Real time aero engine signal analysis

REAL TIME AERO ENGINE SIGNAL ANALYSIS

by

Bruce Martin Bousfield

A Doctoral Thesis

Submitted in partial fulfilment of the requirments
for the award of
Doctor of Philosophy
of the Loughborough University of Technology

February 1988

Supervisor : Professor J.W.R.Griffiths.
Department of Electronic and Electrical Engineering.

ABSTRACT

For many years the analysis of dynamic signals obtained from aero engine mounted transducers has been performed either by using real time spectrum analysers within the test facility, or by making high quality tape recordings during engine tests and analysing the data via an off-line main frame computer. Although real time analysers produce the information where it is most needed, they provide no history of events and the results are operator dependent. Analysis from tape recordings enables information extraction algorithms to be performed and tables and graphs of notable events to be printed. However much of this information could be more effectively utilised if produced within the test facility and in real time. This thesis describes the design and development of a real time data acquisition, signal processing and information extraction system ideally suited for engine health and performance monitoring within test facilities.

The thesis begins with a detailed description of the problems encountered in dynamic signal analysis in the field of aero engine performance testing, and with an overview of digital signal processing and the latest technology signal processing micro processors that have made this project possible. It then describes the problems encountered and the subsequent solutions found during the design and development of the hardware and software needed for the high bandwidth data acquisition and fast signal processing algorithms.

The fast Fourier transform has been used for very many years in the field of spectrum analysis, however this technique has limitations which are overcome by some of the more modern spectrum estimation techniques. This thesis makes an assessment of some of these techniques, noting particularly their performance on aero engine type signals. The results of these tests are recorded and the possible use of the techniques in aero engine analysis is discussed.

ACKNOWLEDGEMENTS

My sincere thanks and appreciation to the following;

My supervisor Professor J.W.R Griffiths for his help and direction in my work at Loughborough.

Rolls Royce plc. (Derby) and SERC for their sponsorship.

All my collegues in the Electronics and Measurement Techniques department at Rolls Royce - in particular for the guidance and assistance of Peter Penny, Goeff Sleath and Tony Wilson. Many thanks also to Ken Frith without whose insight and determination the project would never have started, let alone succeeded.

A special thanks to my wife Sarah, and my parents Tom and Sheila, for their patience, help and inspiration.

ABRIEVIATIONS and ACRONYMS

a(k) - Autoregressive filter term.
A(k) - Input term of Fourier transform.
A(z) - Autoregressive transfer function.
ADC - Analogue to digital convertor.
AR - Autoregressive.
b - Byte (8-bits).
b(m) - Moving average filter term.
B(z) - Moving average transfer function.
BIO - TMS32010 single line asynchronous input.
CPU - Central processing unit.
DAC - Digital to analogue convertor.
dB - Decibel.
DFT - Discrete Fourier transform.
d.p. - Dual port.
DR - Data Reduction.
e - prediction error.
F(k) - Output term of Fourier transform.
FBLS - Forward-backward least squares.
FFT - Fast Fourier transform.
FPE - Final prediction error.
Fs - Sampling frequency.
H(z) - System transfer function.
HP - High pressure.
Hz - Hertz.
IDDAS - Intelligent Dynamic Data Acquisition System.
IP - Intermediate pressure.
IRRIS - Integrated Rolls Royce Instrumentation System.
K - Kilo (10e3).
LP - Low pressure.
LS - Least squares.
μ - Micro (10e-6)
m - Milli (10e-3)
MA - Moving average.
MEM - Maximum entropy method.
N - Number of samples.
n - Nano (10e-9)

n(k) - Input data sequence term.
p - Number of poles.
P(z) - Power.
PHD - Pisarenko harmonic decomposition.
psd - Power spectral density.
PSLE - Prony spectral line estimation.
r2 - Radix 2.
r4 - Radix 4.
R.R. - Rolls Royce.
Rxx - Auto-correlation function.
S - Residual power (sum of squares).
S/N - Signal to noise ratio.
$\sim$ - Gaussian noise.
Tw - Window time period.
x(n) - Filter output sequence term.
x(t) - Sample in time.
w - Word (16 bits).

W - $e^{-j2\pi/N}$
W(t) - Window sample.

DEFINITIONS

Dynamic data	Digitised data samples acquired from a conditioned transducer signal which has a bandwidth of greater than 5 Hz.
Data reduction	- The act of reducing data bandwidth by extracting specific information from a data sequence.
Deterministic process	- Continuous time function, all values can be predicted at all times.
Leakage	- Smearing of signal energy into neighbouring frequencies caused by windowing the signal.
Picket fence effect	- The ripple in amplitude estimation across a DFT spectrum caused by windowing and sampling a signal.
PDP-11	- Mini-computer manufactured by the Digital Equipment Corporation (DEC). Originally manufactured with a 16-bit backplane, now 22-bit. Has had various standards of CPU and runs various operating systems, the most popular being RT-11.
Q-bus	- 22-bit backplane of the PDP-11.
Stochastic process	- Random function or data sequence, stationary stochastic processes have a gaussian probability distribution.
Test facility	- A facility designed for testing aero engines or parts of. Generally consists of a bed in which the engine sits, and a control room to which the multitude of measured signals are relayed, conditioned, displayed and recorded. Also generally designed for either passing off production engines, or researching and testing development engines.

Preface

TABLE OF CONTENTS

CHAPTER 1

Definition of Problem

1.1 Introduction.

During the design and development of aero engines, many hundreds of engine parameters are required to assess engine health and performance. For many years, aero engine instrumentation has provided the means for measuring, analysing and displaying these engine parameters in calibrated engineering units within the test facilities. In more recent years the instrumentation has progressed from individual analogue driven dial type gauges, to graphically generated dial and bar gauges which can be reconfigured for different tests. This progression has demanded the use of dedicated test facility computers to provide the control and means for acquisition, calibration and display of these parameters. The computer also allows menu driven configuration tables to be set up for different engines and various engine manoeuvres.

The instrumentation computer used in most test facilities is the ubiquitous DEC PDP-11 mini-computer, this machine provides an operating system (RT-11) well suited to real time applications, and instrumentation manufacturers provide a multitude of plug-in cards enabling the acquisition of various signals, from thermocouples to high frequency phonic wheels. The combination of this hardware and R.R. software has produced a versatile instrumentation package (known in Rolls Royce as IRRIS) which provides test facility personnel with real time engine information, and the steady state data recording facility with calibrated engineering parameters.

Although IRRIS has made the task of engine instrumentation more versatile and straight forward it has not really introduced any more information about the engine to the test personnel. Due to the relatively slow update rate of the IRRIS screens (ten times a second) and the fact that no real analysis is performed on the measured data other than calibration, only steady state and some limited transient information is available to the observers. To determine overall engine health, and confidently reschedule engine tests when problems have occured, more information is still required. This information is contained in the signals of higher bandwidths (anything between 5 Hz and 50 KHz) produced by such transducers as accelerometers and strain gauges.

At present this information is obtained by recording the signals on 28 track FM tape, and then at a later date and in a different building, by analysing the tapes via a small general purpose main frame with associated array processors. The information, extracted via spectrum analysis, is then made available to design and development engineers in various printed formats, by this time it is obviously of no practical use to the test facility personnel. This system is shown in figure 1.1

1.2 Extending into dynamic data analysis.

In mid 1984 in became obvious that the capability to analyse and display information extracted from high frequency signals, on test facility instrumentation systems such as IRRIS, would soon become essential. The driving force behind this decision was that of trying to keep engine testing costs to a minimum while maximising the quantity and quality of test results. This came as a direct result of increasing pressures in the aero engine market place.

A requirement thus emerged for a system that could perform signal acquisition, spectrum analysis, data reduction

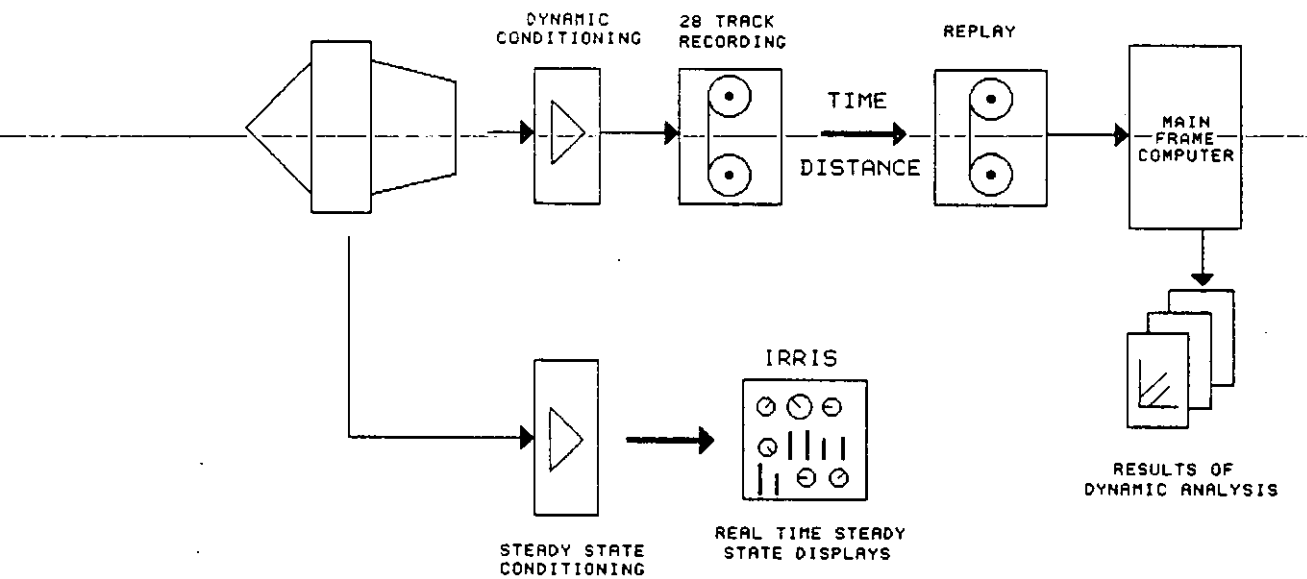


Figure 1.1 - Existing test facility instrumentation

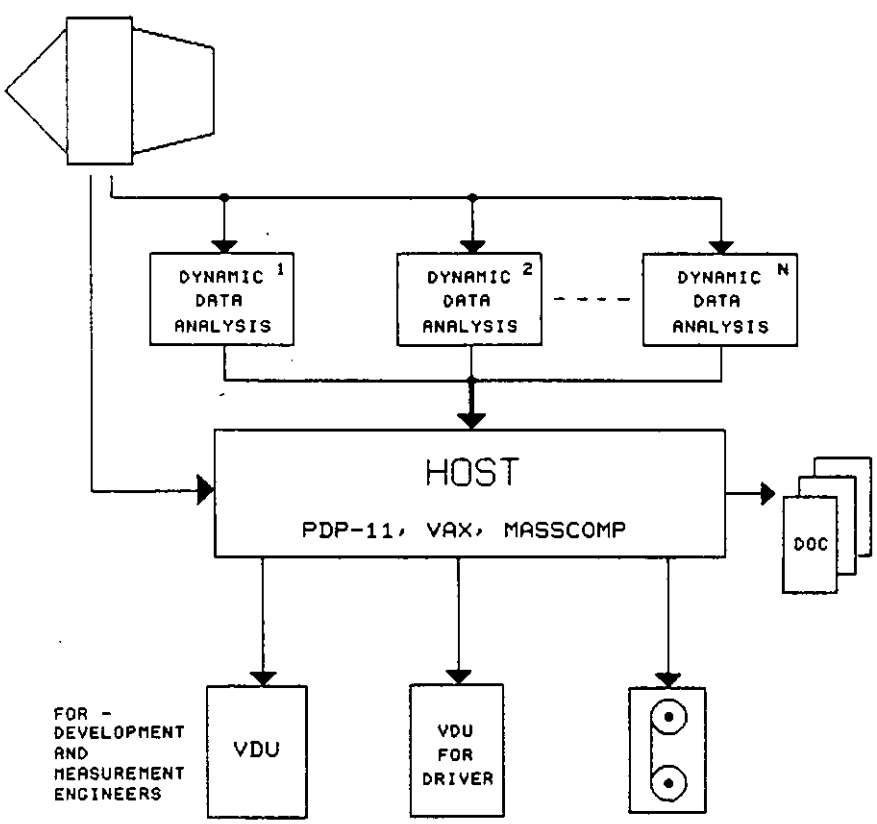


Figure 1.2 - Planned test facility instrumentation

(information extraction), and very importantly be able to communicate with the already existing test facility instrumentation computer. The calibration and display of the dynamic data information also being performed by this computer. It was deemed necessary to have a separate and distinct dynamic data acquisition and analysis system because the computational intensity of spectrum analysis and data reduction algorithms are such that they would bring a general purpose computer, such as the PDP-11 in IRRIS, to a grinding halt. The planned system is shown in figure 1.2.

1.3 Data reduction

One of the desired features of this dynamic data analysis system is the ability to extract information from the high bandwidth input data, thus enabling low bandwidth data streams to be presented to steady state systems such as IRRIS. This type of data reduction can take on many forms, it may simply be to pick out the frequency components with the largest amplitudes, or alternatively and more likely, it may be to track and pick out the amplitudes of various vibration modes. In either case the bandwidth of the resulting information will be significantly lower than that of the original sampled data.

1.4 Engine transducer signals and their spectral content.

Many different types of dynamic signals are received from aero engine instrumentation and in general are produced by three basic types of transducers. These are

1) Accelerometers - used to measure total engine vibration, these are usually mounted on the front and rear of the engine carcass. The signals are usually low in bandwidth (less than 400 Hz) and reveal shaft out-of-balance information.

2) Pressure transducers - used to measure pressure fluctuations around all the aerodynamic surfaces and cavities, e.g. pressures between compressor stages and combustion chamber inlet and outlet pressures. These signals have bandwidths of anything up to 5 KHz and can reveal information about blade passing pressures, compressor surge and stall.

