{lag} = \tan^{-1} \left[\frac{- K.\text{Sin } wT}{1 - K.\text{Cos } wT} \right] \qquad \text{(eq C.5)}$$

if $w << 1/T$, $wT<<1$, Sin wT tends to wT, and Cos wT tends to 1

Therefore

$$\theta_{lag} = \tan^{-1}\left[\frac{-K.wT}{1-K}\right] \qquad \text{(eq C.6)}$$

and $\tan^{-1}(.)$ tends to $(.)$

hence

$$\theta_{lag} \text{ tends to } \left[\frac{-K.wT}{1-K}\right] \qquad \text{(eq C.7)}$$

now $\theta_{lag} = w.T_d$ where T_d = Time delay through filter

and hence approximates to:

$$T_d = \left[\frac{-K.T}{1-K}\right] \qquad \text{(eq C.8)}$$

hence for a coefficient of K= 15/16

$$T_d = -15.T$$

Appendix D

This appendix contains program listings for the programs used in the phase estimator simulation. The programs were written in Pascal and were compiled using The Propero Pascal Compiler.

Program 1 Phase Deviation Due to AWGN

This program generates quadrature noise components for a normalised sigma deviation. The resulting noise PDF is of a bi-variate distribution and is stored as a Histogram with a selected phase cell width. The resulting histogram can be used to calculate the probability of a particular phase error due to AWGN occuring and hence the error function integrations. The following pages contain the program listing for Noise1-1.pas with appropriate comments. Due to the naming conventions used, whereby a meaningfull name is used for a function or procedure, the use of flow charts and line by line commenting is considered superfluous.

NOISE1-1.pas

```
phase_noise_statistics(input,output);
{ calculate statistical distribution and probabilities of
phase errors}

                label 10,20;

                const pi=3.141592653589793;
                      maxloc=360;
                      vers=1.12;
                type historarray=array[-maxloc..maxloc] of
                                 longreal;
                     intarray=array[-maxloc..maxloc] of integer;
```

```
                    real=longreal;
                    choice=0..5;

               var ebno,noisestep,stepsize:real;
                   quit:string;

function antilog(x:real):real;
{calculates 10^x }

begin
    antilog:= exp(x*2.302585093);{ 2.3025= ln 10}
end;

function  prob(x,propc:real):real;
{gaussian probability of x, scale for unit area by propc}
         var y:real;

begin
     y:= -1*sqr(x)/2; {sigmasquared=1}
     prob:=propc*exp(y)/sqrt(2*pi);
end;

procedure quitread(var quit:string);
{sets quit to yes to leave menu area}

begin
      quit:='yes'
end;

procedure proportionalConstant(var propc,stepsize,
                                   noisestep:real);

{calculates constant such that area under calculated curve = 1}

             var x,prob,y,sum:real;
                 n,i:integer;
```

```
begin
     n:=round(5/noisestep);
     sum:=0;
     for i:= -n to n do
       begin
           x:=i*noisestep;
           y:= (-1*sqr(x))/2; {sigmasquared =1}
           prob:= exp(y)/sqrt(2*pi);
           sum:=prob+sum;
       end;
     propc:= 1/sum;
     writeln('propc =',propc);
end;



procedure EboverNo(var rmsEbNo,ebno:real);

{ requests Eb/No ratio in dBs and converts to rms}

          var x:real;
begin
     x:= ebno/10;
     x:=antilog(x);
     x:=x*2;{ split into inphase and quadrature components
                                                  keeping No =1}
     rmsEbNo:=sqrt(x);
end;

procedure clearhist(var thetahist:historarray;
                    var cellcount:intarray);

{clears historgram memory area ready for start of calculation}

         var h:integer;
```

```
begin
     for h:=-maxloc to maxloc do
      begin
       thetahist[h]:=0;
       cellcount[h]:=0;
      end;
end;
procedure add_to_hist(etheta:longreal;probtheta:longreal;
                      stepsize:real;var thetahist:historarray;
                      var cellcount:intarray;
                      var listends:integer);

 {calculates historgram for radian stepsize of probability of
phase deviation}

         var number:integer;
             newprob,oldprob:longreal;

begin
     number:= round(etheta/stepsize);
     cellcount[number]:=cellcount[number]+1;
                           {increment cell count of cell number

     oldprob:= thetahist[number];
     newprob:= oldprob+probtheta;
     thetahist[number]:=newprob;
     if abs(number)>listends then listends:=number;
end;


procedure thetaerror(var thetahist:historarray;
                     var cellcount:intarray; rmsEbNo,propc,
                     stepsize,noisestep:real;
                     var listends:integer);

{calculate array of noise samples  for all values of  noise
variations}
```

```
            var n,i,j,location:integer;
                xi,xj,piby2,pmag,qmag,quadshift,
                probsum,probtheta:real;
                etheta:longreal;

begin
     n:= round(sqr(2*(5/noisestep)+1));
     writeln( 'Calculating ',n:7,' noise samples');
     piby2:=pi/2;
      n:=round(5/noisestep);
      probsum:=0;
      location:=0;
      for i:=-n to n do
       begin
          xi:= i*noisestep;
          for j:= -n to n do
           begin
            xj:=j*noisestep;
                 pmag:=rmsEbNo+ xi;
                 qmag:= xj;
                 if pmag<>0 then
                 begin
                  etheta:=arctan(qmag/pmag);
                  if (pmag<0) and (qmag>0) then
                     etheta:=pi+etheta;
                  if (pmag<0) and (qmag=0) then etheta:=pi;
                  if (pmag<0) and (qmag<0) then
                  etheta:=-pi+etheta;
                 end
                 else
                  begin
                   if qmag=0 then etheta:=0;
                   if qmag>0 then etheta:=piby2;
                   if qmag<0 then etheta:= -piby2;
                  end;
```

```
            probtheta:= prob(i*noisestep,propc)
                        * prob(j*noisestep,propc);
            add_to_hist(etheta,probtheta,stepsize,
                        thetahist,cellcount,listends);
            probsum:=probsum+probtheta;
            location:=location+1;
          end;
      end;
      writeln('probsum= ',probsum:10:5);
      listends:=listends-1;
end;
procedure averagephasevariance(thetahist:historarray;
                               listends:integer;
                               var sigma:real;stepsize:real);

{ calculates average phase variance from sum of
p(i)*error^2}

          var h:integer;
              etheta:real;
              sum:longreal;

begin
     sum:=0;
     for h:= -listends to listends do
       begin
         etheta:= h*stepsize;
         sum:= thetahist[h]*sqr(etheta)+sum;
       end;
     sigma:= sqrt(sum)*180/pi;
     writeln('RMS phase deviation = ',sigma:10:6,'
              degrees'); end;

procedure savedisc(thetahist:historarray;
                   cellcount:intarray;
                   listends:integer;
```

```
                    sigma,stepsize:real);

{ write data to disc as a text file}

       label 10;
       var outfile1:text;
           h:integer;
           sum:real;
           name:string;

begin
    writeln('entered save to disc');
     sum:=0;
  10:writeln('Name of file ( max 8 characters.dat)');
     readln(name);
     if length(name)>12 then goto 10;
     stepsize:=stepsize*180/pi;{convert to degrees}
     assign(outfile1,name);
     rewrite(outfile1);
     writeln(outfile1,listends,stepsize,sigma);
     for h:= -listends to listends do
       begin
       sum:=sum+thetahist[h];
         writeln(outfile1,thetahist[h]);
       end;
       writeln('Sum of Probabilities under curve = ',
               sum:6:4);
end;


procedure write_to_disc(thetahist:historarray;
                        cellcount:intarray;
                        listends:integer;
                        sigma,stepsize:real);
{ question of writing to disc statistical phasenoise data}
```

```
        label 10,20;
        var ch:char;

begin
     10 : writeln('Save phase statistics to disc? (y/n) ');
          readln(ch);
          case ch of
           'y':savedisc(thetahist,cellcount,listends,
                        sigma,stepsize);
            'Y':savedisc(thetahist,cellcount,listends,
                        sigma,stepsize);
            'n':goto 20;
            'N':goto 20;
            otherwise goto 10;
          end;
20:end;

procedure erfc(listends:integer;stepsize:real;
               thetahist:historarray);

{produces complimentary error  function i.e probability  of
error being greater than x}

         var h:integer;
             sum,etheta,x,xdeg:real;

begin
     stepsize:=stepsize*pi/180;{converts to rads}
     writeln('Enter prob(phase error >x degrees)');
     readln(xdeg);
     x:=xdeg*pi/180;
     sum:=0;{calculates complimentary errror function}
     for h:= -listends to listends do
      begin
       etheta:= h*stepsize;
       if abs(etheta)>=x then sum:= sum+thetahist[h];
```

```
                          {probability of etheta>x radians}
      end;
      writeln('probability of phase error >',xdeg:4:2,'
               degs = ',sum:15:12);
end;

procedure setnoisepower(var ebno:real);

begin
     write('Noise power required in dB ');
     readln(ebno);
end;

procedure setnoisestep(var noisestep:real);

         label 10;

begin
10: write('Noisestep size required (include leading zeros)
           in sigma ');
    readln(noisestep);
    if (noisestep<0) or (noisestep>5) then goto 10;
end;

procedure sethiststep(var stepsize:real);

          label 10;

begin
10:  write('Cell width stepsize in degrees (min =0.5 deg )
           ?');
    readln(stepsize);
    if stepsize<0.5 then goto 10;
    if stepsize>10 then
      writeln('Stepsize too large for accuracy');
end;
```

```
procedure computenoisestat(ebno,noisesteps,stepsize:real;
                           var listends:integer;
                           var      thetahist:historarray);
{calculates Phase noise statistics in this procedure}

              var x,propc,sigma,probability,rmsEbNo:real;
                  h,n:integer;
                  cellcount:intarray;



begin
     stepsize:= stepsize*pi/180;{ converts to radians}
     proportionalConstant(propc,stepsize,noisestep);
     EboverNo(rmsEbNo,ebno);
     clearhist(thetahist,cellcount);
     thetaerror(thetahist,cellcount,rmsEbNo,propc,stepsize,
                noisestep,listends);
     averagephasevariance(thetahist,listends,sigma,
                          stepsize);
     write_to_disc(thetahist,cellcount,listends,sigma,
                          stepsize);
end;


procedure computegaussian(ebno,stepsize:real;
                          var listends:integer;
                          var      thetahist:historarray);
{Caluculates Gaussian historgram in this procedure}

     label 10,20,30;

     var scale,sum,y,sigma,
         rmsEbNo,maxangle,etheta:real;
         h:integer;
```

```
        ch:char;
        cellcount:intarray;{dummy}
begin
     sum:=1;
     clearhist(thetahist,cellcount);
     EboverNo(rmsEbNo,ebno);
     write('Enter value of sigma in degrees for which plot
           required ? ');
     readln(sigma); 30:  write('Comparison plot ? (y/n) ');
     readln(ch);
     case ch of
     'y': goto 20;
     'Y': goto 20;
     'N':goto 10;
     'n':goto 10;
     otherwise goto 30;
     end;
10: write('Enter max phase angle to which plot is required
          ?');
    readln(maxangle);
    listends:=round(maxangle/stepsize);
20: scale:=1/sum;
    sum:=0;
    for h:=-listends to listends do
     begin
         etheta:=h*stepsize/sigma;
         y:=(-1*sqr(etheta))/2;
         thetahist[h]:=scale* 0.3989422*exp(y);
         sum:=sum+thetahist[h];
     end;
     if (sum>1.01) or (sum<0.99) then goto 20;
     write_to_disc(thetahist,cellcount,listends,sigma,
                   stepsize);
end;

procedure plot;
```

```
{calls Plot program }

begin
     chain('plotstat');
end;


procedure menu(var ebno,noisestep,stepsize:real;
               var quit:string);
{sets up menu for desired set of statistics}

   label 10,20;
   var value:choice;
       listends:integer;
       thetahist:historarray;

begin

    listends:=0;
10: writeln;
    writeln('************************************');
    writeln('         PHASE NOISE PROBABILITY    ');
    writeln('            version ',vers:6:2);
    writeln('************************************');
    writeln;
    writeln('1) Noise power is  ',EbNo:6:1 ,' dB');
    writeln;
    writeln('2) Noise stepsize is ',noisestep:6:3,
             ' Sigma');
    writeln;
    writeln('3) Historgram cell width ',stepsize:6:3,
            'Degrees');
    writeln;
    writeln('4) Compute Phase noise statistics');
    writeln;
    writeln('5) Compute Gaussian Distribution');
```

```
    writeln;
    writeln('6) Call Plot Program');
    writeln;
    writeln('7) Calculate ERFC ');
    writeln;
    writeln('0) End');
    writeln;
    write('ENTER ');
    readln(value);
    case value of
    0: quitread(quit);
    1: setnoisepower(ebno);
    2: setnoisestep(noisestep);
    3: sethiststep(stepsize);
    4: computenoisestat(ebno,noisestep,stepsize,listends,
                        thetahist);
    5: computegaussian(ebno,stepsize,listends,thetahist);
    6: plot;
    7: erfc(listends,stepsize,thetahist);
    otherwise goto 10;
    end;
    if quit='yes' then goto 20;
    goto 10;
  20:end;

procedure loaddefault(var ebno,noisestep,stepsize:real);

{loads default setting into menu }

begin
     ebno:= 0;
     noisestep:=0.03;
     stepsize:=0.5;
end;

{**** Main program starts here ****** }
```

```
begin {main program}
   quit:='no';
   loaddefault(ebno,noisestep,stepsize);
10:menu(ebno,noisestep,stepsize,quit);
   if quit='yes'then  goto 20;
   goto 10;
20:end.
```