3) Strain-gauges - used to measure blade stress and vibration and indirectly, aerodynamic pressures on the blades. These signals can have bandwidths of at least 25 KHz and reveal information about blade twist and flap, and acoustic resonance.

The spectral content of the signals mentioned above varies between each application. In the case of shaft vibration there may only be two or three sinusoidal components and these would in most cases be quite distinct. However, signals of blade vibration can exhibit twenty or thirty harmonically related engine order components, plus some non-integral components (not-related to engine speed, e.g. blade twist) which cross the other components during engine manoeuvres. The signal to noise ratio of these signals also varies between applications and can be anywhere between 20 and 40 dB.

To cope with the above range of signals the existing dynamic data analysis performed on tape recorded signals requires 1024 point DFTs with spectrum bandwidth capabilities of at least 50 KHz. This type of performance must be replicated in any system introduced into the test facility environment.

1.5 Post analysis and data reduction.

Dynamic data analysis and the subsequent data reduction algorithms performed on tape recordings are as varied as the signals themselves. However, to provide some insight into the

range of analysis techniques which are performed, two brief examples are given;

1) Demonstrating the fact that in many cases the exact information required is not known (this is typical in development engine analysis), a clear graphical representation of all of the information is produced. This is achieved by using density plots where time is in the x-axis, frequency is in the y-axis and the spectrum amplitude is represented by image density. An example of this is shown in figure 1.3, this clearly picks out five engine orders plus a non-integral component during an engine acceleration. The raw signal for this analysis has come from a pressure transducer mounted by the main compressor fan.

2) A case where the information details are clearly defined is in the measurement of engine vibration. The main vibration components are found at the fundamental shaft rotation frequencies. Thus vibration measurement simply consists of picking out the amplitudes of the components at these frequencies, the rest of the data can then be discarded. An example of this is shown in figure 1.4 where the vibration components of a three shaft engine (RB211-535E4) have been tracked and plotted against engine speed. The raw signal has come from an accelerometer mounted on the engine carcass while the engine was subjected to a two minute acceleration and a two minute deceleration.

1.6 Real time dynamic data analysis system specification.

The above brief outline concerning the requirement of real time signal processing within aero engine test facilities resulted in the following specification being derived in late 1984;

A system is required that can be installed into test facility environments and be able to perform the following tasks

1) Acquisition of signals directly from transducer conditioning units.

2) The acquisition hardware should perform all filtering, sampling and digitising.

3) Acquisition rates should be variable and allow signal bandwidths of up to 50 KHz.

4) Perform various signal processing algorithms, in particular a 1024 point Fourier transform.

5) Perform various data reduction algorithms, some of which will inevitably involve knowledge of engine speed thus making measurement of speed also a requirement.

6) Fast execution of the above two tasks, thus indicating the need for dedicated signal processing hardware.

7) The ability to communicate at high speed with the PDP-11 CPU.

8) Allow hardware and software configuration under the control of the PDP-11 application programs.

9) Allow multi-channel inputs for cross-correlation purposes.

10) Calibration of the system and host should be straight forward and only necessary after lengthy periods (e.g. 3-6 months)

11) The cost should not be prohibitive when considering that some facilities may require 28 or more channels of analysis.

1.7 Assessment of the analysis system specification.

There are only really three possible ways of performing the above task of dynamic data aquisition, analysis, data reduction and communication with the host. These three options are 1) using readily available spectrum analysers, 2) using a general purpose computer with an associated array processor and acquisition hardware, and 3) using a DEC compatible card specifically designed to perform these tasks.

Spectrum analysers contain all the necessary hardware to perform data acquisition and spectrum analysis, and they can usually be communicated to, and controlled by, a host computer via the IEEE network. However their data reduction functions are usually basic, their communication bandwith low, and are generally quite bulky units. Spectrum analysers with anything like the sort of processing power required retail for at least £ 10k. which is rather prohibitive.

A general purpose computer using additional acquisition hardware and array processors or maths accelerators to perform the signal processing and data reduction would meet the technical specification. Indeed this is the type of equipment that is used to perform the post analysis on tape recordings of engine signals. Systems of this nature however, have the same two problems as the spectrum analysers, they are prohibitively expensive (at least £10k. per channel) and are excessively large for the limited amount of room available in most test facilities.

The only real solution is a system comprised of one or more PDP-11 compatible plug in cards capable of performing the necessary data acquisition, the dynamic data analysis and the communication with a PDP-11 CPU over the PDP-11 backplane. Only one major problem existed with this solution, at the time when this specification was derived the necessary hardware to do the job did not exist and did not look like coming onto the market for a good deal of time.

Once again, as happens quite regularly at Rolls Royce, the instrumentation requirements needed to test and analyse aero engines had outgrown that for which the instrumentation market could supply. It was for this reason that research, design and development into a real time dynamic data acquisition and analysis system was embarked upon. Due to the limited amount of resource which can be applied to such projects, the entire research, design and development of the system was the sole responsibility of the author.

The following chapters describe how the above specification was turned into a fully working productionised system. This work involved a considerable involvement with digital signal processing techniques, especially the fast Fourier transform, and with the Texas TMS32010 digital signal processing micro-processor. Examples are given in later chapters of how, after two and a half years work, the productionised system was first put to use in both development and production engine test facilities. The final productionised system was called the "Intelligent Dynamic Data Acquisition System" or IDDAS.

1.8 Assessment of modern spectrum analysis techniques.

In certain circumstances the sinusoidal components within a signal either change in frequency or amplitude very quickly during a 1024 samples period, or are very close in frequency to the extent of being indistinguishable. This can lead to inaccurate and misleading results which can only be overcome by making the sample blocks smaller and increasing the resolution. It is of course impossible to do both of these simultaneously with the Discrete Fourier transform.

Over the years many alternative spectrum analysis techniques have been derived. A few of these have been used in real applications (e.g. geophysics) but in general they have been restricted to academic applications. The author has spent

part of his research time (approximately one fifth) in assessing these alternative techniques, and in applying the more relevant ones to aero engine type signals in an attempt to overcome the above problems. This work has been performed on the Loughborough University main frame computer and no attempt has been made towards real time application of these techniques. This work is described in the later chapters of this thesis.

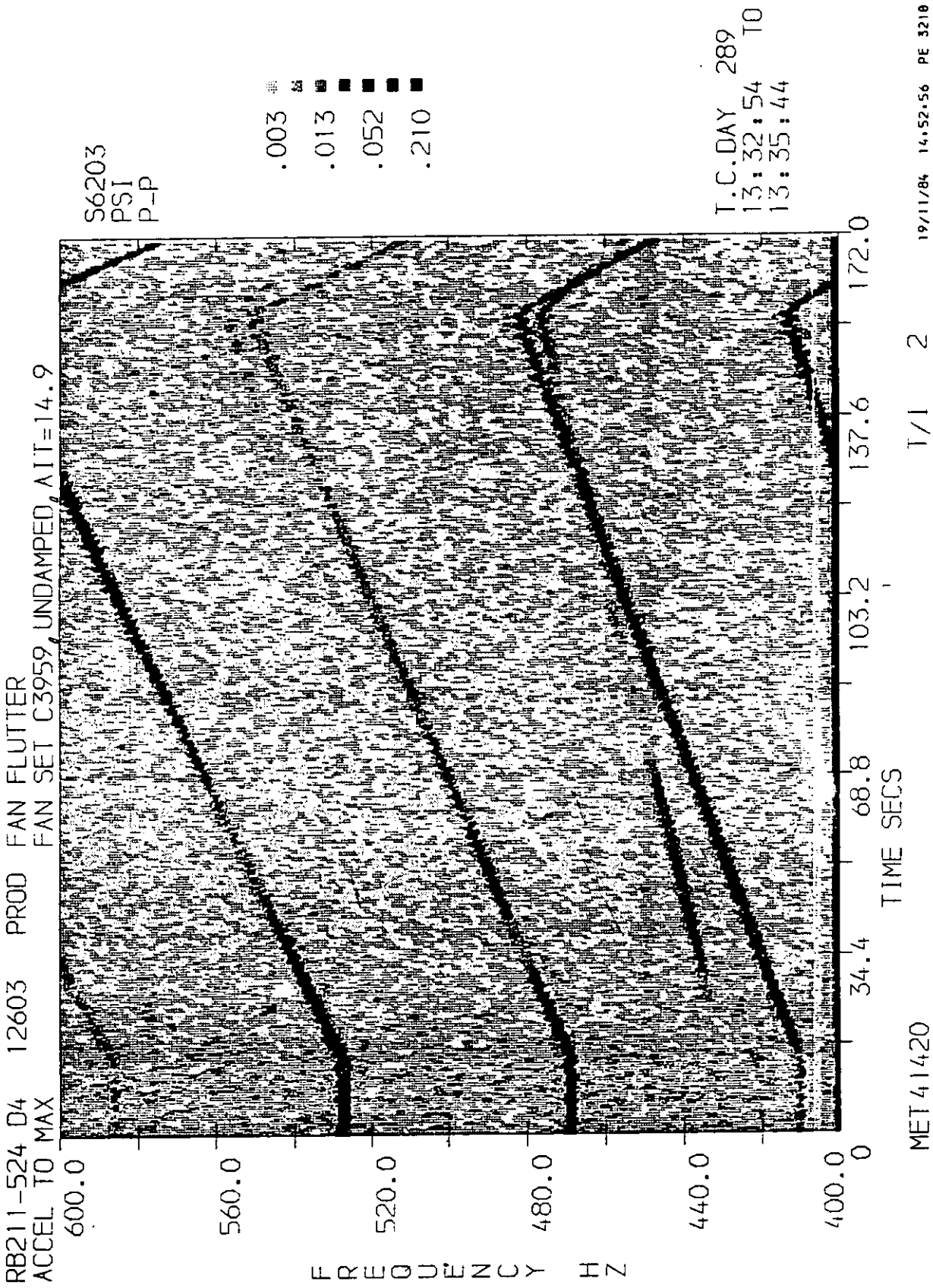


Figure 1.3 - Spectrum density plot produced by Rolls Royce data reduction system from a pressure transducer signal

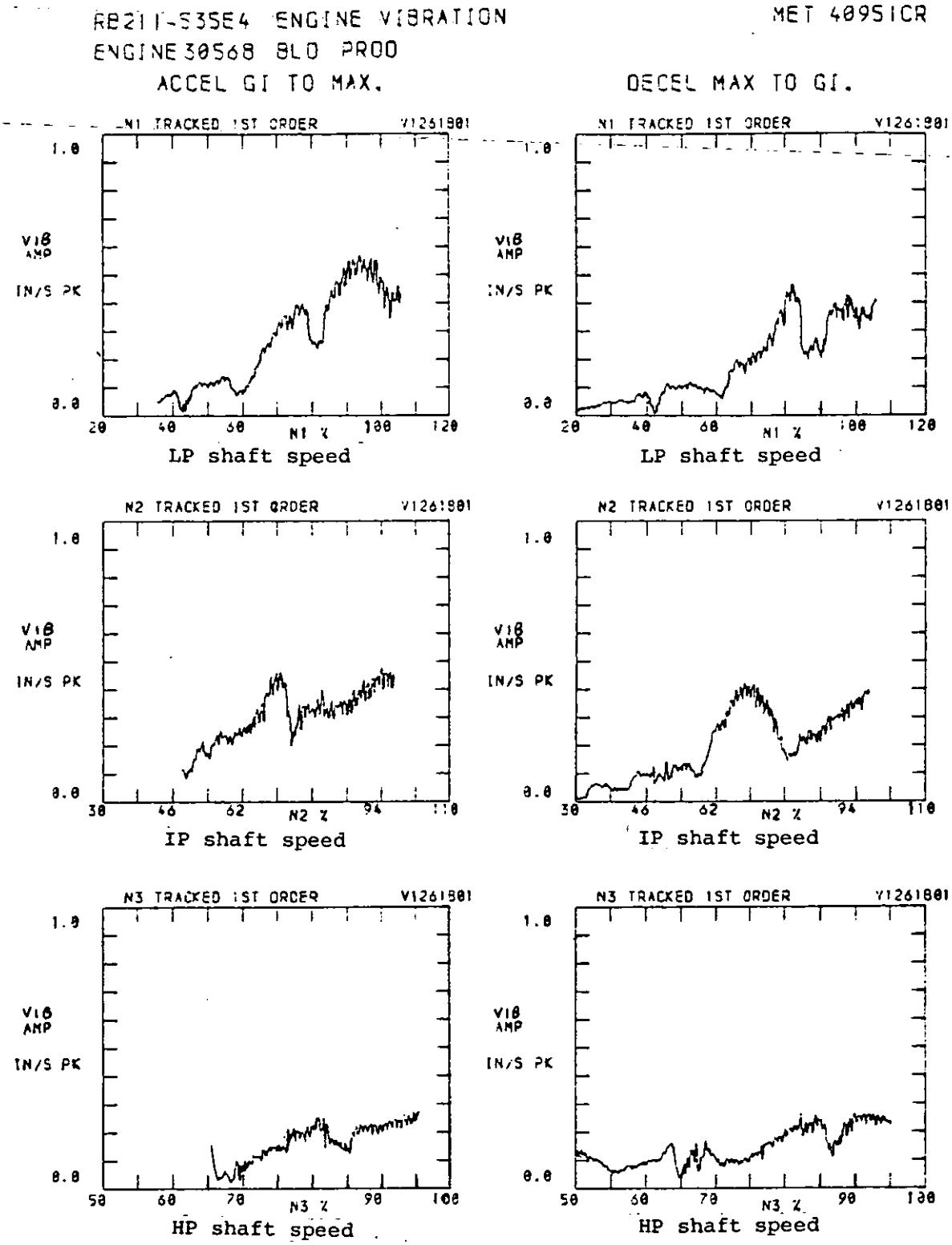


Figure 1.4 - Tracked vibration amplitudes for a three shaft engine during a two minute acceleration and two minute deceleration. Velocity signals taken from an engine carcass mounted accelerometer.

CHAPTER 2

Digital Signal Processing

2.1 Digital signal processing fundamentals.

As with analogue systems, a linear discrete system is one which obeys the principle of superposition, and a time invariant system is one in which the input to output transformation algorithm does not change with time. Linear time invariant systems account for a wide variety of signal processing functions and are the most straight forward to analyse.

To perform signal processing with the aid of digital computers the input data must be represented as discrete values and in a format that a computer can understand, i.e. sampled and digitised into binary format. Certain precautions must be undertaken when sampling a signal at discrete linear time intervals, as not all the information in the signal can be precisely recovered. The information that can be recovered is described in Shannon's sampling theory;

A signal with no frequency components greater than Fs/2 can be uniquely defined by its instantaneous values when sampled at Fs or more.

Frequency Fs is known as the Nyquist sampling rate. Frequency components which occur above half this sampling rate are aliased (transformed) down to a frequency below half the sample rate, these components then become indistinguishable against the true lower frequency components, this is demonstrated in figure 2.1.

To avoid confusion between real components and aliased

components an anti-aliasing filter must be employed prior to sampling a signal to remove all signal components above Fs/2 Hz. Ideally this filter would be a brick wall low pass filter with a cut-off frequency at Fs/2 Hz. Realistically this of course is not possible and the filter would normally have a 3 dB cut off point of about 0.4Fs Hz.

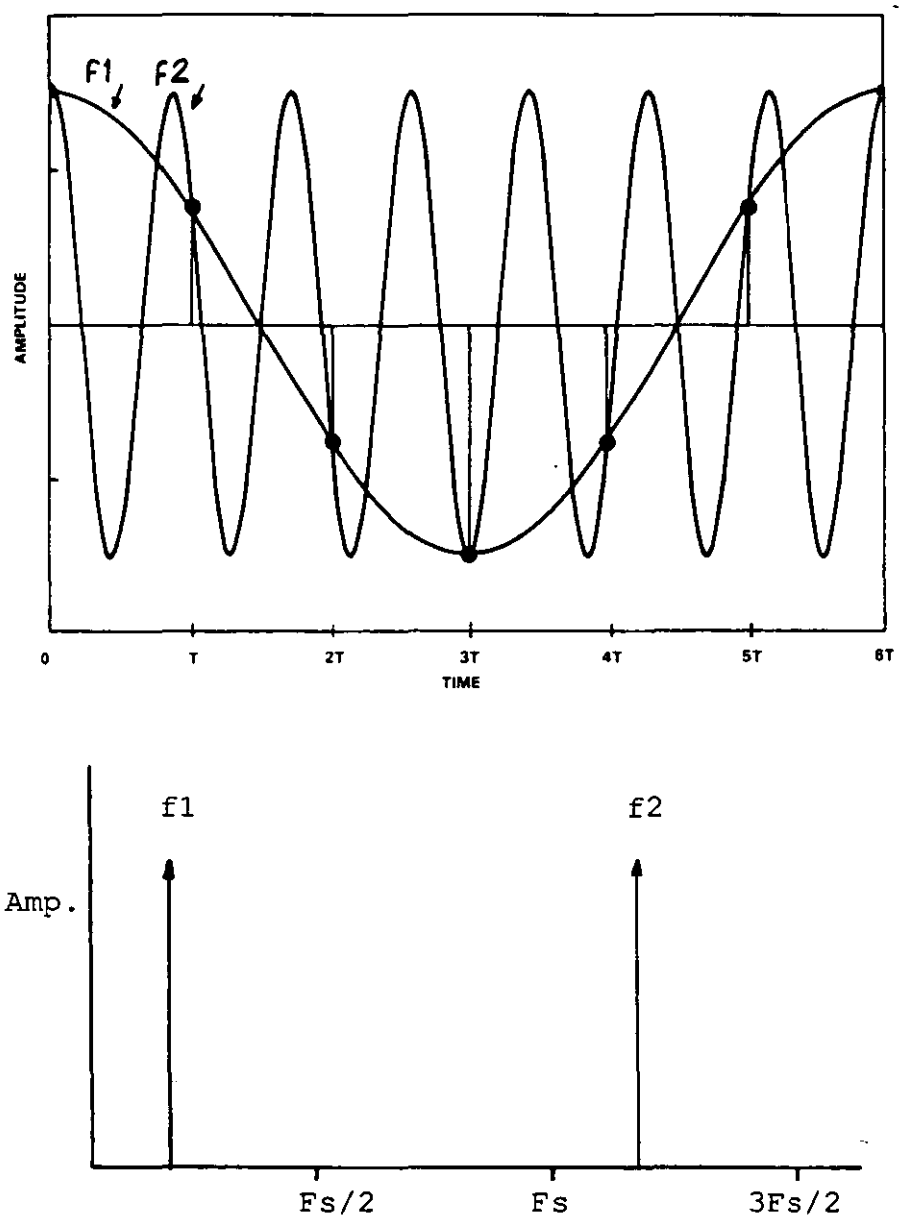


Figure 2.1 - Effects of aliasing

Having set the upper frequency limit and sampled the signal, the discrete value must be converted into a digital (binary) form that can be understood by a computer. The precision to which a signal is digitised depends upon the total number of discrete levels that an instantaneous value can be assigned. This number also defines the digitising signal to noise ratio that occurs due to quantisation errors. This error and its corresponding S/N ratio is shown in figure 2.2.

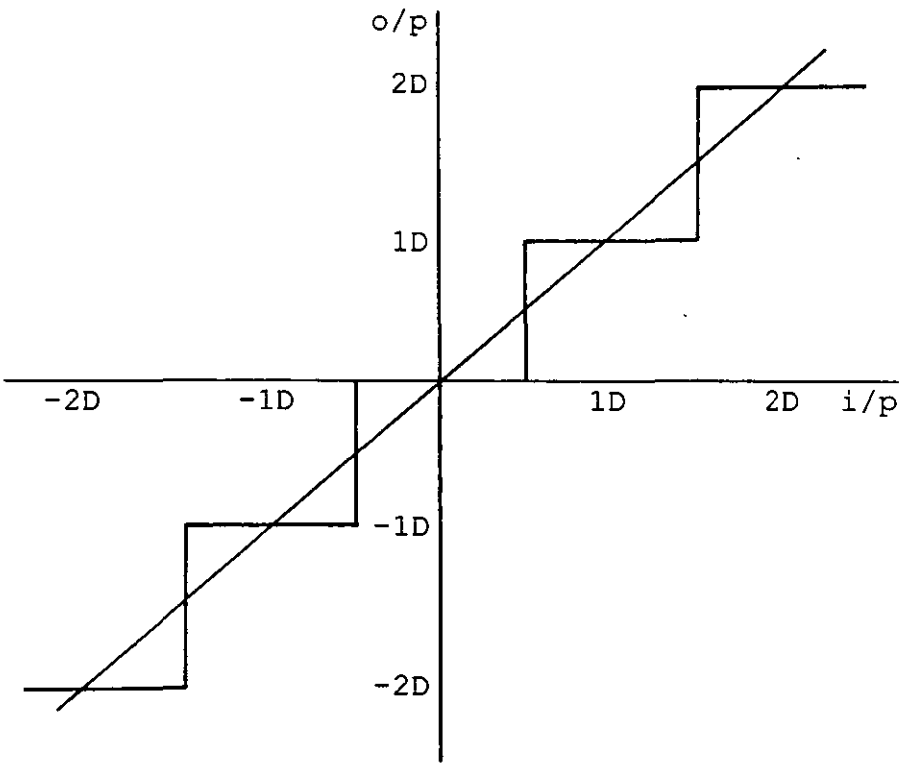


Figure 2.2 - Digitising error

The error (e) between actual signal (i/p) and digitised signal (o/p) is -

$$-D/2 < e < D/2$$

Noise power $\Rightarrow \quad 1/D \int_{-D/2}^{D/2} e^2 . de = D^2/12$

Signal power $\Rightarrow \quad (E/2\sqrt{2})^2 = E^2/8$ where E = p.p. amp. of signal

Signal/Noise => $(E^2/2).(3/D^2)$

However $D = E/2^N$, where N equals number of bits, thus

$$S/N = 3.2^{2N-1} = 4.77 + 3(2N-1)\ dB.$$
$$= 1.77 + 6N\ dB$$

This formula gives the following example values -

N	No. of levels	S/N (dB)	
8	256	50	Note that this is only valid for sinusoidal signals
12	4096	74	
16	65536	98	

The hardware considerations that must be made in the light of the above information are discussed in chapter 4.

2.2 Basic signal processing algorithms.

The following two techniques, correlation and convolution, are at the heart of most signal processing algorithms. The third technique described, the fast Fourier transform, is almost certainly the most highly used of any signal processing algorithm in the engineering and research fields.

2.2.1 Correlation.

The correlation integral of a linear time invariant system is defined as

$$Rxy(T) = \int_{-\infty}^{\infty} fx(t).fy(t-T).dt \qquad -\infty < T < \infty$$

In the above case there are two different time functions and thus is refered to as the cross-correlation integral. More often there is only one time function and the integral is modified to become the auto-correlation function

$$Rxx(T) = \int_{-\infty}^{\infty} fx(t).fx(t-T).dt \qquad -\infty < T < \infty$$

This function has the special property, known as the Wiener Khintchine relationship, in that the double sided energy spectrum of the same time function is its Fourier transform pair. More specifically, when T=0 the auto-correlation function is equal to the energy contained within that signal.

2.2.2 Convolution.

The output of a linear time invariant system is obtained by repeated evaluation of the convolution integral

$$Y(T) = \int_{-\infty}^{\infty} h(t).f(T-t).dt \qquad -\infty < T < \infty$$

where h(t) is the impulse response of the system and f(t) is the input signal. Note that the only difference between this and the cross-correlation function, is in the time reversal of f(t). For discrete systems the convolution integral is slightly modified to produce the convolution sum

$$Y(k) = \sum_{n=-\infty}^{\infty} h(n).x(k-n) \qquad -\infty < k < \infty$$

where h(n) is the unit sample response of the system and x(n) is the input data sequence. This convolution sum can be used in many types of linear discrete systems, in particular, the discrete Fourier transform and windowing, as will be described.

2.3 Fourier analysis.

As stated earlier, the main real time signal processing task required for engine analysis within engine test facilities is spectrum analysis. The easiest and most common way of performing this is via the fast Fourier transform (FFT) [5, 17, 23, 47] and it is by this technique that all engine data post analysis is currently performed. The FFT is a well documented, tried and tested algorithm whose behaviour can be predicted with high confidence.

The following sections give a brief description of how the very popular radix 2 FFT is derived, and of how it can be made still faster by using radix 4 and real inputs only.

2.3.1 Discrete Fourier transform.

The Fourier transform for continuous signals is defined as follows

$$X(f) = \int_{-\infty}^{\infty} x(t).e^{\frac{-j2\pi t}{T}}.dt \qquad \text{where } f=1/T$$

and the complementary discrete Fourier transform (DFT) for sampled finite records is

$$F(k) = \sum_{n=0}^{N-1} x(n).e^{-j2\pi.\frac{nk}{N}} \qquad k=0,1,..,N-1$$

The DFT output is thus a set of N samples taken from the continuous Fourier transform. The fundamental frequency of the samples being Fs/N Hz, where Fs is the sampling frequency, and each sample frequency is given by Fs.k/N Hz. These frequencies do not explicitly appear in the DFT summation however "k" can be interpreted as a harmonic number and "n" as the sample period number. Note also that when the input data sequence is purely real, as is usually the case for linear time samples, the output is a double sided spectrum.

2.3.2 Fast Fourier transform.

Over many years the FFT has proved to be a very efficient and useful algorithm, especially when used in computer programs, and has found applications in many tasks other than just spectrum analysis. For example, in some cases it is faster to do a convolution by performing two FFT's, a multiplication of the resulting spectra and then an inverse FFT. "Fast Fourier transform" is really a generic term for a multitude of slightly different algorithms all derived from the DFT, and designed to significantly reduce the computational effort involved in the DFT. The underlying similarity being that they all break down the DFT into small common blocks by utilising the periodicity and symmetry of the exponential term, thus eliminating a great deal of repetitive complex multiplication. The saving in computational effort is usually refered to as being proportional to [N**2/N.Log((N)], where "N" is the number of samples. It will be shown however that the saving in real application programs is even greater than this.

2.3.3 Derivation of radix 2 FFT.

For the sake of convenience (especially for the typist) the following definition will be used

$$e^{\frac{-j2\pi.nk}{N}} = W(nk)$$

The DFT summation shown above, can now be rewritten as

$$F(k) = \sum_{n=0}^{N-1} x(n).W(nk) \qquad k=0,1,..,N-1$$

If N is even, then the even and odd terms can be split as follows

$$F(k) = \sum_{n=0}^{\frac{N}{2}-1} [x(2n).W(2nk) + x(2n+1).W((2n+1)k)]$$

$$\Rightarrow F(k) = \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x(2n).W(2nk)}_{A} + W(k) \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x(2n+1).W(2nk)}_{B}$$

i.e. $F(k) = A(k) + W(k).B(k)$

where A(k) and B(k) are N/2 point transforms. If "k" is now replaced with "k + N/2" we have

$$F(k+N/2) = A(k+N/2) + W(k+N/2).B(k+N/2)$$

However A and B are N/2 point transforms where

$$A(k+N/2)=A(k),$$
$$B(k+N/2)=B(k),$$
$$\text{and } W(k+N/2)=-W(k)$$

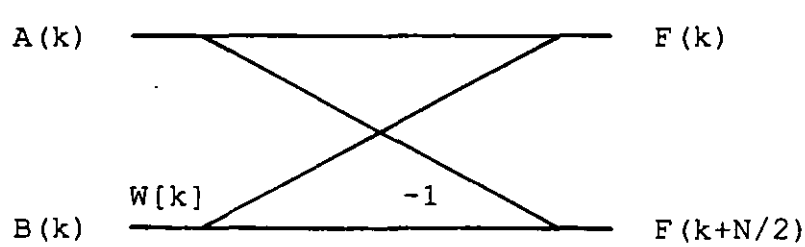
Inserting these equates into the above expression gives

$$F(k+N/2) = A(k) - W(k).B(k)$$

The original N point DFT algorithm has now been split into two N/2 point DFT's plus the overhead of the following set of equations

$$F(k) = A(k) + W(k).B(k) \quad k=0,1,..,N/2-1$$
$$\text{and } F(k+N/2) = A(k) - W(k).B(k)$$

The operation formed by combining these equations is known as a butterfly and can be graphically represented as follows



This operation involves one complex multiply and two complex additions, the arrangement of these butterflies in a 16 point Fourier transform is demonstrated in figure 2.3.