Phase Estimator Simulation Program

The phase estimator simulation program inputs noise statistic data from a series of files created by Noise1-1.pas for different Eb/No values. This values are take for integer steps in SNR. The program then simulates the behaviour of the phase estimator algorithm for a given set of variables which can be selected via a set of menus. The resultant simulation output is stored to disc, providing records of the setup conditions and the performance evaluation for post processing. The following pages contain the program listing for Est1-18.pas which was used in the evaluation of the algorithm. As before flow diagrams and line by line commenting is considered superfluous.

EST1-18.pas

```
program Estimator(input,output);

{program allows reading of statistical data from files of
different Eb/No. This program is for the analysis of the
phase estimator using an IIR filter with single order
coefficient selectable from , 15/16,31/32,63/64,127/128 .
The Frequency information is now obtained outside the loop
of the estimator and feeds forward to the predicted phase
mean }

      label 10,20;
```

```
const maxint=4194304;
      pi=3.14159263;
      nmax=128;
      m=256;
      offset=0;
      ver=1.18;

type sample=  array[0..nmax] of real;
     intsample=array[0..nmax] of integer;
     intresults= array [0..nmax] of integer;
     intarray= array[-360..360] of integer;
     rlresults= array [0..nmax] of real;
     delayspace=array[0..32] of integer;
    samplearray= array[-360..360] of real;
    weightingarray=array [0..721,0..1] of real ;
{ could be too small if noise stepsize<0.5 degrees}
     choice=1..6 ;
     timeseconds=0..60;
     results=record
               trap:string;
               elapsedarray:rlresults;
               anglearray:rlresults;
               binsample:intsample;
               datain:intsample;
               noiseimpulsearray:intresults;
               noiseyloarray:intresults;
               meanphasearray:intresults;
               projectslopearray:intresults;
               predphasearray:intresults;
               noise:sample;
               cerror:sample;
               dataoutarray:intresults;
               end;
     menurecord= record
                   ebno,esno:integer;
                   trapsprung:boolean;
```

```
                    trapcount:integer;
                    errate:real;
                    errcount:integer;
                    sigma:real;
                    fo:real;
                    fs:real;
                    runtime:real;
                    mag:real;
                    quantbits:integer;
                    datatype:string;
                    dstate:string;
                    nstate:string;
                    filter:string;
                    slope:string;
                    mean:string;
                    slopecoef:string;
                    meancoef:string;
                    coefinc:string;
                    bits:integer;
                    refstepsize:real;
                    reflistends:integer;
                    auto:string;
                  end;

statrecord= record
              errormean:real;
              errordev:real;
              carriermean:real;
              carrierdev:real;
              end;


var value:choice;
    etheta,sigma,stepsize,maxy:real;
    ycord:samplearray;
    noise:sample;
```

```
            esno,h,listends:integer;
            name,quit,sesno:string;
            readbefore:boolean;
            waittime:timeseconds;
            setup:menurecord;

function rand:real;                                    external;
function cstat:boolean;                                external;
procedure time(var hours,minutes,seconds,
                        hundreths:integer);            external;

function convertdegtoqnt(x:integer):integer;
{converts degrees to quantized level number}

begin
     convertdegtoqnt:=x*m div 360;
end;

procedure wait(var waittime:timeseconds);
{waits for a set time period before returning}

        var hours,minutes,seconds,hundreths,
            starttime:integer;

begin
     time(hours,minutes,seconds,hundreths);
     starttime:=seconds;
     repeat
     time(hours,minutes,seconds,hundreths);
     until seconds>starttime+waittime;
end;

procedure quitread(var quit:string);
{sets quit variable to yes}

 begin
```

```
     quit:='yes';
 end;

procedure keypressed(var quit:string);
{sets acknowlegdement if key pressed}
       label 10,20;
       var value:choice;

begin
    quit:='no';
 10:writeln(' Key pressed :-');
    writeln;
    writeln(' 1) Continue ');
    writeln;
    writeln(' 0) Quit ');
    writeln;
    write(' ENTER ');
    readln(value);
    case value of
     0:quitread(quit);
     1:goto 20;
    otherwise goto 10;
    end;
20:end;
procedure clearcellcount(var cellcount:intarray);
{clears histogram array cells}

   var h:integer;

begin
     for h:=-360 to 360 do
     begin
      cellcount[h]:=0;
     end;
end;
```

```
procedure disksave(setup:menurecord;statresult:statrecord;
                   phaseresult:results);
{saves setup and results to disk}

        label 10;

        var filename:string;
            estimatorfile:text;
            h:integer;
begin
     str(setup.trapcount,filename);
     if setup.trapcount>99 then filename:='ZZ';
     filename:=concat('a:phdata',filename,'.dat');
     assign(estimatorfile,filename);
     rewrite(estimatorfile);
     writeln(estimatorfile,
         'Eb/No = ',setup.ebno:6,' dB,',
         ' Noise SD = ',setup.sigma:6:2,' degrees,',
         ' Add Noise ',setup.nstate,',',
         ' Add Data ',setup.dstate,'-',setup.datatype);
     writeln(estimatorfile,
        'Frequency offset ',setup.fo:6:3,' Hz,',
        ' Symbol rate = ',setup.fs:6:1,' KHz,',
        ' Filter is ',setup.filter);
     writeln(estimatorfile,
        'Sim time = ',setup.runtime:8:6,'sec,',
        ' Quantizing Bits = ',setup.quantbits,',',
       ' Magnitude of signal = ',setup.mag);
     writeln(estimatorfile,
       'Error count = ',setup.errcount,',',
       ' Error rate inclusive of this data block ',
                                  setup.errate:8:6);
     writeln(estimatorfile,'Type of Trap is ',
             phaseresult.trap);
     writeln(estimatorfile,'Mean is ',setup.mean,
                 ' Coefficient = ', setup.meancoef);
```

```
    writeln(estimatorfile,'Slope is ',setup.slope,
                  'Coefficient = ', setup.slopecoef);
    writeln(estimatorfile,'No. of Bit in IIR feedback is '
           , setup.bits);
    writeln(estimatorfile);
    writeln(estimatorfile,'Elapsed':8,'Data':5,'angle':6,
           'bin':6,'noise':5,'offsetP':8,'mean':5,
           'slope':6,'thetaP':7,'ModO/p':7,'C/Est ':7,
                                                'Data':6);
    writeln(estimatorfile,'sec':8,'degs':5,'degs':6,
           'qnt':5,'deg':5,'qnt':8,'qnt':5,'qnt':6,
           'qnt':7, 'qnt':7,'degs':6);
     for h:=0 to nmax do
      begin
       writeln(estimatorfile,
       phaseresult.elapsedarray[h]:8:6,
       phaseresult.datain[h]:5,
       phaseresult.anglearray[h]:6:1,
       phaseresult.binsample[h]:5,
       phaseresult.noiseimpulsearray[h]:5,
       phaseresult.noiseyloarray[h]:8,
       phaseresult.meanphasearray[h]:5,
       phaseresult.projectslopearray[h]:6,
       phaseresult.predphasearray[h]:7,
       phaseresult.noise[h]:7:1,
       phaseresult.cerror[h]:6:1,
       phaseresult.dataoutarray[h]:6);
      end;
10: close(estimatorfile);
end;

procedure  aver(n:integer;ransample:sample;var  mean:real);
{calulates mean of integer samples}

          var h:integer;
              sum:real;
```

```
begin
     sum:=0;
      for h:= 0 to nmax do
        begin
         sum:= sum+ ransample[h];
        end;
      mean:=sum/(nmax+1);
end;



procedure variance(randsample:sample;var oldmean,mean,
                   sd:real);
{calculates variance of random input samples}

          var h,n:integer;
              sum,sigmasqrd,mean1:real;

begin
       n:=nmax;
       aver(n,randsample,mean);
       sum:=0;   mean1:=oldmean;{uses current mean value}
       for h:=0 to nmax do
         begin
             sum:= sqr(randsample[h]-mean1)+sum;
         end;
      sigmasqrd:= sum/(nmax);
      sd:= sqrt(sigmasqrd);
end;
{**********************phase lock procedures************}

{These procedures are all used in the phase estimator
simulation }

function convert2scom(y, x:integer):integer;
{Converts sign magnitude to 2's complement notation}
```

```
begin
     if x<0 then
        convert2scom:= y+x
     else
        convert2scom:=x;
end;


function inverse2s(y, x:integer):integer;
{converts from 2's complement to sign magnitude}

begin
     if x<0 then writeln('ERROR in Inverse 2s complement');
     if x>=y div 2 then
        inverse2s:=x-y
     else
        inverse2s:=x;
end;

function subcyclic( augend,addend:integer):integer;
{performs 2s compliment subtraction on a circular basis
ignoring overflows}

begin
     subcyclic:= (augend+(m-addend)) mod m;
end;

function addcyclic( augend,addend:integer):integer;
{performs 2's complement addition on a circular basis
ignoring overflows}

begin
     addcyclic:=(augend+addend) mod m;
end;

function power(x,y:real):real;
```

```
{calculates x to power of y}

begin
     power:= exp(y*ln(x));
end;

function log(x:real):real;

{calculates log to base 10}

begin
     log:=ln(x)/ln(10);
end;


function divide2scom(y,x,denom:integer):integer;
{divides 2's complement number by denom}

begin
         x:=inverse2s(y,x);
         x:=x div denom;
         divide2scom:=convert2scom(y,x);
end;



procedure initializeloop(var startphase:real;
                         var predphase,avl,adl,dt,accum,
                             slopaccuminus1:integer;
                         var averphase,averslope,
                             slopedly:delayspace);

{initialize arrays and predicted phase}

          var h:integer;
```

```
begin
     startphase:=0;
     predphase:=0; {initial phase prediction value}
     accum:=0; {initial value of average accumulator}
     avl:=16;{ set length of averaging circuit}
     dt:=8 ;{set slope detect over 8 symbols}
     adl:=16;{ set slope averager to 16 symbols}
     slopaccuminus1:=0;{set slope average accumulator }
     for h:=0 to avl do {flush out to initialize averaging
                         space}
       begin
       averphase[h]:=0;
      end;
      for h:= 0 to adl do
                  { flush out averaging space for slope
                    detector}
       begin
         averslope[h]:=0;
       end;
       for h:= 0 to dt do
                  { flush out delay space for slope
                    detector}
        begin
          slopedly[h]:=0;
        end;
end;

procedure greaterthan180(var x:real);
{if x greater than 180 degrees converts to negative value}

begin
      if x<-180 then x:=360+x;
      if x>180 then x:=x-360;
end;
```

```
procedure modulus( s,y:real;var ans:real);
{produces modulo arithmetic}
          var multi:real;

begin
     multi:=trunc(s/y);
     ans:= s - (multi*y);
end;



procedure shiftintarray(var inputarray:delayspace;
                              length:integer);
{shifts data through array space}

          var h:integer;

begin
     for h:= length-1 downto 0 do
        begin
             inputarray[h+1]:=inputarray[h];
        end
end;

procedure setinsample(var phaseresult:results;
                      var worksample:sample) ;
{sets up a set of input samples}

          var h,y:integer;
              temp:real;

begin
     y:=360;
      for h:=0 to nmax do
       begin
```

```
  phaseresult.noiseimpulsearray[h]:=round(worksample[h]);
  temp:=phaseresult.anglearray[h]+phaseresult.datain[h]
                    +worksample[h];
            modulus(temp,y,temp);
            greaterthan180(temp);
            worksample[h]:=temp;
       end;
end;

procedure convert_to_m_binary(inputsample:sample;
                              var binsample:intsample);
{converts real input sample (simulated from prom output) to
m bit binary representation}

         var h:integer;

begin
     for h:= 0 to nmax do
       begin
           binsample[h]:=round(inputsample[h]*m/360);
           if binsample[h]= 128 then binsample[h]:=-128;
       end
end;

procedure removedata(qsk, currentsample:integer;
                     var phaserr:integer);
{removes data form incoming signal and calcultes phase
error of predicted value and local oscillator phase}

        var multi:integer;

begin
     multi:=2+2*qsk;
     currentsample:=(currentsample*multi) mod m ;
     phaserr:= divide2scom(m,currentsample,multi);
end;
```

```
procedure locate_cell(cerror:real;var cellnumber:integer;
                      setup:menurecord);

{locates the cell number that  a particular value to  thete
resides in}
          var theta:real;
 begin
     theta:=cerror;
     if theta<0 then
      cellnumber:=round((theta+setup.refstepsize/2)
                    /setup.refstepsize)
      else
       cellnumber:=round((theta-setup.refstepsize/2)
                         /setup.refstepsize);
end;


procedure referrorplot(phaseresult:results;
                       setup:menurecord;
                       var              cellcount:intarray);
{accumulates  count  of  cells  that  theha  falls  in  for
calculation of estimated phase deviation}

        var h,cellnumber:integer;
begin
     for h :=0 to nmax do
      begin
       locate_cell(phaseresult.cerror[h],cellnumber,setup);
       cellcount[cellnumber]:=cellcount[cellnumber]+1;
     end;
end;

procedure savedisc(cellcount:intarray;listends:integer;
                                   sigma,stepsize:real);
{ write data to disc as a text file}
```

```
      var outfile1:text;
          h,numberofsamples:integer;
          etheta:real;
          name:string;

begin
     numberofsamples:=trunc(setup.fs*1e3*setup.runtime);
     str(setup.ebno,name);
     name:=concat('OP',setup.datatype,name,'.dat');
     assign(outfile1,name);
     rewrite(outfile1);
     writeln(outfile1,listends,stepsize,sigma);
     for h:= -listends to listends do
       begin
         writeln(outfile1,cellcount[h]/numberofsamples);
                                  {calculates probability}
       end;
end;
procedure refdatasave(cellcount:intarray;setup:menurecord;
                      statresult:statrecord);
{saves reference data to file }

begin
     savedisc(cellcount,setup.reflistends,
              statresult.carrierdev,
              setup.refstepsize);
end;

procedure saveresult(setup:menurecord;
                     statresult:statrecord; filename,
                     display,trap,ercount:string);
{Svaes results to disk}

     var  outfile1:text;

begin
```

```
assign(outfile1,filename);
rewrite(outfile1);
writeln(outfile1);
  writeln(outfile1,'****************** PHASE
          ESTIMATOR COMPUTE MENU**********************
          *******');
  writeln(outfile1, 'Eb/No = ',setup.ebno:5,' dB,',
          ' SIM time = ',setup.runtime:6:4,' sec,',
   ' Data =  ',setup.dstate,',',' Type = ',
          setup.datatype);
  writeln(outfile1,'Noise =  ',setup.nstate,',',
         ' Frequency offset ',setup.fo:10:1,' Hz,',
         ' Symbol rate = ',setup.fs:6:1,' Khz');
  writeln(outfile1,'Number of traps = ',
          setup.trapcount:4,',',','
          Bit Error rate = ', setup.errate:10:6,',',
         ' Symbol Error Count = ',setup.errcount:4 );
  writeln(outfile1,'Slope detector is ', setup.slope,
          '   Mean filter output is ',  setup.mean);
  writeln(outfile1,'Number of feedback bits in IIR= ',
            setup.bits:3);
  writeln(outfile1,'Slope Coefficient = ',
          setup.slopecoef,
     '     Mean Coefficient = ',setup.meancoef);
  writeln(outfile1,'Input noise mean = ',
          statresult.errormean:6:3 ,' deg',
   '  Input standard deviation = ',
        statresult.errordev:6:3,
   ' deg');

  writeln(outfile1,'**************************
        ****************************************');
  writeln(outfile1);
  writeln(outfile1,'  1) Number of quantizing bits ',
          setup.quantbits);
  writeln(outfile1);
```

```
        writeln(outfile1,'  2) Magnitude of input signal
                (0-1) ',setup.mag:6:2);
        writeln(outfile1);
        writeln(outfile1,'  3) Display Estimator Run Time
                              Data ',display);
        writeln(outfile1);
        writeln(outfile1,'  4) Error Trap Data and save to
                              disk, Trap on ',trap);
        writeln(outfile1);
        writeln(outfile1,'  5) Error rate Count  ',
                ercount);
        writeln(outfile1);
        writeln(outfile1,'  6) Compute');
        writeln(outfile1);
        writeln(outfile1,'  7) Auto compute');
        writeln(outfile1);
        writeln(outfile1,'  0) End');
        writeln(outfile1);
        close(outfile1);
end;

procedure iirfilter( x:integer;var filout,
                     accminus1:integer;
                     coef:string; setup:menurecord);
{IIR filter function  using 16 bit  arithmetic to  simulate
constructed system}
{For 16 bit  accumulator,16 bit  output from  accum is  fed
back,as n bit prom address}
{plus 1 bit for complement signal}

         var acc,dif,multin,multout,augend,addend,
             shift:integer;
             difbits:real;

begin
     difbits:=16-setup.bits;
```

```
      augend:=inverse2s(m,x);
      x:=augend;
      if coef= '15/16'then x:=x*16;
      if coef='31/32' then x:=x*8;
      if coef='63/64' then x:=x*4;
      if coef= '127/128' then x:=x*2;{converts 8 bit input
                                        to}
                                    {faction of max i.e.+/-
                                     32768}
      shift:=round(power(2,difbits));
      multin:= (accminus1 div shift)*shift;
                        { 16 bits rounded to n bit address}
      dif:= augend- (multin div 256);
              {converts to 8 bits from 16 bits for direction
               test}
      if dif>128 then
        multin:=65536+multin;
      if dif<-128 then
       multin:=-65536+multin;
       if coef='15/16'then multout:= 15*multin div 16;
       if coef='31/32' then multout:=31*multin div 32;
       if coef='63/64' then multout:= 63*multin div 64;
       if coef='127/128' then multout:= 127*multin div 128;
      addend:=multout; augend:=x;
      acc:=(augend+addend) ;{accum is 16 bits}
      accminus1:=acc;
      filout:=acc;
end;

procedure iirslopedet(noiseyslope,dt,qsk:integer;
                      var projectslope,slopaccuminus1:integer;
                      var slopedly:delayspace;
                          setup:menurecord);
{calculates the slope of the changing phase vector}

      var meanslope,slope:integer;
```

```
            coef:string;
            multi:real;
begin
    coef:=setup.meancoef;
    slopedly[0]:= noiseyslope;
    slope:=subcyclic(slopedly[0],slopedly[dt]);
    iirfilter(slope,meanslope,slopaccuminus1,
            setup.slopecoef,setup);
                          {meanslope is 16 bit result}
    meanslope:=meanslope div 16;
     if meanslope<-m/2*16 then meanslope:= m*16+meanslope;
     if meanslope>((m/2*16)-1) then
         meanslope:=meanslope-(m*16);
     shiftintarray(slopedly,dt);
     if coef='15/16' then multi:=16.5;
     if coef='31/32' then multi:=32.5;
     if coef='63/64' then multi:=64.5;
     if coef='127/128' then multi:=128.5;
  projectslope:= round(multi*meanslope/(dt*16*(2+2*qsk)));
    {*16 converts to 8 bits from 12}
    projectslope:=convert2scom(m,projectslope);
end;

procedure sloperemovedata(qsk,inputvalue:integer;
                          var fourxslope:integer);

{removes data from input slope signal}

         var multi:integer;

begin
     multi:=2+2*qsk;
     fourxslope:=(inputvalue*multi) mod m;
end;

procedure freqdetect(inputvalue,qsk,dt:integer;
```

```
                    var pslope,slopaccuminus1:integer;
                    var slopedly:delayspace;
                    setup:menurecord);

{calculates frequency offset from nominal}

        var fourxslope:integer;

begin
     sloperemovedata(qsk,inputvalue,fourxslope);
     iirslopedet(fourxslope,dt,qsk,pslope,slopaccuminus1,
                 slopedly,setup);
end;


procedure errortrap(phaseresult:results;
                    var setup:menurecord;
                    loop:integer;var lastdata:integer;
                    var cycleslip,symerror:boolean);
{looks for errors and cycle slips in the decoded data}

        var h,trapointer,startdata,comparedata,
            datain:integer;

begin
      cycleslip:=false;
      symerror:=false;
      if phaseresult.datain[0]<0 then
        datain:=360+phaseresult.datain[0]
       else
        datain:=phaseresult.datain[0];
      datain:=convertdegtoqnt(datain);
      h:=1;
      trapointer:=0;
      if loop=0 then
        startdata:= phaseresult.dataoutarray[0]-datain
```

```
        else
        startdata:=lastdata;
      while h<= nmax do
       begin
       if phaseresult.datain[h]<0 then
        datain:=360+phaseresult.datain[h]
       else
        datain:=phaseresult.datain[h];