The rearrangement of the input data sequence into the transform exhibits an interesting pattern which is known as bit reverse mapping. It is thus called because if each consecutive input sample assumes an address of 0 to N-1, then its corresponding address into the transform if found by reversing the binary version of that address. For example, refering to figure 2.3, the input address of A3 in binary is 0011, the corresponding bit reversed address is then 1100. Thus at the beginning of the transform A3 is positioned where A12 is and subsequently A12 positioned where A3 is.

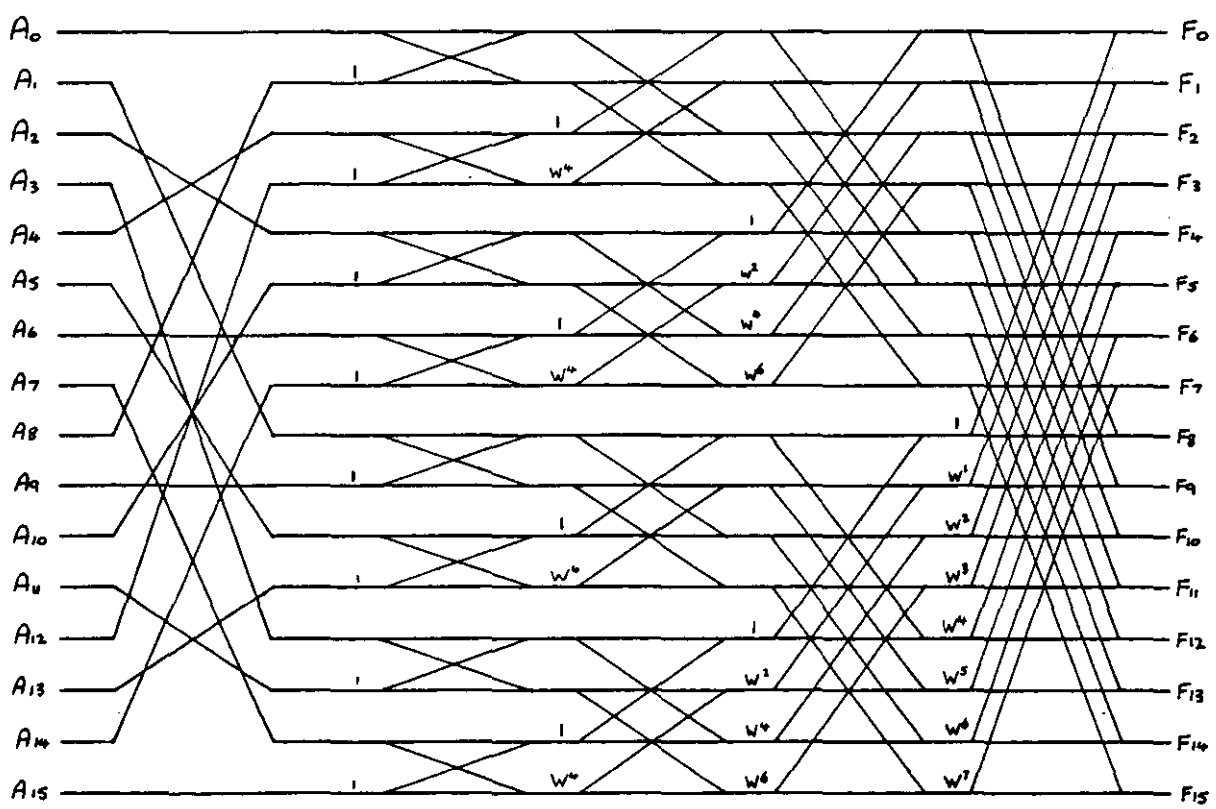


Figure 2.3 - 16 point, radix 2 FFT

2.3.4 Computational saving of radix 2 FFT.

It is now possible to demonstrate the computational saving that the FFT has over the DFT, note that in the following assessment a real multiply operation is represented by "M", a real addition by "A", and that there are "N" samples. It is also assumed that a multiply and an addition take equal times to execute (this is true for the TMS32010 micro-processor).

DFT computation : Each point requires N complex multiplies and N complex additions. Total of N points.

$$\Rightarrow \quad N.[N(4M+2A) + N.2A] = 8N^2 \text{ operations}$$

FFT computation : Each butterfly requires 1 complex multiply and 2 complex additions. There are N/2 butterflies in each column and Log2(N) columns.

$$\Rightarrow \quad \text{Log2(N)}.N/2.[(4M+2A) + 2.2A]$$
$$= 5N.\text{Log2(N)} \text{ operations}$$

However to be more precise, there are no complex multiplies in the first two columns, thus it can be modified to

$$\Rightarrow \quad \text{Log2(N)}.N/2.(4M+6A) - 2.N/2.(4M+2A)$$
$$= 5N.\text{Log2(N)} - 6N \text{ operations}$$

Applying these two expressions to a 1024 point transform the following values are obtained

DFT : 8388608 operations
FFT : 45056 operations

the ratio of computation effort (DFT:FFT) being 186:1.

2.4 Radix 4 FFT.

The analysis and factorisation of the discrete Fourier transform into a radix 2 fast Fourier transform, as shown above, is the most straight forward and easiest to understand of all FFT derivations. As a result, the FFT is almost always documented and introduced in its radix 2 form (as has been done here!), thus resulting in almost exclusive use of the radix 2 FFT in computer programs. However, just as the DFT can be broken down into butterflies of 2 inputs and outputs, it can also be broken down into butterflies of 4 inputs and outputs. As will be demonstrated the radix 4 FFT is significantly more efficient than the radix 2 FFT.

2.4.1 Derivation of Radix 4 FFT.

As with radix 2 (r2), the radix 4 (r4) derivation breaks down the weighted summations of the DFT into smaller groups of summations by taking advantage of the periodices and symmetry of the DFT. Again we start with the definition of the DFT

$$F(k) = \sum_{n=0}^{N-1} x(n).W(nk) \qquad k=0,1,..,N-1$$

Factorising into four equal length parts we get

$$F(k) = \sum_{n=0}^{\frac{N}{4}-1} x(4n).W(4nk) + \sum_{n=0}^{\frac{N}{4}-1} x(4n+1).W((4n+1)k)$$

$$+ \sum_{n=0}^{\frac{N}{4}-1} x(4n+2).W((4n+2)k) + \sum_{n=0}^{\frac{N}{4}-1} x(4n+3).W((4n+3)k)$$

$$F(k) = \underbrace{\sum_{n=0}^{\frac{N}{4}-1} x(4n).W(4nk)}_{A0} + W(k) \underbrace{\sum_{n=0}^{\frac{N}{4}-1} x(4n+1).W(4nk)}_{A1}$$

$$+ W(2k) \underbrace{\sum_{n=0}^{\frac{N}{4}-1} x(4n+2).W(4nk)}_{A2} + W(3k) \underbrace{\sum_{n=0}^{\frac{N}{4}-1} x(4n+3).W(4nk)}_{A3}$$

Thus

$$F(k) = A0(k) + W(k).A1(k) + W(2k).A2(k) + W(3k).A3(k)$$

This expression is true for any k, and in particular is true for [k+r.N/4], thus giving

$$F(k+r.N/4) = A0(k+rN/4) + W(k+rN/4).A1(k+rN/4) + W(2(k+rN/4)).A2(k+r.N/4) + W(3(k+rN/4)).A3(k+r.N/4)$$

But

$A0(k+r.N/4) = A0(k)$
$A1(k+r.N/4) = A1(k)$
$A2(k+r.N/4) = A2(k)$
$A3(k+r.N/4) = A3(k)$

These are outputs from an N/4 point transform, hence are periodic in N/4.

and

$$W(rN/4) = e^{-j\frac{\pi}{2}r} = (-j)^r$$

$$W(2rN/4) = e^{-j\pi r} = (-1)^r$$

$$W(3rN/4) = e^{-j\frac{\pi}{2}3r} = (+j)^r$$

Letting r=0, 1, 2, 3

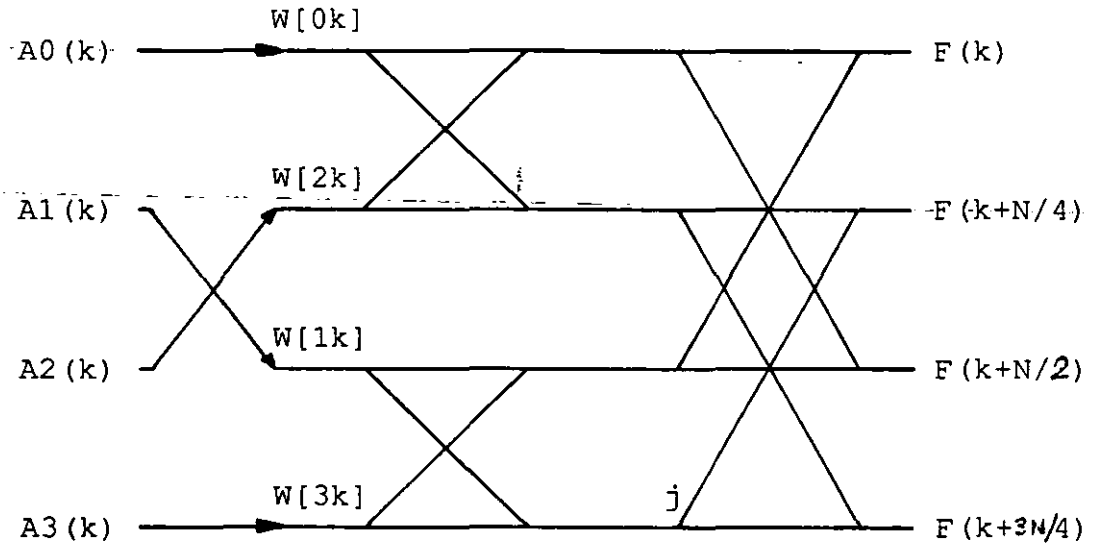
$$F(k) = A0(k) + W(k).A1(k) + W(2k).A2(k) + W(3k).A3(k)$$

$$F(k+N/4) = A0(k) - jW(k).A1(k) - W(2k).A2(k) + jW(3k).A3(k)$$

$$F(k+2N/4) = A0(k) - W(k).A1(k) + W(2k).A2(k) - W(3k).A3(k)$$

$$F(k+3N/4) = A0(k) + jW(k).A1(k) - W(2k).A2(k) - jW(3k).A3(k)$$

As with the r2 butterfly we can now construct a graphical representation of these equations.



2.4.2 Combining Radix 4 Butterflies.

Having developed the basic building block for an r4 FFT a larger matrix can be built to show how the inputs, outputs and exponentials are arranged. Figure 2.4 shows this for a 16 point transform. Each 4 point transform takes it inputs from the outputs of the preceeding 4 point transforms. Note that the initial inputs are also rearranged via a modulo 4 address reversal (as compared to modulo 2 for the r2 FFT).

It can be seen from figure 2.4 that the r4 butterflies still involve rotation of the two middle inputs before being multiplied by W[k] and W[2k]. In order to simplify the matrix, and thus also of the programming, the rotating of butterfly inputs can be moved to the very front of the matrix. This is demonstrated in figure 2.5 together with the expansion of the first four 4 point transforms. It can be seen that the rearrangement of the initial inputs is now modulo 2 bit reversal. It can be shown that the construction of this more simplified matrix is general to any 4**n sized FFT.