      datain:=convertdegtoqnt(datain);
      comparedata:=phaseresult.dataoutarray[h]-datain;
      if comparedata<>startdata then
      begin
       setup.errcount:=setup.errcount+1;
       symerror:=true;
       trapointer:=trapointer+1;
       if trapointer>10 then trapointer:=10;
      end;
      if trapointer= 10 then
         begin
           startdata:=comparedata;
                  { if different values of data for 10 }
                  {samples then reset startdata to compare
                   data}
           cycleslip:=true;
         end;
       if comparedata=startdata then
       begin
        trapointer:=trapointer-1;
        if trapointer<0 then trapointer:=0;
       end;
         h:=h+1;
    end;
    lastdata:=startdata;
end;
```

```
procedure phase_estimate1(var phaseresult:results;
                  setup:menurecord;qsk:integer;
                  startphase:real;binsample:intsample;
                  display,trap,ercount:string;
                  avl,adl,dt,loop:integer;
                  var accum,slopaccuminus1,predphase,
                  numberoftraps,lastdata:integer;
                  var averphase,averslope,
                  slopedly:delayspace);

{uses binsamples to estimate the carrier phase and
frequency error using IIR Filter with variable coefficient
+ freqency offset detection and projection, in parallel,
from difference samples taken over 8 symbol periods and
averaged using an iir filter}

        label 10,20;

        var minuscarrier,promsample,noiseylophase,phaserr,
            projectslope,newsample,phasemean,mphaseout,
            dataout,noiseimpulse:integer;
            modcerror,h,erraccum:integer;
            angle,angle1,elapsedtime:real; begin
    if display='no' then goto 10;
      writeln('Elapsed':8,'Data':5,'angle':6,'bin':4,
              'noise':5,'offsetP':8,'mean':5,'slope':6,
              'thetaP':7,'ModO/p':7,'C/Est ':7,'Data':6);
      writeln('sec':8,'degs':5,'degs':6,'qnt':5,'deg':5,
              'qnt':8,'qnt':5,'qnt':6,'qnt':7,
              'qnt':7,'degs':6);

10:  for h:=0 to nmax do
{ process binsamples in a sequential manner to obtain
output sequence}
    begin
       angle1:=phaseresult.anglearray[h]*m/360;
```

```
                                        {convert to binary
                                         term}
     phaseresult.cerror[h]:=angle1+(m-predphase);
modulus(phaseresult.cerror[h],m,phaseresult.cerror[h]);
     modcerror:=round(phaseresult.cerror[h]);
     removedata(qsk,modcerror,modcerror);
     phaseresult.cerror[h]:=modcerror;
     if phaseresult.cerror[h]>m/2-1 then
        phaseresult.cerror[h]:=phaseresult.cerror[h]-m;
  phaseresult.cerror[h]:=phaseresult.cerror[h]*360/256;
     promsample:=convert2scom(m,binsample[h]);
     if setup.slope='ON' then
      freqdetect(promsample,qsk,dt,projectslope,
                 slopaccuminus1,slopedly,setup)
     else
      projectslope:=0;
     minuscarrier:=subcyclic(promsample,predphase);
     removedata(qsk,minuscarrier,phaserr);
     phaserr:=convert2scom(m,phaserr);
     dataout:= subcyclic(minuscarrier,phaserr);
     noiseylophase:=addcyclic(phaserr,predphase);
     if setup.mean='ON' then
      iirfilter(noiseylophase,phasemean,accum,
                setup.meancoef,setup)
     else
      phasemean:=0;
     phasemean:= (phasemean div 256)  ;
     if phasemean<-m/2 then phasemean:= m+phasemean;
     if phasemean>m/2 -1 then phasemean:=phasemean-m;
     phasemean:=convert2scom(m,phasemean);
     predphase:=addcyclic(projectslope,phasemean);
     mphaseout:= inverse2s(m,phasemean);
     noise[h]:=inverse2s(m,phaserr);
     elapsedtime:=(loop*(nmax+1)+(h+1))/(setup.fs*1e3);
     phaseresult.elapsedarray[h]:=elapsedtime;
     phaseresult.noiseyloarray[h]:=noiseylophase;
```

```
        phaseresult.meanphasearray[h]:=mphaseout;
        phaseresult.projectslopearray[h]:=projectslope;
        phaseresult.predphasearray[h]:=predphase;
        phaseresult.dataoutarray[h]:= dataout;
        phaseresult.binsample[h]:=binsample[h];

   phaseresult.noise[h]:=noise[h];
        if display='no'then goto 20;
        writeln(elapsedtime:8:6,phaseresult.datain[h]:5,
             phaseresult.anglearray[h]:6:1,binsample[h]:5,
             phaseresult.noiseimpulsearray[h]:5,
             noiseylophase:8,mphaseout:5,projectslope:6,
             predphase:7,noise[h]:7:1,
             phaseresult.cerror[h]:6:1,dataout:6);
   20:end;
end;


procedure mean2scom(var meandif:delayspace;newsample,
                    len:integer;var phasemean:integer;
                    var accum:integer);
{returns a  current mean  value from  a sample  per  sample
input}


         var differ,y:integer;
begin
     y:=len*m;
     meandif[0]:= newsample;
     differ:= subcyclic(meandif[0],meandif[len]);
     differ:= inverse2s(m,differ);
     differ:= convert2scom(y,differ);
                                {converts from 8bit to 12
                                 bit}
     accum:= (accum + differ) mod y;
     phasemean:= divide2scom(y,accum,len);
```

```
     phasemean:= inverse2s(y,phasemean);
     phasemean:= convert2scom(m,phasemean);
     shiftintarray(meandif,len);
end;

procedure slopedet(noiseylophase,dt,avl,adl:integer;
                   var projectslope,slopaccum:integer;
                   var slopedly, averslope:delayspace);

     var meanslope,slope:integer;

begin
    slopedly[0]:= noiseylophase;
    slope:=subcyclic(slopedly[0],slopedly[dt]);
    mean2scom(averslope,slope,adl,meanslope,slopaccum);
    shiftintarray(slopedly,dt);
    meanslope:=inverse2s(m,meanslope);
    projectslope:= round((avl/2+0.5)/dt*meanslope);
    projectslope:=convert2scom(m,projectslope);
end;


procedure phase_estimate2(qsk:integer;binsample:intsample;
                          var noise:sample;display,trap,
                              ercount:string;
                          avl,adl,dt:integer;
                    var accum,slopaccum,predphase:integer;
                     var averphase,averslope,
                          slopedly:delayspace);

{uses binsamples to estimate the carrier phase and
frequency error using mean phase by averaging + freqency
offset detection and projection, in parallel, from
difference samples taken over 8 symbol periods and
averaged}
```

```
        label 10,20;

        var minuscarrier,promsample,noiseylophase,phaserr,
            projectslope,newsample,phasemean,mphaseout,
            dataout:integer;
            h:integer;

begin
     writeln('Entered phase estimator');
     if display='no' then goto 10;
       writeln('h':10,'binsample':10,'meanphase':10,
               'lophase':10,'predicted':10,'phaserr':10,
               'Data out':10);
 10: for h:=0 to nmax do
{ process binsamples in a sequential manner to obtain
output sequence}
     begin
        promsample:=convert2scom(m,binsample[h]);
        minuscarrier:=subcyclic(promsample,predphase);
        removedata(qsk,minuscarrier,phaserr);
        noiseylophase:=addcyclic(phaserr,predphase);
        mean2scom(averphase,noiseylophase,avl,phasemean,
                  accum);
        slopedet(noiseylophase,dt,avl,adl,projectslope,
                 slopaccum, slopedly,averslope);
        predphase:=addcyclic(projectslope,phasemean);
        mphaseout:= inverse2s(m,phasemean);
        dataout:= subcyclic(minuscarrier,phaserr);
        noise[h]:=inverse2s(m,phaserr);
        if display='no'then goto 20;
        writeln(h:10,binsample[h]:10,mphaseout:10,
                noiseylophase:10,predphase:10,
                noise[h]:10:1, dataout:10);
   20:end
 end;
```

```
procedure phaselock(var phaseresult:results;
                        setup:menurecord;
                        worksample:sample;
                        startphase:real;qsk:integer;
                        display,trap,ercount:string;
                        avl,adl,dt,loop:integer;
                    var accum,slopaccuminus1,predphase,
                        numberoftraps,lastdata:integer;
                    var averphase,averslope,
                        slopedly:delayspace);

        var binsample:intsample;
            meandif:delayspace;
            ch:char;
            h:integer;



begin{phaselock main body}
    writeln('entered phaselock');
    startphase:=startphase*180/pi;{converts to degrees};
    setinsample(phaseresult,worksample);
    convert_to_m_binary(worksample,binsample);
    writeln('converted to binary');
    if setup.filter='IIR' then
     phase_estimate1(phaseresult,setup,qsk,startphase,
                     binsample, display,trap,ercount,
                     avl,adl,dt,loop,accum,slopaccuminus1,
                     predphase,numberoftraps,lastdata,
                     averphase,averslope,slopedly)
    else
     phase_estimate2(qsk,binsample,noise,display,trap,
                     ercount,avl,adl,dt,accum,
                     slopaccuminus1,predphase,averphase,
                     averslope,slopedly);
end;
```

```
{************************************************************
************}
{set up menu procedures}

procedure  readname(var name,quit:string);
{read file name}

begin
     write('File name= ');
     read(name);
     quit:='no';
end;

procedure readdata(name:string;var ycord:samplearray;
                   var maxy:real;var listends:integer;
                   var sigma,stepsize:real);
{read data of phase noise deviation from file}

       var h:integer;
           infile1:text;
begin
     assign(infile1,name);
     reset(infile1);
     readln(infile1,listends,stepsize,sigma);
     stepsize:=stepsize*pi/180;
     sigma:=sigma*pi/180;
     h:=-listends;
     while h<=listends do
       begin
         readln(infile1,ycord[h]);
         if ycord[h]>maxy then maxy:=ycord[h];
                 {find maximum value for y}
         h:=h+1;
       end;
       close(infile1);
```

```
      if  h<> listends+1  then writeln('errror  in  reading
file');
end;

procedure readfile(var listends:integer;var sigma:real;
                   var stepsize:real;
                   var name,quit:string;
                   var ycord:samplearray;var maxy:real;
                   var readbefore:boolean;
                   var esno:integer);

{reads data from named file}
         label 10,20,30;

         var h:integer;
             inebno:real;
             checkok:boolean;
             ch:char;

begin
     Writeln('Symbol Energy Es/No dB  required (integer
              values only)');
     readln(inebno);
     esno:=round(inebno);
     str(esno,name);
     name:=concat('ebno',name,'.dat');
10:  checkok:=fstat(name);
     if checkok=false then
      begin
20:  writeln('File not found :Enter file name (y/n)');
       readln(ch);
       case ch of
         'y': readname(name,quit);
         'Y': readname(name,quit);
         'n': quitread(quit);
         'N': quitread(quit);
```

```
         otherwise goto 20;
         end;
        goto 10
      end;
      if quit='yes' then goto 30;
      readdata(name,ycord,maxy,listends,sigma,stepsize);
      readbefore:=true;
30:end;

procedure erfc(listends:integer;stepsize:real;
               ycord:samplearray);

{produces complimentary error  function i.e probability  of
error being greater than x}

         var h:integer;
             etheta,x,x1:real;
             sum,prob_error:longreal;

begin
     writeln('Enter prob(phase error >x degrees)');
     readln(x);
     x1:= x*pi/180;
     sum:=0;{calculates complimentary errror function}
     for h:= -listends to listends do
      begin
       etheta:= h*stepsize;
       if abs(etheta)<=x1 then sum:= sum+ycord[h];
           {probability of etheta<x1 radians}
      end;
      prob_error:=1-sum;
      writeln('Probability of phase error >',x:4:2,' deg  =
              ',prob_error:15:10);
end;
```

```
procedure pseudorandom(qsk:integer;var iv:integer;var
                       random:intsample);

{from an initial set up value,iv, a pseudo random stream is
produced using 23 stage shift register}


      Type  rnumber=array[1..23] of integer;

      var   shiftreg : rnumber;
            i,ip,temp,h,j:integer;
            p:real;



begin
      {initialize shift register with start point}
      for h:= 23 downto 1 do { turns seed value into binary
                               form}
        begin
          shiftreg[h]:=0;
          p:= power(2,h);
          ip:=round(p);
            shiftreg[h]:=iv div ip;
            iv:=iv mod ip;
        end;
  for j:=0 to qsk do
  begin
   for i:= 0 to nmax do
    begin
      temp:=( shiftreg[18]+shiftreg[23]) mod 2;
      if j=0 then
        random[i]:=shiftreg[23]
      else
        random[i]:=random[i]+2*shiftreg[23];
      for h:= 22 downto 1 do
```

```
          begin
              shiftreg[h+1]:=shiftreg[h];
          end;
      shiftreg[1]:=temp;
    end;
   end;
end;


procedure nodata(var data:intsample);

{clears data  array}
   var h:integer;

begin
     for h:=0 to nmax do
      begin
        data[h]:=0;
      end;
end;



procedure nonoise(var worksample:sample);

{fills noise locations with zero}

       var h:integer;
begin
      for h:= 0 to nmax do
      begin
      worksample[h]:=0;
      end;
end;

procedure noiseset(ycord:samplearray;var worksample:sample;
```

```
                    listends:integer;stepsize:real);

       var n,h:integer;
          sum,uniform:real;
begin
     for n:=0 to nmax do
     begin
      uniform:=round(1e9*rand)/1e9;
      if uniform=1 then uniform:=1-1e-9;
      sum:=0;
      h:=-listends;
      while h<=listends do
      begin
       sum:=sum+ycord[h];
       if sum>uniform then
       begin
        worksample[n]:=h*stepsize;
        h:=listends+1; {leave loop}
       end;
       h:=h+1;
      end;
     end;
end;



procedure denzero(num:integer;var ramp:real);

{set num if denominator zero}

begin
      if num=0 then ramp:=0;
      if num>0 then ramp:=pi/2;
      if num<0 then ramp:=-pi/2;
end;
```

```
procedure promerror(angle,ramp:real;
                    var sum:real;h:integer);

        var error,y:real;

{calculates error due to prom quantization}

begin
            y:=360;
            angle:=180*angle/pi;
            modulus(angle,y,angle);
            ramp:= 180*ramp/pi;{converts to degrees}
            if ramp<0 then ramp:= ramp+360;
            ramp:=round(ramp*m/360);{converts  to   m   bit
binary}
            ramp:= ramp*360/m;{ converts  back to real  for
error}
            error:=angle+(360-ramp);
            modulus(error,y,error);
            if error>180 then error:=error-360;
            sum:=sum+sqr(error);
end;

procedure frequencyoffset(var phaseresult:results;
                          fo,fs,mag:real;
                          quantsteps:integer;
                          var startphase,nextstartphase:real);

{allows frequency offset of carrier to be set and  converts
to phase change/sample}

        label 10;
        var y,h,num,den:integer;
            sum,sigma,normf,angle,ramp,noisesam:real;
            waittime:timeseconds;
```

```
begin
        y:=360;
        normf:=fo/fs;
        sum:=0;
        if fo=0 then
        begin
          for h:=0 to nmax do
            begin
            phaseresult.anglearray[h]:=0;
            end;
         goto 10;
        end;
        for h:= 0 to nmax do
         begin
           angle:= 2*pi*normf*h+startphase;
           num:=trunc(sin(angle)*mag*quantsteps);
           den:=trunc(cos(angle)*mag*quantsteps);
           if den=0 then
           denzero(num,ramp)
           else
            ramp:= arctan(num/den);
           if (num=0)and (den=0) then ramp:=0;
           if (num>=0) and (den<0) then ramp:= pi+ramp;
           if (num<0) and (den<0) then ramp:= -pi+ramp;
           promerror(angle,ramp,sum,h);
           {convert to degrees}
           ramp:= ramp*180/pi;
           modulus(ramp,y,ramp);
           phaseresult.anglearray[h]:=ramp;
        end;
    10: nextstartphase:=2*pi*normf*(nmax+1)+startphase;
        modulus(nextstartphase,2*pi,nextstartphase);
        if nextstartphase>pi then
           nextstartphase:=nextstartphase-(2*pi);
        sigma:=sqrt(sum/(nmax));
        writeln('Prom Sigma (standard deviation) from
```

```
                    phase angle = ',sigma:6:2,'Degrees');
end;

procedure bpskdata(var phaseresult:results);
{set up data with random phase angles}

         var iv,h,qsk:integer;
             random:intsample;
begin
          qsk:=0;
          iv:=4197376;
          pseudorandom(qsk,iv,random);
          h:=0;
          while h<=nmax do
          begin
            case random[h] of
              0: phaseresult.datain[h]:=0;
              1: phaseresult.datain[h]:=180;
            end;
            h:=h+1;
           end;
end;

procedure qpskdata(var phaseresult:results);
{set up qpsk random phase samples}

             var h,i,iv,qsk:integer;
                 random:intsample;

begin
              qsk:=1;
              iv:=4065273;
              pseudorandom(qsk,iv,random);
              h:=0;
              while h<=nmax do
                begin
```

```
                case random[h] of
                   0:phaseresult.datain[h]:=0;
                   1:phaseresult.datain[h]:=90;
                   2:phaseresult.datain[h]:=180;
                   3:phaseresult.datain[h]:=-90;
                 end;
                 h:=h+1;
                end;
end;

procedure datayes(var qsk:integer;var phaseresult:results);
{creates data}
         var ch:char;
             i,iv:integer;
 begin
        case qsk of
          0:bpskdata(phaseresult);
          1:qpskdata(phaseresult);
          otherwise
            writeln('qsk=',qsk);
        end;
end;
procedure setoffset(var fo:real);

{sets offset frequency }

begin
    writeln('Enter frequency offset in Hz');
    readln(fo);

 end;

procedure setout(var out:string);
{set display output on or off}

begin
```

```
     if out='ON' then out:='OFF'
     else
       out:='ON';
end;

procedure setcoef(var coef:string); {set coefficent value}

     var temp:string;

begin
     if coef='15/16' then
      temp:='31/32';
     if coef='31/32' then
      temp:='63/64';
     if coef='63/64' then
      temp:='127/128';
     if coef='127/128' then
      temp:='15/16';
     coef:=temp;
end;

procedure setbits( var number:integer);
{set number of bits in feedback variable}

     label 10;

begin
  10:write('Enter number of feedback address bits ');
     readln(number);
     if (number <1) or (number>16) then
      goto 10;
end;


procedure setiirmenu( var setup:menurecord);
```

```
{sets up iir filters in phase and slope processing
sections}

        label 10,20;
        var value:choice;
            quit:string;

begin
     quit:='no';
     setup.filter:='IIR';
10:  writeln('*****************************************
          ********************');
     writeln('*                    IIR Filters menu
          *');

     writeln('*******************************************
          ******************');
     writeln;
     writeln(' 1) Slope output ',setup.slope);
     writeln;
     writeln(' 2) Slope Coefficient ',setup.slopecoef);
     writeln;
     writeln(' 3) Mean output ',setup.mean);
     writeln;
     writeln(' 4) Mean Coefficient ', setup.meancoef);
     writeln;
     writeln(' 5) Number of Address Bits for Feedback Word
          ',setup.bits);
     writeln;
     writeln(' 0) End');
     writeln;
     write(' ENTER ');
     readln(value);
     case value of
      0: quitread(quit);
      1: setout(setup.slope);
```

```
      2: setcoef(setup.slopecoef);
      3: setout(setup.mean);
      4: setcoef(setup.meancoef);
      5: setbits(setup.bits);
      otherwise goto 10;
      end;
      if quit='yes' then goto 20;
      goto 10;
20:end;


procedure setfilter(var setup:menurecord);
{calls setup procedure for setting filter types}

      label 10,20;

      var value:choice;
          quit:string;
begin
      setiirmenu(setup);
      goto 20;{ skip this menu}
      quit:='no';
  10: writeln(' 1) IIR filter ');
      writeln;
      writeln(' 2) Averageing');
      writeln;
      writeln(' 0) End');
      writeln;
      write('ENTER ');
      readln(value);
      case value of
      0: quitread(quit);
      1: setiirmenu(setup);
      2: setup.filter:='Averaging';
      otherwise goto 10;
      end;
```

```
      if quit='yes' then goto 20;
      goto 10;
20:end;


procedure setnoise(var nstate:string);
{sets state of noise samples}

begin
     if nstate='yes' then
     begin
       nstate:='no'
     end
    else
      nstate:='yes'
end;

procedure setdstate(var dstate:string);
{set state of data samples}

begin
     if dstate='no' then dstate:='yes'
     else
      dstate:='no';
end;

procedure setbpsk(var qsk:integer;var datatype:string);
{set type of PSK to BPSK}

begin
     qsk:=0;
     datatype:='BPSK';
end;

procedure setqpsk(var qsk:integer; var datatype:string);
{set type of PSK to QPSK}
```

```
begin
     qsk:=1;
     datatype:='QPSK';
end;

procedure setdata( var dstate,datatype:string;var
                   qsk:integer);
{calls setting of PSK type}
      label 10,20;
      var ch:char;

begin
     if dstate='yes ' then