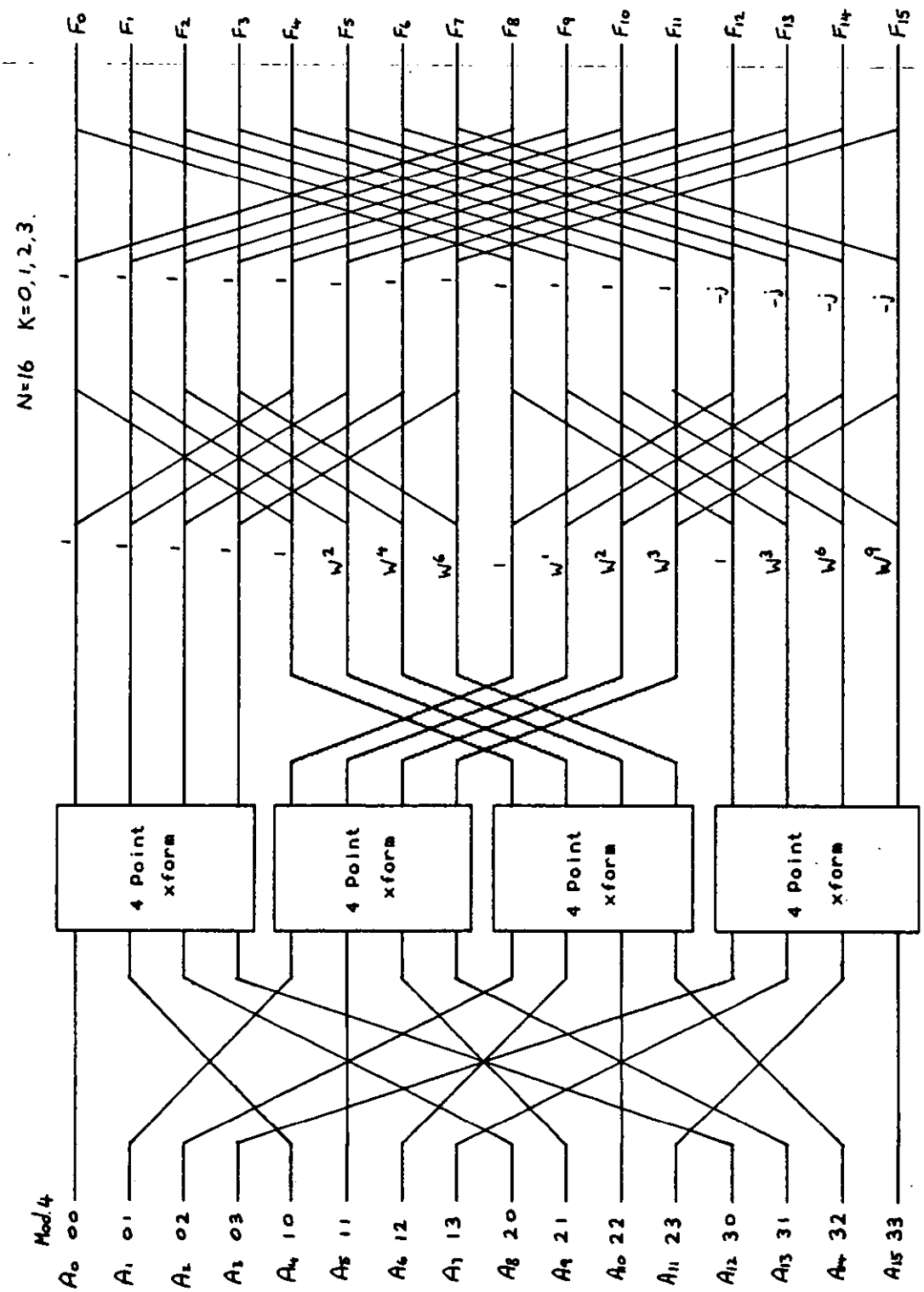


Figure 2.4 - 16 point, partly expanded radix 4 FFT

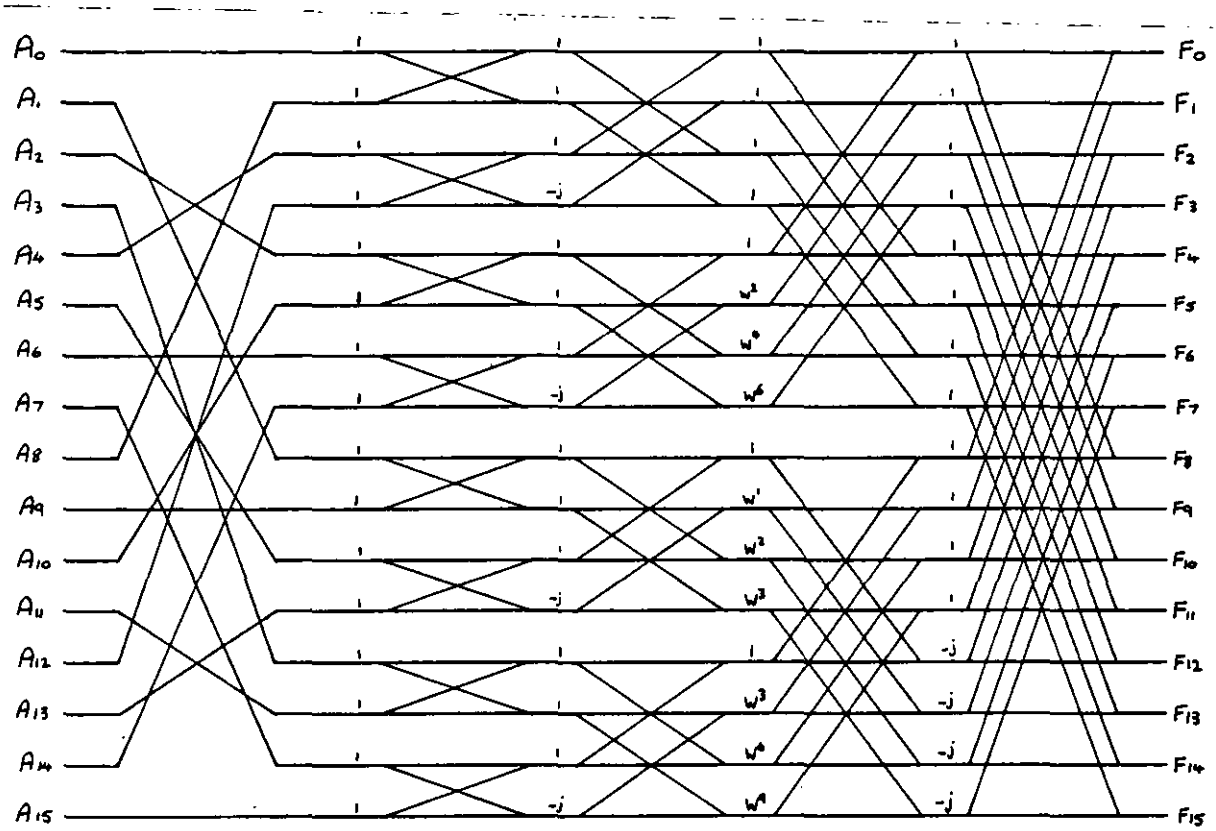


Figure 2.5 - 16 point, radix 4 FFT

2.4.3 Computational saving of a radix 4 FFT.

To compute the radix 4 butterfly the following operations must be performed

Temp1 = W(k).A1(k), Temp2 = W(2k).A2(k), Temp3 = W(3k).A3(k)
Temp4 = A0(k) + Temp2, Temp5 = A0(k) - Temp2
Temp6 = Temp1 + Temp3, Temp7 = Temp1 - Temp3

F(k) = Temp4 + Temp6 F(k+N/2) = Temp4 - Temp6
F(k+N/4) = Temp5 - jTemp7 F(k+3N/4) = Temp5 + jTemp7

The above computation requires a total of 3 complex multiplies and 8 complex additions thus giving a total of 12 real multiplies and 22 real additions. In a complete FFT where there are N/4 butterflies in each column, and Log2(N)/2 columns, then the number of operations would be

=> log2(N)/2.(N/4).(12M + 22A)

If it is also taken into account that there are no multiplies in the first column then this is modified to

=> log2(N)/2.(N/4).(12M + 22A) + (N/4).(16A-12M-22A)
= N.log2(N).(1,5M + 2,75A) - N.(3M + 1,5A)

= 4,25N.Log2(N) - 4,5N operations.

As shown earlier, the number of operations for an r2 FFT is

5N.Log2(N) - 6N operations.

Comparing the number of arithmetic operations to perform a 1024 point transform via the r2 and r4 FFT's, the following values are obtained

Radix 2 -- 45056 operations
Radix 4 -- 38912 operations

Thus the r4 FFT only requires 86% of the r2 FFT operations, this is a fair reduction in operations, and it will be shown later that when coded up for the TMS32010 microprocessor the benefits are increased further due to there also being less memory transfers.

2.5 Real input FFT.

When using the discrete Fourier transform to convert data from the time to the frequency domain, the input sequence is purely real. Thus data is only put into the real components of the complex input array and the imaginary components are initialised to zero. The result of this real input transform is a double sided spectrum where the information from 0 to Fs/2 is repeated from 0 to -Fs/2. This double sided spectrum has properties which allow more efficient use to be made of the transform by putting input data into both components of the complex array. It should be remembered that the FFT derivations shown so far are complex input algorithms. An analysis of this real input technique now follows [6].

2.5.1 Properties of a double sided spectrum.

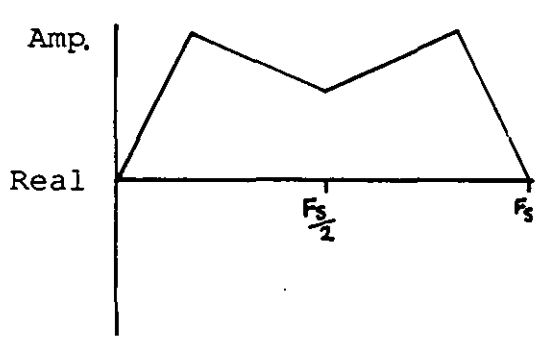
When real data is used as the input into a complex input DFT, i.e. by setting the imaginary components to zero, a double sided-spectrum is produced at the output. The real components of this spectrum are even in nature, and the imaginary components are odd (both centred around Fs/2), as demonstrated by the following relationships

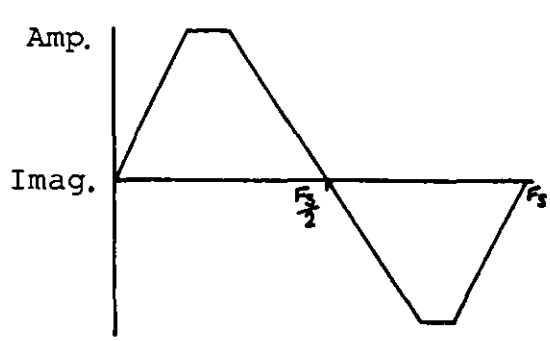
$$F = DFT(U) \qquad U \text{ is real,}$$

$$Real[F(n)] = Real[F(N-n)] \qquad 1 < n > N-1$$

$$Imag[F(n)] = -Imag[F(N-n)]$$

$F(0)$ and $F(N/2)$ are purely real.

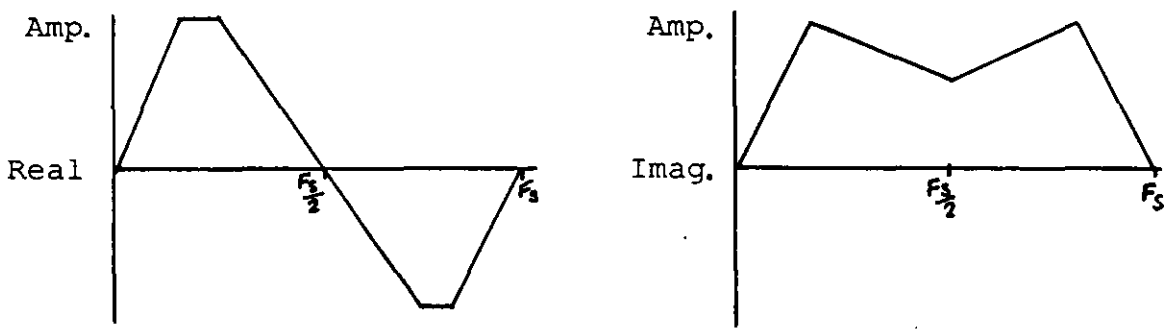




If the real data is put into the imaginary components instead of the real components (and the real components subsequently set to zero) then the double sided spectrum which is produced displays slightly different characteristics to that above. In this case the real components are odd in nature and the imaginary components are even, giving the following relationships

$$G = DFT(j.U) \qquad U \text{ is real,}$$
$$Real[G(n)] = -Real[G(N-n)] \qquad 1 < n > N-1$$
$$Imag[G(n)] = Imag[G(N-n)]$$

G(0) and G(N/2) are purely imaginary.



2.5.2 Derivation of a real input FFT.

The relationships between components within double sided spectrums, as described above, can be used to good advantage. Due to the slight differences between the purely real input DFT and the purely imaginary input DFT it is possible to incorporate two separate sets of data into a single DFT and extract from the resulting spectrum the two double sided spectra that would have been produced by analysing each set of input data separately. If one set of data [U] is put into the real components and another set [V] is put into the imaginary components then the two double sided spectra can be extracted from the DFT output as follows

$$A = DFT[U]$$
$$B = DFT[j.V]$$
$$F = DFT[U+j.V]$$

$$\begin{aligned}
Real[A(n)] &= Real[F(n)+F(N-n)]/2 \qquad 1 < n > N-1 \\
Imag[A(n)] &= Imag[F(n)-F(N-n)]/2 \\
Real[B(n)] &= Real[F(n)-F(N-n)]/2 \\
Imag[B(n)] &= Imag[F(n)+F(N-n)]/2 \\
Real[A(0)] &= Real[F(0)] \\
Imag[B(0)] &= Imag[F(0)]
\end{aligned}$$

Note that it would be very unusual to actually want such a spectrum as B, and is much more likely that spectrum C, where C=DFT[V], would be required. To get this latter result the following transform can be performed on spectrum B

$$\begin{aligned}
C(n) &= j.B(n) \\
Real[C(n)] &= Imag[B(n)] \qquad 0 < n < N-1 \\
Imag[C(n)] &= -Real[B(n)]
\end{aligned}$$

or
$$\begin{aligned}
Real[C(n)] &= Imag[F(n)+F(N-n)]/2 \qquad 1 < n < N-1 \\
Imag[C(n)] &= Real[F(N-n)-F(n)]/2 \\
Real[C(0)] &= Imag[F(0)]
\end{aligned}$$

Thus two real input N-sized DFT's can be performed for practically the same computational effort as one complex N-sized DFT. This technique can be utilised to greatly improve the efficiency of an FFT.

It can be seen in figure 2.6 that a DFT performed via the FFT technique can easily be split into two half sized FFT's for all except the final set of butterflies. Thus a real input FFT could be performed by splitting the input data into two halves, performing two real input N/2 point FFT's within one complex input FFT by using the above technique, and then reconstituting and performing the final stage of the larger FFT. This process is shown in figure 2.7. This method of performing a real input FFT requires only a touch more

computational effort than that for a complex FFT of half its size. The two to one relationship becoming closer as the size of the FFT increases.