     begin
      dstate:='no';
      goto 10;
     end
     else
     dstate:='yes';
   20: writeln('BPSK or QPSK (b/q/n) ?');
       readln(ch);
       case ch of
       'b':setbpsk(qsk,datatype);
       'B':setbpsk(qsk,datatype);
       'q':setqpsk(qsk,datatype);
       'Q':setqpsk(qsk,datatype);
       'n':setdstate(dstate);
       'N':setdstate(dstate);
       otherwise goto 20;
      end;
10: end;

procedure setsymbolrate(var fs:real);
{sets the symbol rate of the simulation}

       label 10;
```

```
begin
  10:   write('Enter symbol rate in Khz ');
        readln(fs);
        if (fs<=0)  then goto 10;
end;

procedure setruntime(var runtime:real);
{sets the simulated run time of the simulation}

     label 10;
begin
10 : write('Enter  required simulated run  time in  seconds
');
     readln(runtime);
     if runtime<=0 then goto 10;
end;

procedure degnoise(var worksample:sample);
{converts  from  radian  noise  samples  to  degree  noise
samples}

       var h:integer;

begin
     for h:= 0 to nmax do
       begin
       worksample[h]:=worksample[h]*180/pi;
       end;
end;

procedure maxmin(phaseresult:results);
{calculates maximum and minimum phase noise samples}

     var h:integer;
         max,min:real;
```

```
begin
     max:=-128; min:=127;
     for h:=0 to nmax do
     begin
     if max<phaseresult.noise[h] then
        max:= phaseresult.noise[h];
     if min>phaseresult.noise[h] then
        min:= phaseresult.noise[h]
     end;
     writeln('max phase error = ',max:5:1,' qnts');
     writeln('min phase error = ',min:5:1,' qnts' );
end;

procedure erroratio(loop:integer;var setup:menurecord);
{calculates error rate of simulation}
         var numberofsymbols,numberofbits:integer;
begin
     numberofsymbols:=(loop+1)*nmax;
     numberofsymbols:=(loop+1)*nmax;
     if setup.datatype='BPSK' then
       numberofbits:=numberofsymbols
     else
      numberofbits:=2*numberofsymbols;
      setup.errate:=setup.errcount/numberofbits;
end;
procedure sumvariance(var sd:real;oldsigma:real;
                      loop:integer);
{calculates current variance  of complete simulation  after
new simulated block}

         var currentvar,oldvar,newvar:real;

begin
     currentvar:=sqr(sd);
     oldvar:=sqr(oldsigma);
     newvar:=(nmax*loop*oldvar+(nmax*currentvar))
```

```
                /((loop+1)*nmax);
     sd:=sqrt(newvar);
end;

procedure resultstats(var statresult:statrecord;
                      var cellcount:intarray;
                      phaseresult:results;
                      loop,numberoftraps,
                      errcount:integer;
                      fs,errate:real);

{output statistical results to screen}

       var sd,sddeg,mean:real;
begin
      writeln('Elapsed time is ',
               (loop+1)*nmax/(fs*1e3):8:6,' sec');
      writeln('<<<<<Phase error output statistics>>>>>');
      maxmin(phaseresult);
      variance(phaseresult.noise,statresult.errormean,
               mean,sd);
      sddeg:= 360/m * sd;
      sumvariance(sddeg,statresult.errordev,loop);
      sd:=sddeg*m/360;
      writeln('Mean = ',mean:6:3,' qnts':5,' or ',
               mean*360/m :6:3,' deg');
      writeln('Standard Deviation =',sd:6:3,' qnts':5,
              ' or ', sddeg:6:3,' degrees':8);
      statresult.errormean:=mean;
      statresult.errordev:=sddeg;
      writeln('<<<<<Carrier Estimator statistics>>>>>');
      variance(phaseresult.cerror,statresult.carriermean,
               mean,sd);
      mean:=(mean*nmax+statresult.carriermean*nmax*loop)
             /(nmax*(loop+1));
      referrorplot(phaseresult,setup,cellcount);
```

```
      sumvariance(sd,statresult.carrierdev,loop);
      writeln('Mean = ',mean:6:3,' degrees':8);
      writeln('Standard Deviation = ',sd:6:3,' degrees':8);
      writeln('Number of Results trapped = ',
               numberoftraps);
      writeln('Bit Error Rate = ',errate:10:8);
      writeln('Symbol Error Count = ',errcount:6);
      writeln('****************************************
              ******************');
      statresult.carriermean:=mean;
      statresult.carrierdev:=sd;
end;


procedure compute(var setup:menurecord;
                  var statresult:statrecord;
                  stepsize:real; display,trap,
                  ercount:string;
                  var quit:string;qsk:integer;
                  ycord:samplearray;
                  listends:integer);

{computes phase estimation simulation from here}

       label 10,30;

       var    cellcount:intarray;
              worksample:sample;
              fs,startphase,nextstartphase,realquantbits,
              sigma1:real;
              predphase,avl,adl,dt,accum,slopaccuminus1,
              totalnumber,numberloops,loop,quantsteps,
              lastdata, presentcount:integer;
              averphase,averslope,slopedly:delayspace;
              noiseimpulsearray,noiseyloarray,
```

```
                meanphasearray,projectslopearray,
                predphasearray,dataoutarray:intresults;
                anglearray,elapsedarray:rlresults;
                phaseresult:results;
                trapsprung,symerror,cycleslip,
                keypress:boolean;

begin
    writeln('Computing- Please wait');
    quit:='no';
    realquantbits:=setup.quantbits;
    quantsteps:=round( power(2,realquantbits-1));{single
                                                    sided}
    fs:=setup.fs*1e3;{converts to Hz}
    sigma1:=setup.sigma*180/pi;
    totalnumber:=trunc(setup.runtime*fs);
    statresult.carriermean:=0;
    statresult.carrierdev:=0;
    clearcellcount(cellcount);
    numberloops:=totalnumber div nmax;
    initializeloop(startphase,predphase,avl,adl,dt,accum,
             slopaccuminus1,averphase,averslope,slopedly);
    setup.errate:=0;
    setup.trapcount:=0;
    setup.errcount:=0;
    statresult.errormean:=0;
    statresult.errordev:=0;
    lastdata:=0; loop:=0;
    while loop<= numberloops-1 do
    begin
     keypress:=false;
     trapsprung:=false;
     if setup.nstate='no' then
      nonoise(worksample)
     else
      begin
```

```
      noiseset(ycord,worksample,listends,stepsize);
      degnoise(worksample);
     end;
    writeln('Noise standard deviation (sigma)  = ',
            sigma1:6:1,' Degrees');
    frequencyoffset(phaseresult,setup.fo,fs,setup.mag,
            quantsteps,startphase,nextstartphase);
    if setup.dstate='no' then
    begin
     nodata(phaseresult.datain);
     goto 10;
    end
    else
     datayes(qsk,phaseresult);
10: writeln('*********************************************
            ******************');
    writeln('Frequency offset = ',setup.fo:10:2,' Hz');
    writeln('Noise ',setup.nstate);
    writeln('Data  ',setup.dstate,setup.datatype:6);
    phaselock(phaseresult,setup,worksample,startphase,qsk,
             display,trap,ercount,avl,adl,dt,loop,accum,
             slopaccuminus1,predphase,setup.trapcount,
             lastdata,averphase,averslope,slopedly);
    startphase:=nextstartphase;
    if ercount= 'no' then
     presentcount:=setup.errcount;
    errortrap(phaseresult,setup,loop,lastdata,cycleslip,
             symerror);
    if ercount='no' then setup.errcount:=presentcount;
    erroratio(loop,setup);
    writeln('Eb/No = ',setup.ebno:3,'dB', ' Simulation time
            = ', setup.runtime:6:4,' sec');
    writeln('Slope coefficient = ',setup.slopecoef,'  Mean
            coefficient = ', setup.meancoef);
    resultstats(statresult,cellcount,phaseresult,loop,
           setup.trapcount,setup.errcount,setup.fs,
```

```
            setup.errate);
    if trap= 'No' then goto 30;
    if (trap= 'Cycle slip') and (cycleslip=true) then
     trapsprung:=true;
    if (trap= 'Symbol error') and  (symerror=true) then
     trapsprung:=true;
    setup.trapsprung:=trapsprung;
    if trapsprung=true then
    begin
      phaseresult.trap:=trap;
      disksave(setup,statresult,phaseresult);
       setup.trapcount:=setup.trapcount+1;
    end;
 30:loop:=loop+1;
    keypress:=cstat;
      if keypress=true then
        keypressed(quit);
      if quit='yes'then
       begin
        writeln('!!!!!!!!!! EXIT AT END OF NEXT BLOCK
                 !!!!!!!!!!');
         loop := numberloops;
       end;
   end;
    refdatasave(cellcount,setup,statresult);
end;

procedure setststp(var value:integer);
{set up start and stop values for auto computation}

    label 10;
    var invalue:real;

begin
 10:write('Enter Eb/No ');
    readln(invalue);
```

```
    value:=round(invalue);
    if (value<3) or (value>10) then goto 10;
end;


procedure setcoefinc(var coefinc:string);
{set up for auto coefficent increment}

begin
  if coefinc='no' then
    coefinc:='yes'
  else
   coefinc:='no';
end;

procedure run(var setup:menurecord;
              var statresult:statrecord;
              stepsize:real;
              display,trap,ercount,filename:string;
              qsk:integer;ycord:samplearray;listends,
              filenumber,start,stop:integer);

{runs program manually for one set up position}

       label 10,40;
       var h,slopecoefno:integer;
           runtime,fs:real;
           quit,name :string;
           checkok:boolean;

begin
     h:=start; quit:='no';
     while h<= stop do
     begin
      if setup.datatype='QPSK' then
         setup.esno:=h+3
```

```
 else
   setup.esno:=h;
 str(setup.esno,name);
 name:=concat('ebno',name,'.dat');
 checkok:=fstat(name);
 if checkok=false then
  begin
    writeln('File ',name,' not found');
    h:=stop+1;
    quit:='yes';
  end;
  h:=h+1;
end;
if quit='yes' then goto 10;
quit:='no';
if setup.coefinc='yes' then
  slopecoefno:=1
else
  slopecoefno:=4;
while slopecoefno<=4 do
 begin
 h:=start;
 while  h<= stop do
 begin
   if setup.datatype='QPSK' then
    setup.esno:=h+3
   else
   setup.esno:=h;
   str(setup.esno,name);
   name:=concat('ebno',name,'.dat');
   writeln('Reading data from file ',name);
   readdata(name,ycord,maxy,listends,sigma,stepsize);
   setup.sigma:=sigma;
   str(filenumber,filename);
   filename:=concat('R',filename,'.dat');
   fs:=setup.fs*1e3;
```

```
          case h of
           3:runtime:=(100/2.28e-2)/fs;
           4:runtime:=(100/1.25e-2)/fs;
           5:runtime:=(100/5.95e-3)/fs;
           6:runtime:=(100/2.38e-3)/fs;
           7:runtime:=(100/7.7e-4)/fs;
           8:runtime:=(100/1.91e-4)/fs;
           9:runtime:=(100/3.4e-5)/fs;
           10:runtime:=(100/3.8e-6)/fs;
           otherwise runtime:=0;
           end;
           setup.runtime:=runtime;
           compute(setup,statresult,stepsize,display,trap,