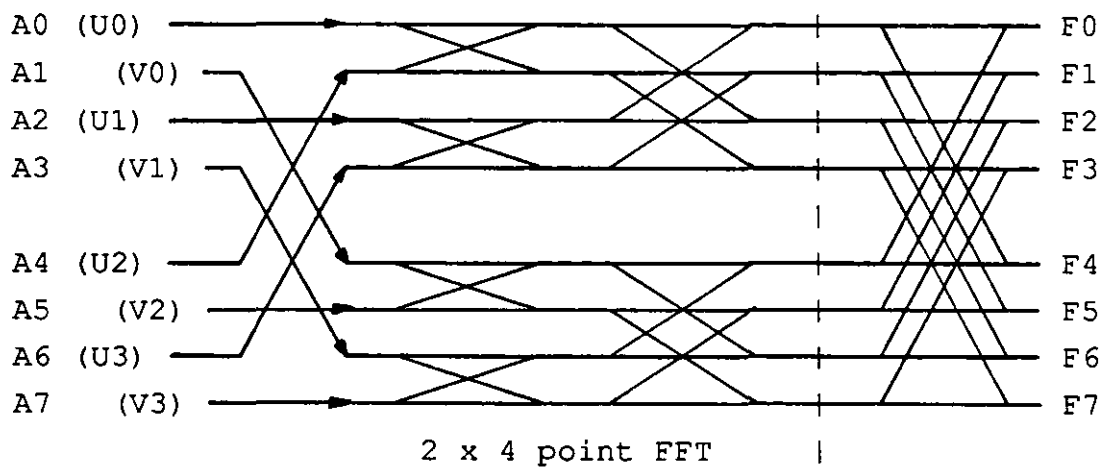


Figure 2.6 - complex input 8 point FFT

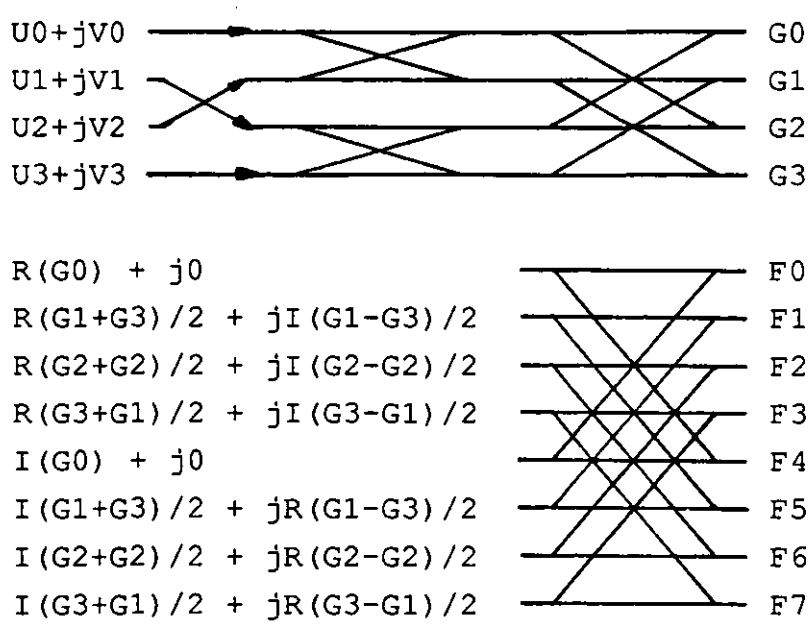


Figure 2.7 - real input 8 point FFT

There is also the possibility of halving the memory requirement when using the real input FFT. This can be done by expanding out only two complex components of the combined N/2 point spectra, at a time. A butterfly operation can then be performed on the two points and one of the double sided spectra components discarded. Thus only half of the complex components that would be required for an N point FFT are ever stored.

2.5.3 Computational saving of a real input FFT.

Again, the computational effort required to perform an FFT using this last method can be estimated. An N point transform now consists an N/2 point FFT, a spectrum splitting transform and then the last column of an N point FFT.

If radix 4 butterflies are being used, which will be assumed to be the case, the equation to define the number of operations necessary will depend upon whether the number of points (N) is a power of four. If it is a power of four then the last butterfly will have to be split into two columns of radix 2 butterflies to enable the N point FFT to be halved right up to the last radix 2 column. If it is not a power of four then the existing radix 4 FFT will already have a final column of radix 2 butterflies allowing the FFT to be readily split up to this last column.

The computational effort will only be estimated for a transform where N is a power of four, as it was also only this case that was considered in the r4 FFT case (chapter 2.4.3). Note that this includes the 1024 point case which is of most interest. The computation consists of -

An N/2 point complex FFT :
Two N/4 point radix 4 FFT's = $2[4,25(N/4).\mathrm{Log2}(N/4)-4,5(N/4)]$
plus an N/2 point column of radix 2 butterflies = $5N/2$

A spectrum separation :
An N/2 point transform = N/2.4A = 2N

Last column of an N point FFT :
An N point column of radix 2 butterflies = 5N

Total number of operations = 2,125N.Log2(N/4) + 8,375N

Thus a real input radix 4 FFT requires 25984 operations to perform a 1024 point transform, this is just 67% of the operations required for the complex input radix 4 FFT, and only 58% of the original complex input radix 2 FFT. This reduction in computational effort is very significant and it was this last algorithm that was coded up for the TMS32010 signal processing micro processor, as will be described in chapter 4.

CHAPTER 3

FFT windowing and power estimation

3.1 Why the need for Fourier transform windowing.

The Fourier transform operates on discrete data sequences which are usually taken from continuous processes. The act of taking these data blocks and subsequently assuming that everything outside the block is zero, affects the true frequency spectrum of that part of the data sequence. The extent of the spectrum modification as seen at the output of a discrete Fourier transform is variable and dependent upon the signal characteristics inside the data block. An analysis of the sampling process and of the Fourier transform output characteristics shows why this is so.

Taking a block of samples from a continuous time function can be represented more precisely by

$$x(t) = x(t) \qquad -Tw/2 < t < Tw/2,$$
$$x(t) = 0 \qquad \text{elsewhere.}$$

i.e. the signal is multiplied by a window defined by

$$W(t) = 1 \qquad -Tw/2 < t < Tw/2,$$
$$W(t) = 0 \qquad \text{elsewhere.}$$

Now, it is well known that multiplying signals in the time domain is equivalent to convolving them in the frequency domain, thus the above windowing has the effect of convolving the frequency spectrum of the sampled signal with that of the window. The above window is known as the rectangular window and its frequency response is derived, using the Fourier transform, as follows

$$Xw(f) = \int_{-\infty}^{\infty} W(t).e^{-j2\pi ft}.dt = \left[\frac{e^{-j2\pi ft}}{-j.2\pi f}\right]_{-Tw/2}^{Tw/2}$$

$$Xw(f) = [(Cos(2\pi fTw/2) - jSin(2\pi fTw/2)) - (Cos(2\pi fTw/2) + jSin(2\pi fTw/2)] / -j2\pi f$$

$$Xw(f) = \frac{Tw.Sin(\pi fTw)}{\pi fTw} = Tw.sinc(fTw)$$

This function is commonly known as the "sinc" function and is shown in figure 3.1.

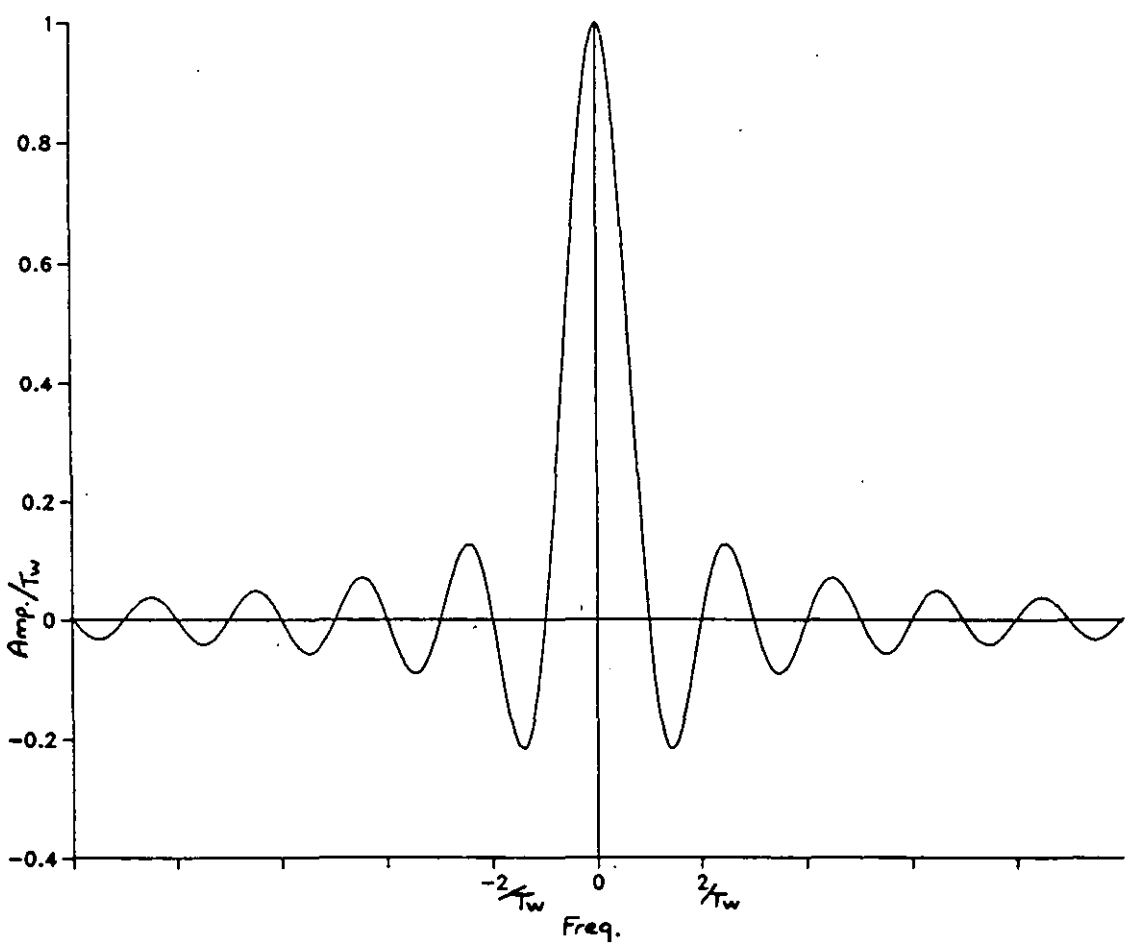


Figure 3.1 - Sinc function

The result of the convolution of this spectrum with that of the sampled signal, is presented for analysis to the DFT, an example of this is shown in figure 3.2.

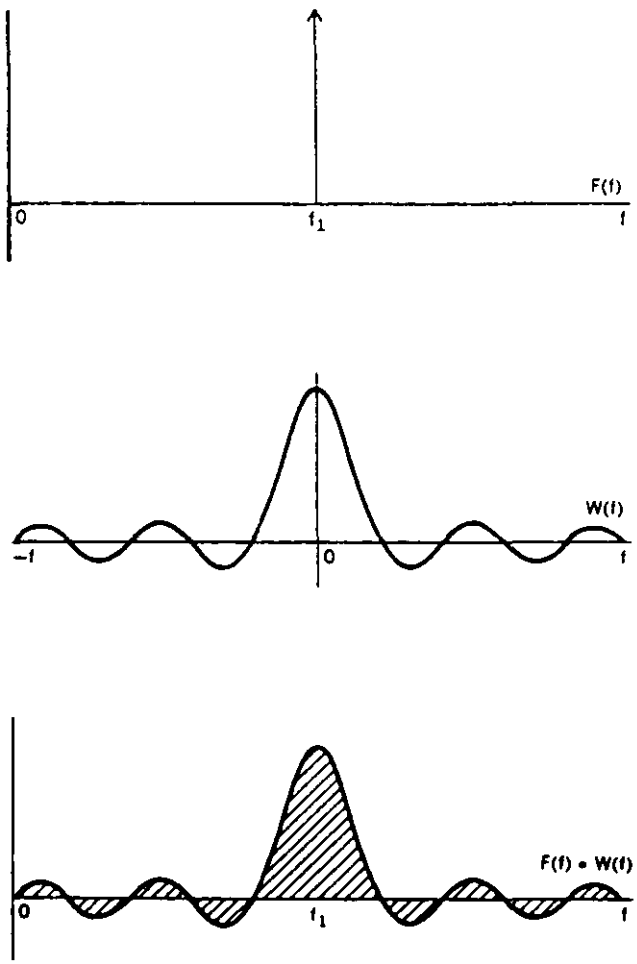


Figure 3.2 - Convolution of sampled signal with the sinc function

Comparing the characteristics of the DFT output and of the sinc function we find the following

The output from a DFT is a discrete double sided spectrum, where each discrete filter is spaced at Fs/N, where Fs is the sampling frequency and N is the number of samples.

The sinc function has zero points wherever f = 1/Tw, where Tw is the time duration of the rectangular window.

Now Fs/N = 1/(Ts.N) and Ts.N = Tw, hence the frequency spacing of the sinc function zero points and the filter spacing of the DFT output are the same.

The effect of this relationship can be observed if we look at the spectrum for a signal with an integer number of cycles within the sample block, and also for a signal with a fractional number of cycles within the block.

Case 1 : Integer number of cycles.

It can be seen from figure 3.3 that when a sampled signal contains an exact number of cycles, one of the filters will lie exactly at the peak of the sinc function central lobe and all the others will lie on the zero points of the function. As a result the actual spectrum seen at the output of the DFT is the true spectrum of the original signal.

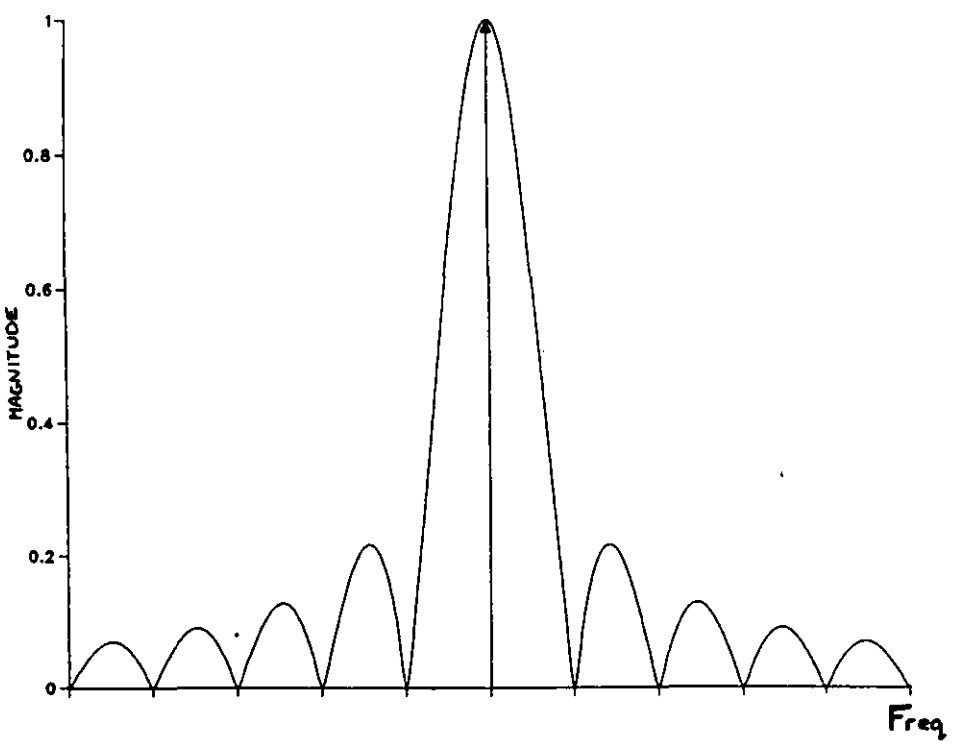


Figure 3.3 - Effect of windowing an integer number of cycles

Case 2 : Fractional number of cycles.