                   ercount,quit,qsk,ycord,listends);
           saveresult(setup,statresult,filename,display,trap,
                      ercount);
           filenumber:=filenumber+1;
           if quit='yes' then h:=stop;
           h:=h+1;
        end;
        if quit='yes' then
         begin
          slopecoefno:=5;
          goto 40;
         end;
        slopecoefno:=slopecoefno+1;
        setcoef(setup.slopecoef);
 40:end;
10:end;


procedure autocompute(var setup:menurecord;
                      var statresult:statrecord;
                          stepsize:real;
                          display,trap,ercount:string;
                          qsk:integer;ycord:samplearray;
```

```
                              listends:integer);
{automatically computes  simulated  runs over  a  range  of
Eb/No}

        label 10,20;
        var filenumber,start,stop:integer;
            filename,quit:string;
            value:choice;


begin
     quit:='no';
     write('Enter initial file number (1-9999) ');
     readln(filenumber);
     str(filenumber,filename);
     filename:=concat('R',filename,'.dat');
     start:=3;
     stop:=7;
     setup.coefinc:='no';
10:  writeln('Saving Files from ',filename);
     writeln;
     writeln(' 1) Start Eb/No (min 3dB) = ',start:4,' dB');
     writeln;
     writeln(' 2) Stop Eb/No (max 10dB) = ',stop:4,' dB');
     writeln;
     writeln(' 3) Increment Slope Coefficient Values
             ',setup.coefinc);
     writeln;
     writeln(' 4) Run' );
     writeln;
     writeln(' 0) End ');
     writeln;
     write('ENTER ');
     readln(value);
     case value of
      0:quitread(quit);
```

```
      1:setststp(start);
      2:setststp(stop);
      3:setcoefinc(setup.coefinc);
      4:run(setup,statresult,stepsize,display,trap,ercount,
        filename,qsk,ycord,listends,filenumber,start,stop);
    otherwise goto 10;
    end;
    if quit='yes' then goto 20;
    goto 10;
20: end;

procedure setquantbits(var quantbits:integer);
{sets number of quantizing bits}

    label 10;
    var realquantbits:real;


begin
10:  Write('Enter number of quantizing bits required ');
     readln(realquantbits);
     quantbits:=round(realquantbits);
     if quantbits<1 then goto 10;
end;


procedure setmag(var mag:real);
{set magnitude of input signal}
    label 10;

begin
10:  write('Enter magnitude of input signal ');
     read(mag);
     if mag<=0 then goto 10;
end;
```

```
procedure setdisplay(var display:string);
{set display on or off}

begin
    if display='no' then display:='yes'
    else
      display:='no';
end;

procedure errortrapmenu(var trap:string);
{set error trap option}

   label 10;
   var value:choice;

begin
10:  writeln('*****************ERROR TRAP
              MENU*******************');
     writeln;
     writeln('!!!!!!!!!!INSERT DISK INTO DRIVE A - NOW
              !!!!!!!!!!');
     writeln;
     writeln(' 1) No trap ');
     writeln;
     writeln(' 2) Trap on cycle slip ');
     writeln;
     writeln(' 3) Trap on Symbol error');
     writeln;
     write(' ENTER ');
     readln(value);
     case value of
     1:trap:='No';
     2:trap:='Cycle slip';
     3:trap:='Symbol error';
     otherwise goto 10;
     end;
```

```
end;


procedure seterrorcount(var ercount:string);
{set error count string}

begin
     if ercount='no' then
        ercount:='yes'
     else
       ercount:='no';
end;


procedure computemenu(var setup:menurecord;
                      var statresult:statrecord;
                      stepsize:real;
                      qsk:integer; ycord:samplearray;
                      listends:integer;
                      var     display,ercount,trap:string);
{setup menu for compute options}

       label 10,20;
       var value:choice;

begin
       quit:='no';
   10: writeln;
       writeln;
       writeln('******************* PHASE ESTIMATOR COMPUTE
                MENU*****************************');
       writeln( 'Eb/No = ',setup.ebno:5,' dB,',' SIM time =
                ',setup.runtime:6:4,' sec,',
     ' Data =  ',setup.dstate,',',' Type = ',
```

```
setup.datatype);
writeln('Noise =  ',setup.nstate,',',
' Frequency offset ',setup.fo:10:1,' Hz,',
' Symbol rate = ',setup.fs:6:1,' Khz');
writeln('Number of traps = ',setup.trapcount:4,',',
        ' Bit Error rate = ',
  setup.errate:10:6,',', ' Symbol Error Count =
  ',setup.errcount:4 );
writeln('Slope detector is ', setup.slope,'   Mean
  filter output is ', setup.mean,
  ' Feedback bits in IIR = ',setup.bits:3);
writeln('Slope Coefficient = ',setup.slopecoef,
        '     Mean Coefficient = ',
         setup.meancoef);
writeln('Error noise mean = ',
         statresult.errormean*360/m:6:3 ,' deg',
 '  Error standard deviation = ',
 statresult.errordev:6:3,' deg');
 writeln('Sim. Noise S.D = ',
         setup.sigma*180/pi:6:3,' deg',
 ' Est S.D = ',statresult.carrierdev :6:3,' deg');
 writeln('****************************************
       ********************************');
 writeln('  1) Number of quantizing bits ',
           setup.quantbits);
 writeln;
 writeln('  2) Magnitude of input signal (0-1) ',
             setup.mag:6:2);
 writeln;
 writeln('  3) Display Estimator Run Time Data ',
        display);
 writeln;
 writeln('  4) Error Trap Data and save to disk,
           Trap on ', trap);
 writeln;
 writeln('  5) Error rate Count  ', ercount);
```

```
        writeln;
        writeln('  6) Compute');
        writeln;
        writeln('  7) Auto compute');
        writeln;
        writeln('  0) End');
        write('ENTER ');
        readln(value);
        case value of
        0: quitread(quit);
        1:setquantbits(setup.quantbits);
        2: setmag(setup.mag);
        3:setdisplay(display);
        4: Errortrapmenu(trap);
        5:seterrorcount(ercount);
        6:compute(setup,statresult,stepsize,display,trap,
          ercount, quit, qsk,ycord,listends);
        7:autocompute(setup,statresult,stepsize,display,
            trap,ercount,qsk,ycord,listends);
        otherwise goto 10;
        end;
        if quit='yes' then goto 20;
        goto 10;
    20:quit:='no';
end;


procedure estimatemenu(var setup:menurecord;ycord:samplearray;
                       var stepsize:real;
                       var qsk:integer;listends:integer);

      label 10,20;

      var value:choice;
          display,ercount,trap:string;
```

```
            statresult:statrecord;
begin

        display:='no';
        trap:='no';
        ercount:='no';
        quit:='no';
  10:   if setup.datatype='QPSK'then setup.ebno:=setup.esno-3
        else
         setup.ebno:=setup.esno;
        setup.errate:=0;
        setup.trapcount:=0;
        setup.errcount:=0;
        statresult.errormean:=0;
        statresult.errordev:=0;
        statresult.carrierdev:=0;
        statresult.carriermean:=0;
        writeln;
        writeln;
        writeln('*********PHASE ESTIMATOR MENU********');
        writeln('*          Eb/No    =    ',setup.ebno:5,'dB
*');
        writeln('*               version         ',ver:6:2,'
*');
        writeln('************************************');
        writeln;
        writeln('  1)  Frequency offset    ',setup.fo:10:1,'
Hz');
        writeln;
        writeln('  2) Add noise  ',setup.nstate);
        writeln;
        writeln('  3) Add data   ',setup.dstate,
setup.datatype:10);
        writeln;
        writeln('  4) Filter is ',setup.filter);
        writeln;
```

```
        writeln('  5) Symbol Rate ',setup.fs :6:1,'KHz ');
        writeln;
        writeln('  6) Simulated Run Time is ',
setup.runtime:10:6,' sec');
        writeln;
        writeln('  7) Compute');
        writeln;
        writeln('  0) End');
        writeln;
        write('  ENTER ');
        readln(value);
        case value of
        0: quitread(quit);
        1: setoffset(setup.fo);
        2: setnoise(setup.nstate);
        3: setdata(setup.dstate,setup.datatype,qsk);
        4: setfilter(setup);
        5: setsymbolrate(setup.fs);
        6: setruntime(setup.runtime);
        7: computemenu(setup,statresult,stepsize,qsk,ycord,
             listends,display,ercount,trap);
        otherwise goto 10;
        end;
        if quit='yes' then goto 20;
        goto 10;
20:end;


procedure callestimatemenu(listends:integer;stepsize,
                   sigma:real;ycord:samplearray;
                   var setup:menurecord);

         var qsk:integer;

begin
     setup.nstate:='no';
```

```
     setup.dstate:='no';
     setup.datatype:='QPSK';
     qsk:=1;{set to qpsk}
     setup.fo:=0;
     setup.filter:='IIR';
     setup.fs:=256;{in kHz}
     setup.runtime:=3e-3;
     setup.quantbits:=6;
     setup.mag:=1;
     setup.slope:='ON';
     setup.mean:='ON';
     setup.slopecoef:='63/64';
     setup.meancoef:='31/32';
     setup.bits:=16;
     setup.sigma:=sigma;
     setup.refstepsize:=0.5;{ stepsize in degrees,min=0.5}
     setup.reflistends:=round(180/setup.refstepsize);
     estimatemenu(setup,ycord,stepsize,qsk,listends);
end;



procedure menu(var value:choice;var quit:string;
               sesno:string; var setup:menurecord);

{produces menu of choices}

begin
     quit:='no';
     Writeln;
     writeln;
     writeln('************MENU**************');
     writeln('      Es/No = ',sesno);
     writeln('       version ',ver:6:2);
     writeln('********************************');
```

```
      writeln;
      writeln('Please make slection');
      writeln;
      writeln;
      writeln('  1) Readfile from disc.');
      writeln;
      writeln('  2) Calculate Erfc.  ');
      writeln;
      writeln('  3) Phase Estimator performance.');
      writeln;
      writeln('  0) End');
      writeln;
      write('  ENTER ');
      readln(value);
end;

begin{main program}
      readbefore:=false;
      sesno:='NOT SET';

10:   menu(value,quit,sesno,setup);
      if (readbefore=false) and ((value=2) or (value=3))
      then
       begin
        writeln('Warning:Must read from file first!!');
        waittime:=2;
        wait(waittime);
        goto 10;
      end
      else
        case value of
         1:readfile(listends,sigma,stepsize,name,quit,
            ycord,maxy,readbefore,esno);
         2:erfc(listends,stepsize,ycord);
         3:callestimatemenu(listends,stepsize,sigma,ycord,
             setup);
```

```
      0: goto 20;
     otherwise goto 10;
    end;
    setup.esno:=esno;
    str(esno,sesno);
    sesno:=concat(sesno,' dB');
    case value of
      1:waittime:=0;
      2:waittime:=5;
      3:waittime:=0;
      0:waittime:=0;
    end;
    wait(waittime);
   goto 10;
20:end.
```