If a fractional number of cycles is sampled, the FFT output is somewhat different, the worst case being when half a cycle is involved. Considering this case and refering to figure 3.4, it can be seen that the central lobe is only represented by two attenuated filters either side of the peak. The amplitude of these filters is

$$\text{Filter spacing} = Fs/N = 1/Tw$$
$$\text{Therefore half spacing} = 1/(2.Tw).$$

$$\Rightarrow \frac{\text{Sin}(\pi\, Tw/(2.Tw))}{\pi\, Tw/(2.Tw)} = \frac{\text{Sin}(\pi/2)}{\pi/2}$$

$$= 0.637 = -3.9 \text{ dB}$$

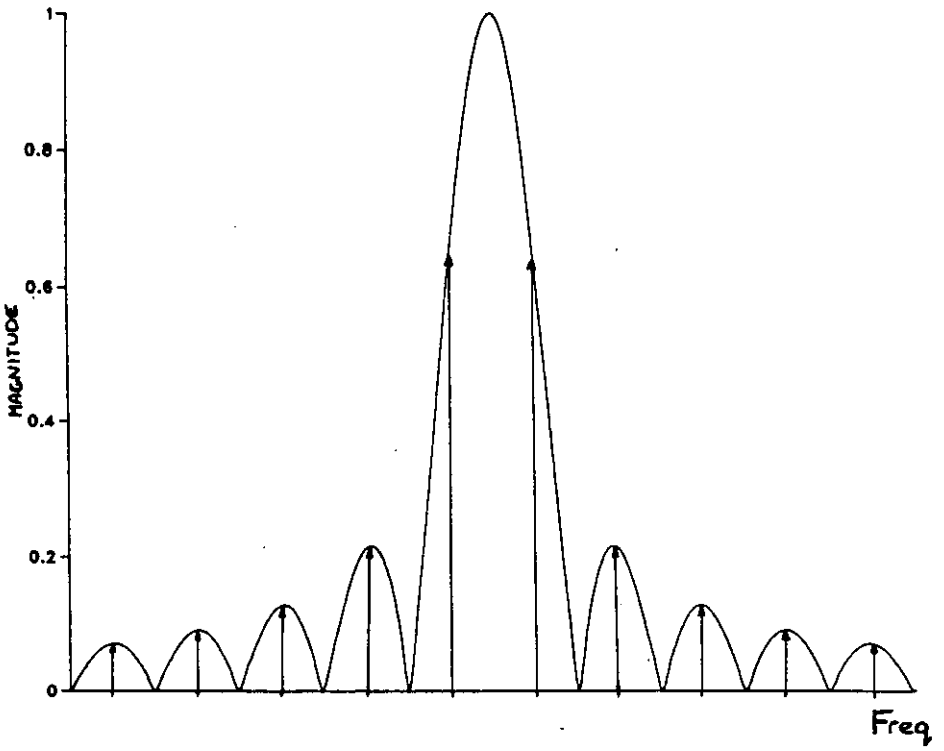


Figure 3.4 - Effect of windowing a fractional number of cycles

It can also be seen that all of the filters outside of the central lobe lie exactly on the peaks of the side-lobes, thus the DFT output contains many spectral components not present in the original signal, this effect is called leakage. The first side lobe is 13.5 dB down on the central lobe amplitude and only 9.5 dB down on the central lobe estimated peak.

As a result of the spectrum missing the peak of the central lobe in all conditions other than case 1, the signal amplitude appears to vary by up to 3.9 dB as it moves from dc to Fs/2, this is known as the picket fence effect [5] and appears as in figure 3.5

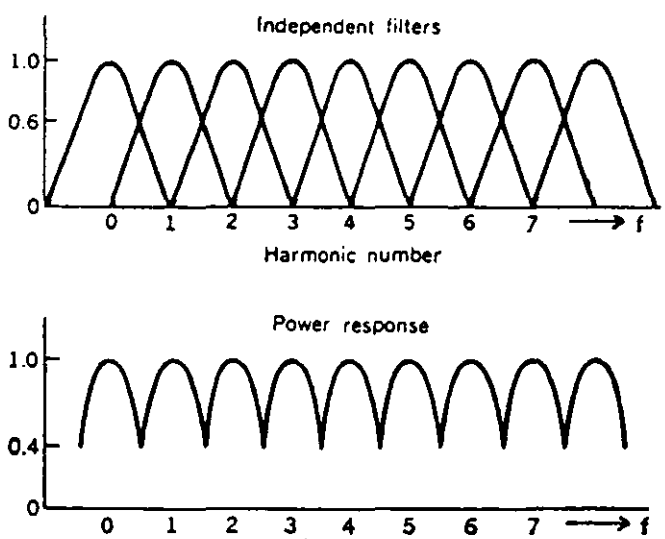


Figure 3.5 - Picket fence effect

Apart from the amplitude estimation errors caused by the picket fence effect, the sinc function leakage is significant enough that a small signal can be obscured by a neighbouring large signal. This is demonstrated in graphs 3.1.1 & 3.1.2 which show how a component five and a half filters away from another, and 20 dB smaller than it, is almost completely obscured by the larger ones side lobes components.

Thus there are two distinct problems associated with the rectangular window which must be alleviated if the DFT is to be of any real use -

1) the picket fence effect must be reduced,
and 2) the side-lobes (leakage) must be reduced.

3.2 Aero engine transducer signal characteristics.

Before tackling the above problems it is worth recapping on the type of signals which will be presented to the real time spectrum analysis system, as described in chapter 6. The characteristics of these signals and the information required from them are briefly

1) The signal to noise ratio of engine signals is typically no better than 40 dB although on some rare occasions it can be better. The inherent level of the spectrum floor due to windowing should be lower than this noise floor so as not to obscure any information within it. Hence a spectrum amplitude range of approximately 50-60 dB is required over as much of the spectrum frequency range as possible.

2) It is possible to have very many engine order related components within a signal, but more significantly there can also be non-integral components which inevitably pass through the other components during engine manoeuvres. Hence components should be distinguishable at frequencies as close together as possible.

3) The results of the spectrum analysis will be passed on to some expert system to assess engine health and performance. To enable the expert system to extract as much information as possible, the general component shapes should be consistent irrespective of their actual position within in the spectrum.

3.3 Superior windows.

The performance of the Fourier transform can be altered and improved if the shape of the convolution spectrum is modified by reshaping the time domain rectangular window. Bearing the above three points in mind, the desirable features that this modified window should provide at the output of the FFT are .

1) narrow central lobe,
2) minimal spectrum modification by side lobes above -60 dB of the largest component,

and 3) consistent shape of spectrum components.

Over the years many variations of weighting functions have been derived [22], all attempting to make the best compromise between central lobe width and side lobe levels. One of the most popular and simplest is the Hanning window, its time domain function being

$$W(t) = 0,5 + 0,5.\mathrm{Cos}(2\pi t/Tw) \qquad -Tw/2 < t < Tw/2$$

This function is shown in figure 3.6 and its frequency response in graph 3.2.1. The side lobes are now much smaller than those for the rectangular window although the central lobe has widened, this is because the side lobe energy has effectively been transferred into the central lobe. This widening is not totally detrimental as the picket fence ripple is now only 1.42 dB. In general the central lobe cannot be narrower than two filter widths (2/Tw), as for the rectangular window, and a 20 dB drop in the side lobe levels results in one filter width (1/Tw) increase of the central lobe. Most of the superior windows tend to have a central lobe width of 3-4 filters. Some other window functions include

$$\text{Hamming} \quad - W(t) = 0,54 + 0,46.\mathrm{Cos}(2\pi t/Tw)$$

$$\text{Gaussian - } W(t) = e^{-\left[\left(\frac{2t}{Tw}\right)^2 \frac{\beta^2}{2}\right]}$$

$$\text{Dolph Tchebyshev - } W(t) = FFT^{-1}\left[\frac{\text{Cos}\left[Tw.\text{Cos}^{-1}\left(\alpha \text{Cos}\left(\frac{2\pi t}{Tw}\right)\right)\right]}{\text{Cosh}\left(Tw.\text{Cosh}^{-1}(\alpha)\right)}\right]$$

$$\text{where } \alpha = \text{Cosh}\left[(1/Tw).\text{Ln}\left(10^{\beta} + \sqrt{10^{2\beta}-1}\right)\right]$$

$$\text{Kaiser Bessel - } W(t) = \frac{Io\left[\beta\sqrt{1-(2t/Tw)}\right]}{Io[\beta]} \qquad Io[x] = 1+\sum_{K=1}^{\infty}\left[\frac{(x/2)^K}{K!}\right]^2$$

In a paper by F.J.Harris [22] the above described windows, and others, are compared against each other for properties similar to those stated in chapter 3.2. He concludes in his paper that the Kaiser Bessel and the Dolph Tchebyshev window functions give the best results for the test signal that he applied, and for this reason it is these two window functions that have been chosen for further analysis. The time domain functions are shown in figure 3.6 and their frequency responses in graphs 3.2.2 & 3.2.3. This analysis is also performed on the Hanning window as this is in common use and serves as a standard for the comparisons.

One reason for the superiority of the Kaiser Bessel and Dolph Tchebyshev windows is because they contain a variable parameter (β) which can be tailored to suit the requirements, and their side lobes have a near flat response. These parameters effectively vary the amount of energy that is distributed between the side lobes and the central lobe.

3.3.1 Extent of window analysis.

As stated above, the three windows of interest are the Hanning, the Kaiser Bessel, and the Dolph Tchebyshev. The Hanning window has no variable parameters and thus is

straight forward to implement, however the other two windows both have a variable parameter which affects their side lobe levels. A considerable amount of pre-investigative work has been carried out on these window functions by the author [9] using various parameter values on a number of different data sequences. The findings shown below are a summary of this work and thus only include the results for the best Kaiser Bessel window and the best Dolph Tchebyshev window. The value of the parameter β used in the following cases was 6.

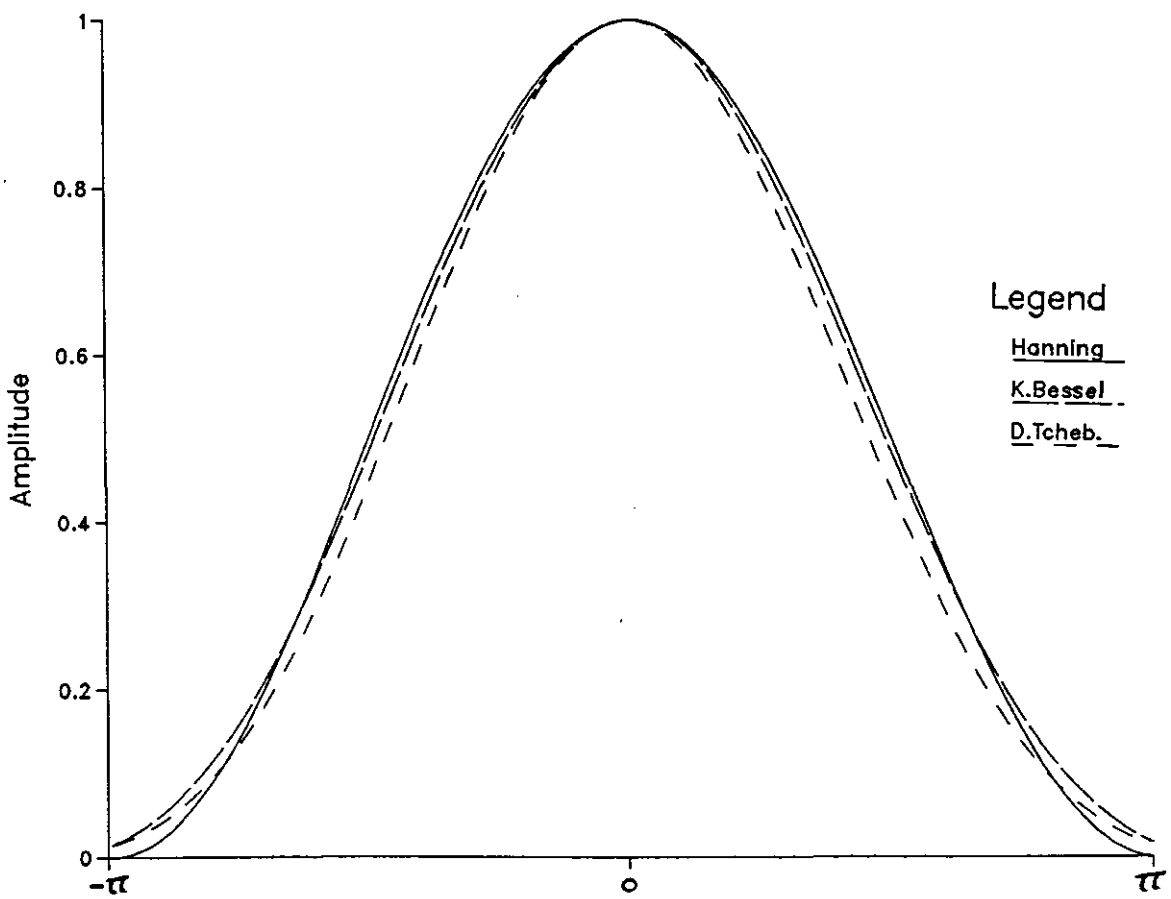


Figure 3.6 - Time domain resonse for Hanning, Kaiser Bessel and Dolph Tchebyshev

3.3.2 Simulated test data.

To compare the windows against each other and to test that they meet the requirements, each window has been applied to various simulated data sequences. However, for brevity, this summary only includes three of the data sequences that were used in the original work. The data sequences and the reasons for using them are as follows

The first two data sequences (64 points each) are designed to show how well the windows allow small frequency components to be identified when they are close to larger components, and to show how the position of the components, on or inbetween filters, affect the overall spectrum. These two simulated data sequences are as follows

1. Signal composed of two sinusoids -
   i) one of amplitude 1.0 at harmonic 10
   and ii) one of amplitude 0.01 at harmonic 16

2. Signal composed of two sinusoids -
   i) one of amplitude 1.0 at harmonic 10.5
   and ii) one of amplitude 0.01 at harmonic 16

The third data sequence (512 points) is designed to show the effect when there are several components, and what the general shape of the side lobes is over more realistically dimensioned DFT's.