Plot Phase Statistics Program

This program reads data from the phase estimator simulation a
nd

the noise gemeration program to enable the PDF's of the phase variations to be plotted.

Plot1-24.pas

```
program PlotPhaseStat(input,output);
{program reads data from named disc file and plots
probability distribution of phase error}

       label 10,20;

       const ver= 1.24;
       type gsxint =0..32767;
            coord=real;
            plotarray=array[0..720] of coord;
            choice=0..10;
       var quit,magx:string;
           mag:integer;

procedure     openworkstation(ident:gsxint);      external;
procedure     closeworkstation;                   external;
```

```
procedure     clearworkstation;                    external;
procedure setwindow(left,right,bottom,top:coord);
                                                   external;
procedure setviewport(left,right,bottom,top:coord);
                                                   external;
procedure     newpoly;                             external;
procedure polypoint(x,y:coord);                    external;
procedure drawpoly;                                external;
procedure clip;                                    external;
procedure plotstring(x,y:coord;s:string);          external;
procedure     stringheight(h:real);                external;
procedure setgetmode(device,mode:gsxint);          external;
procedure getstring(stringdev:gsxint;echo:boolean;
    x,y:coord;var     characters:string);          external;
procedure getvaluator(valdev:gsxint;var value:gsxint;
                      var terminator :char;
                      var status:gsxint);          external;


procedure quitread(var quit:string);

begin
     if         quit='no'         then         quit:='yes';
end;

procedure averageplot(var ycord:plotarray;var filend,
                      centreloc:integer;
                      var stepsize:real;var maxy:coord);

         label 10,20;

         var avecentre,centre,averend,averageover,i,h,
             n:integer;
             sum:real;
             averycord:plotarray;

begin
     averageover:=round(2/stepsize);
     if odd(averageover) = false then
       averageover:=averageover+1;
     stepsize:=averageover*stepsize;
     centre:= filend div 2;
     averend:=(filend+1) div averageover;
     avecentre:= (averend div 2);
     n:=avecentre;
     maxy:=0;
     h:=centre-(averageover div 2);
     while h<=filend do
        begin
        sum:=0;
        i:=0;
        while i<=averageover-1 do
          begin
          if h+i>filend then goto 10;
```

```
            sum:=ycord[h+i]+sum;
            ycord[h+i]:=0;{clears ycord array}
10:          i:=i+1;
        end;
        averycord[n]:=sum;
        h:=h+averageover;
        if averycord[n]>maxy then
        begin
          maxy:=averycord[n];
          centreloc:=n;
        end;
        n:=n+1;
     end;
     h:=centre- (averageover div 2)-1;
    {averages in negative direction starting from just
     below centre}
     n:=avecentre-1;
     while h>=0 do
        begin
        sum:=0;
        i:=0;
        while i<= averageover-1 do
          begin
          if h-i<0 then goto 20;
            sum:=ycord[h-i]+sum;
            ycord[h-i]:=0;
20:          i:=i+1;
         end;
        averycord[n]:=sum;
        h:=h-averageover;
        if averycord[n]>maxy then
         begin
          maxy:=averycord[n];
          centreloc:= n;
        end;
        n:=n-1;
     end;
     for h:= 0 to averend do
       begin
       ycord[h]:=averycord[h];
       end;
       filend:=averend;
end;

procedure normalize(var ycord:plotarray;filend:integer;
                    var maxy:coord);

        var h:integer;

begin
    for h:= 0 to filend do
    begin
       ycord[h]:=ycord[h]/maxy;
   end;
```

```
   maxy:=1; end;

procedure readfile(var filend,centreloc:integer;
                   var sigma:coord;
                   var stepsize:real;var name:string;
                   var   ycord:plotarray;var   maxy:coord);
{reads data from named file}
         label 10;

         var listend,h:integer;
             checkok:boolean;
             infile1:text;

begin
  10:Writeln('File to be plotted ');
     readln(name);
      checkok:= checkfn(name);
     if checkok=false then goto 10;
     assign(infile1,name);
     reset(infile1);
     readln(infile1,listend,stepsize,sigma);
     filend:= 2*listend;
     h:=0;
     while not eof(infile1) do
       begin
         readln(infile1,ycord[h]);
         if ycord[h]>maxy then
         begin
            maxy:=ycord[h];{find maximum value for y}
            centreloc:=h;
         end;
         h:=h+1;
       end;
      if h-1<> filend then
       writeln('errror in reading file');
      if stepsize<2 then
       averageplot(ycord,filend,centreloc,stepsize,maxy);
       writeln('max y =',maxy);
       normalize(ycord,filend,maxy);
end;

procedure   datatostring(var    result:string;indata:real);
{converts real data to string}

        var dataint,reslength,intlength:integer;
            intpart,fract:string;

begin
     dataint:=round(indata*100);
     str(dataint,result);
     reslength:=length(result);
      if reslength=1 then
      begin
         fract:=result;
```

```
          result:=concat('0.0',fract);
        end;
      if reslength=2 then
        begin
        fract:=result;
        result:=concat('0.',fract);
        end;
        if reslength>=3 then
        begin
        intlength:=reslength-2;
         intpart:=copy(result,1,intlength);
         fract:= copy(result,intlength+1,2);
      result:=concat(intpart,'.',fract);
      end;
end;

procedure setscreen(maxy:coord; filend,mag:integer);
{sets up screen co-ordinates}
      var windowtop,windowright:coord;

begin
      windowtop:= 1.1*maxy;
      windowright:=1.1*filend/mag;
      setwindow(0,windowright,0,windowtop);
      setviewport(0,1,0,0.8);
end;

procedure draw_scale(maxy,sigma:coord;filend,centreloc,
                     mag:integer;stepsize:real);
{draw vertical scales  on sceen}

           var x,centrex,y,stopx,starty,stopy:coord;
               prob:real;
               value,sprob:string;
               nmax,xsigscale,h,locbetweenline:integer;

begin
      stopx:= 1.1*filend/mag;
      stopy:= 1.1*maxy;
      starty:=0.1*maxy;
      for h:= 1 to 10 do
       begin
       newpoly;
        prob:= maxy*h/10;
          y:= starty+prob;
       polypoint(0,y);  polypoint(stopx,y);
       drawpoly;
       datatostring(sprob,prob);
       sprob:=concat(sprob,' ');
       plotstring(1,y-maxy/40,sprob);
      end;
      centrex:= 1.1*centreloc;
      locbetweenline:=round(30/(stepsize*mag));
      nmax:= round(centrex/locbetweenline)-1;
```

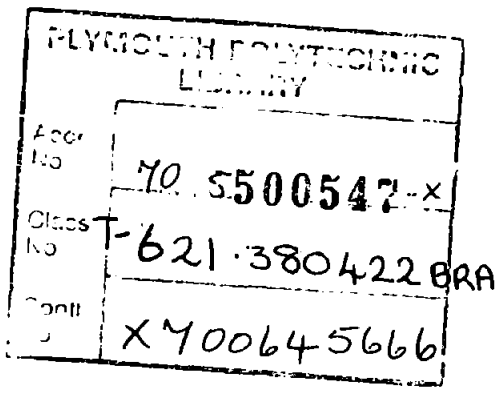
```
      for h:=-nmax to nmax do
       begin
         x:=centrex+locbetweenline*h;
         newpoly;
         polypoint(x,starty); polypoint( x,stopy);
         drawpoly;
         xsigscale:=round(h*locbetweenline*stepsize);
         str(xsigscale,value);
         plotstring(x,starty-maxy/20,value);
      end;
      value:=' deg ';
      plotstring(0.5,0,value); end;

procedure draw_axis(maxy,sigma:coord;filend,centreloc,
                    mag:integer;stepsize:real);
{draw axis for plot}

       var stopx,starty,stopy:coord;


begin
     starty:= 0.1*maxy;
     stopx:=1.1*filend/mag;
     newpoly;
     polypoint(0,starty);   polypoint(stopx,starty);
     stopy:=1.1*maxy;
     polypoint(stopx,stopy);   polypoint(0,stopy);
     polypoint(0,starty);
     drawpoly;
     draw_scale(maxy,sigma,filend,centreloc,mag,stepsize);
end;

procedure draw_data(ycord:plotarray;maxy:coord;
                    filend,mag,centreloc:integer);
{reads data in plot array and plots on workstation}

         var h,short:integer;
             oldx,oldy,startx,starty,x,y:coord;

begin
      h:=centreloc-(centreloc div mag);
      startx:=0.05*filend/mag;
      starty:=0.1*maxy;
      oldx:=startx;
      oldy:=starty;
      while h<filend/mag do
      begin
      short:=0;
       newpoly;
       polypoint(oldx,oldy);
       while short<10 do
         begin
            x:=h+startx;
            y:=ycord[h]+starty;
```

```
            oldx:=x;
            oldy:=y;
            polypoint(x,y);
            short:=short+1;
            h:=h+1;
            if h>= filend then short:=10;
        end;
         drawpoly;
       end; end;

procedure  addtitle(name,title:string;sigma,stepsize:real);
{adds title to plot}

      var sigmas:string;
          Istepsize,isigma:integer;
          echo:boolean;

begin
    echo:=true;
    setwindow(0,1,0,1);
    setviewport(0,1,0.82,1);
    title:=concat('Title = ',title);
    plotstring(0,0.8,title);
    datatostring(sigmas,sigma);
    sigmas:=concat('RMS phase error  = ',sigmas,' degs');
    plotstring(0,0.5,sigmas);
    name:=concat('File= ',name);
    plotstring(0,0.2,name);
end;



procedure plotfile(mag:integer);

       var sigma1,maxy:coord;
           ycord:plotarray;
           title,name:string;
           sigma,stepsize:real;
           filend,centreloc,h:integer;


begin
    openworkstation(1);
    clearworkstation;
    closeworkstation;
    maxy:=0;
    readfile(filend,centreloc,sigma,stepsize,name,ycord,-
             maxy);
    sigma1:= sigma/stepsize;
    writeln('Title ?');
    readln(title);
    openworkstation(1);
    clearworkstation;
    clip;
```

```
    setscreen(maxy,filend,mag);
    draw_axis(maxy,sigma1,filend,centreloc,mag,stepsize);
    draw_data(ycord,maxy,filend,mag,centreloc);
    addtitle(name,title,sigma,stepsize);
end;

procedure plotsingle(mag:integer);

{plots single set of data to screen}

         label 10,20;

       var terminator:char;
           value,status:gsxint;



begin
        plotfile(mag);
        setviewport(0,1,0,1);
     10:plotstring(0.8,0,'MENU ?');
        setgetmode(1,2);
        getvaluator(1,value,terminator,status);
        case  terminator of
         'Y': goto 20;
         'y': goto 20;
         'n':quitread(quit);
         'N':quitread(quit);
         otherwise goto 10;
         end;
20:   closeworkstation;
end;

procedure plotmulti;
{multiple plot program will go in here}

begin
 writeln('multiplot');
end;


procedure magnifx(var magx:string;var mag:integer);
{magnifies area of display around centre}

      var temp:string;

begin
    if magx= 'X1'then
     begin
      temp:='X2';
      mag:=2;
    end;
    if magx= 'X2'then
    begin
```

```
      temp:='X3';
      mag:=3;
    end;
    if magx= 'X3'then
    begin
      temp:='X5';
      mag:=5;
    end;
    if magx= 'X5'then
    begin
      temp:='X10';
      mag:=10;
    end;
    if magx= 'X10'then
    begin
      temp:='X1';
      mag:=1;
    end;
    magx:=temp; end;


procedure menu(var magx,quit:string;mag:integer);
{menu for plot setup }

      label 10,20;

      var value :choice;

begin
      if quit='yes' then goto 20;
   10:writeln;
      writeln('***************************************');
      writeln('     Plot Phase  Noise statistics');
      writeln('          version ',ver:6:2);
      writeln(' CGA version ');
      writeln('***************************************');
      writeln;
      writeln('1) Plot single file');
      writeln;
      writeln('2) Plot Multiple files');
      writeln;
      writeln('3) Magnification X-axis ',magx);
      writeln;
      writeln('0) End ');
      writeln;
      write( 'ENTER ');
      readln(value);
      case value of
      0:quitread(quit);
      1:plotsingle(mag);
      2:plotmulti;
      3:magnifx(magx,mag);
      otherwise goto 10;
      end;
```

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that the copyright rests with its author and that no quotations from the thesis and no information derived from it may be published without the author's prior consent.


PLYMOUTH POLYTECHNIC LIBRARY
Acc No 90 5500547-X
Class No T-621.380422 BRA
Contl No X900645666

```
      if quit='yes' then  goto 20;
      goto 10; 20: end;

begin{main program}
    quit:='no';
    magx:='X1';
    mag:=1;
10: menu(magx,quit,mag);
    if quit='yes' then goto 20;
    goto                                                    10;
20:end.
```