3. Signal composed of seven sinusoids -
   i) five of amplitude 1.0 at harmonics 50.5, 55, 100.5, 150.5, and 200
   and ii) two of amplitude 0.01 at harmonics 106 and 155.5.

3.3.3 Real vibration data.

The three windows have also been applied to a real vibration signal taken from an aero-engine mounted accelerometer. The vibration data was obtained from a conditioned signal via a 12-bit analogue to digital convertor applied after an anti-aliassing filter. More information about this signal and its spectral content is given in chapter 9 where it is also applied to other spectrum estimation techniques. This signal obviously does not contain all the different conditions found in real signals, but it does give a good indication of the signal to noise ratio and of the noise colouration.

3.4 Assessment of window performance using simulated data.

The three windows are assessed in two tests using the three sets of data sequences

3.4.1 Test A - Data sequences 1 and 2:

Hanning window, graphs 3.3.1 & 3.3.2 :- The Hanning window is a significant improvement on the rectangular window although it still displays very inconsistent side lobe amplitudes as components move across the filters. Although the 0.01 amplitude component is clearly visible in the first spectra, it is almost completely obscurred in the second. In this latter case the side lobes are quite large around the central lobe although they do eventually decrease to very low levels.

Kaiser Bessel window, graphs 3.4.1 & 3.4.2 :- This window has produced two very similar spectra. In each case the central lobe is quite compact and does not spread out excessively at its base. The side lobes very quickly reduce to about 55 dB below the central lobe peak and then form a gently sloping base going beyond 60 dB. The 0.01 amplitude component is clearly visible in both spectra.

Dolph Tchebyshev window, graphs 3.5.1 & 3.5.2 :- This window produces fairly consistent spectra although there are some differences between their side lobe amplitudes. These side lobes however are very flat at 60 dB below the central lobe peak. The 0.01 amplitude component is clearly visible in both spectra.

Comparing the performance of the Fourier transforms for the three windows it can be seen that there is little difference between any effects the central lobes may have, but the side lobe responses are quite different for each. As far as being able to distinguish the small component from the larger one, the Hanning window fairs badly compared to the other two which perform very similarly.

3.4.2 Test B - Data sequences 3;

Hanning window, graph 3.6 :- This graph clearly shows the extent to which the side lobe amplitudes decay, and demonstrates how the smaller components are again nearly obscurred by the side lobes of larger components. The inconsistency between central lobe shapes is also quite apparent.

Kaiser Bessel window, graph 3.7 :- This window has produced consistently shaped central lobes with a amplitude floor at about 60 dB below the largest components, the smaller components are also very distinguishable from their larger neighbours.

Dolph Tchebyshev window, graph 3.8 :- This window has produced a very unusual effect in that the side lobe amplitudes appear to have combined to raise the amplitude of the floor to less than 50 dB below the largest components. This is a significant rise in the floor to the extent that the smaller components are nearly lost in it. The shape of the central lobes is however consistent and the amplitude

floor is extremely flat.

Comparing with the first test, the Hanning and Kaiser Bessel windows have performed as expected. The Dolph Tchebyshev window has however produced an unusual and detrimental effect on the performance of the Fourier transform.

3.5 Assessment of window performance using real vibration data.

As can be seen from graphs 3.9.1, 3.9.2, & 3.9.3 the three different windows have produced very few differences in their Fourier transform outputs and certainly no more information can be extracted from any one of the three spectra. This however is not surprising as the noise floor is only about 25 dB below the largest peaks. None of the windows has got any where near displaying their side lobes at this level.

3.6 Overall assessment of window functions.

The comparisons and tests shown above, together with those performed in previous work, indicate that the Kaiser Bessel (beta=6) window will produce the optimum spectra for use in further analysis and data reduction from aero engine transducer data. As the last test shows, in many cases the window characteristics are actually of no real concern because of the high noise content in the signal, and only in relatively clean signals will the benefits of the Kaiser Bessel window be reaped. There are however no disadvantages in using this window and has thus been employed in the real time dynamic data analysis system, as described in chapter 5.

3.7 Fourier transform power estimation.

The amplitude of sinusoidal components estimated by the Fourier transform can be up to 15% in error (28% for power)

due to the picket fence effect (see figure 3.5), note that this value does alter slightly with different window types. The actual error for any particular component varies between 0% and 15% depending upon where the component lies within the Fourier transform output filters. This error is quite significant and is certainly higher than most measurement and instrumentation systems could tolerate. In the measurement of dynamic engine parameters total system accuracies are generally specified to anything between ±0.5% and ±5%. To improve on this situation a method is required which can more accurately extract a component's amplitude irrespective of its frequency.

After conversion into the frequency domain via the Fourier transform, a sinusoidal component's power is distributed between its central and side lobes right across the spectrum. Thus the actual power of the sinusoid is proportional to the sum of all the filter powers across the spectrum. It would of course be ridiculous to estimate power this way because of other components which would inevitably be present in the spectrum. It should be noted however that the majority, and certainly a constant amount, of power is concentrated in the central lobe. Thus, over a particular range of filters around the central lobe, the power in this region will remain approximately proportional to the true power.

The power in each filter is proportional to the amplitude squared, thus the true amplitude can be estimated via the root sum square of a number of filters around a sinusoidal components central lobe. The table below demonstrates the improved performance this technique provides and also shows how the number of filter summations affects the accuracy.

Frequency of input signal	Root sum square of amplitudes between Fb	Fb +/- 1	Fb +/- 2
Fb	1.0000	1.2513	1.25272
Fb+0.1Fs/N	0.9938	1.2505	1.25272
Fb+0.2Fs/N	0.9762	1.2489	1.25272
Fb+0.3Fs/N	0.9473	1.2459	1.25272
Fb+0.4Fs/N	0.9076	1.2409	1.25272
Fb+0.5Fs/N	0.8488	1.2335	1.25272
Worst case error -	15%	1.4%	0% (to 5 d.p.)

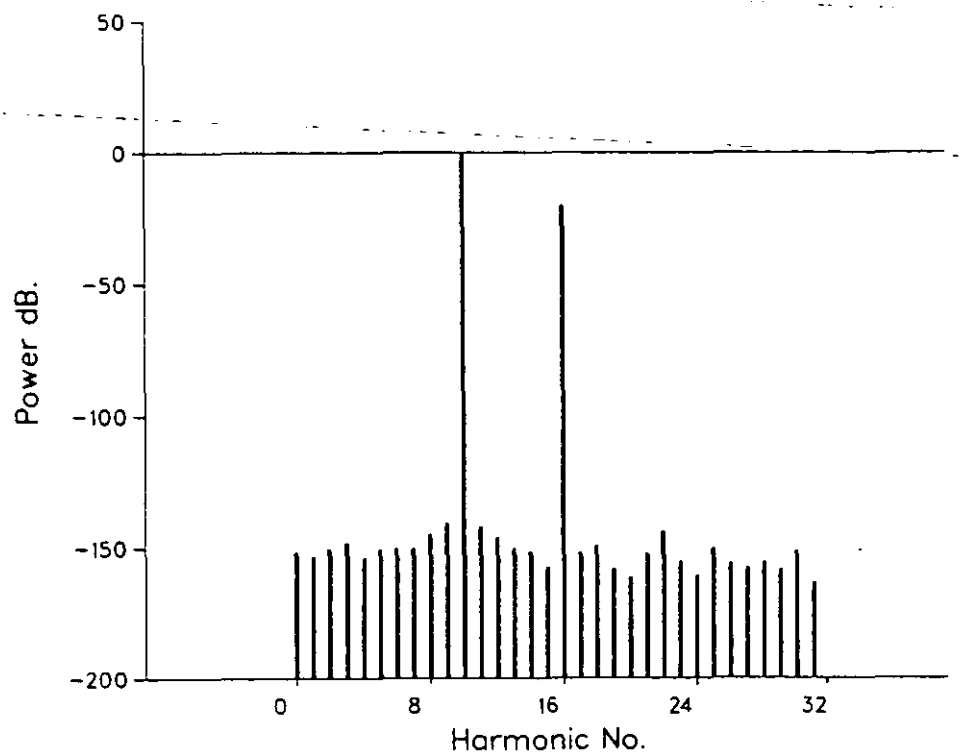
The information above comes from six Fourier transforms outputs (all using Kaiser Bessel windows), all of which have been applied to sinusoids of unity amplitude but of different frequencies. The frequencies vary from an arbitrary base (Fb) which lies exactly on an output filter, up to that base plus half the output resolution.

i.e. from Fb = n.Fs/N to Fb + 0,5Fs/N

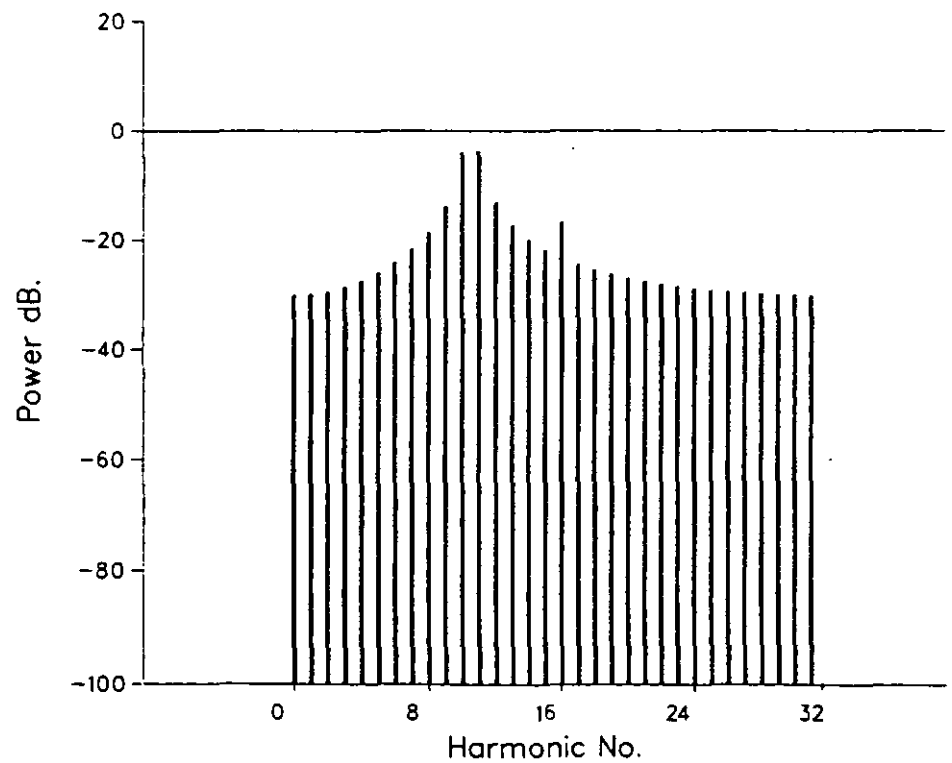
The three amplitude columns refer to the new amplitudes calculated by including 0, 1 and 2 filters from either side of the filter close to the central lobe peak (i.e. Fb).

It can be seen from the table that a very marked increase in accuracy is obtained by taking into consideration the amplitudes of the central lobe surrounding filters, the accuracy of the last column itself being extremely good. The amplitude values in columns 2 and 3 are of course higher than the true amplitude, but this can easily be rectified by normalisation. In real applications, even this is not necessary as all values are assumed to be of "banana" units and are effectively normalised during a calibration routine, assuming of course that the amplitude of the original calibration signal was estimated using this technique.

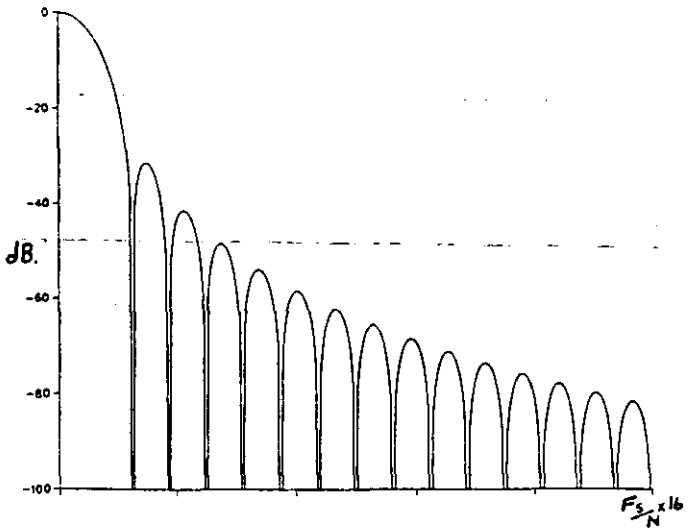
This technique can be applied to most Fourier transform applications as long as it is remembered that very close components will have an effect on each other. For cases such as vibration, as in graph 3.9, this technique is ideally suited and as will be seen in chapter 6, it has successfully been employed in a real time application.



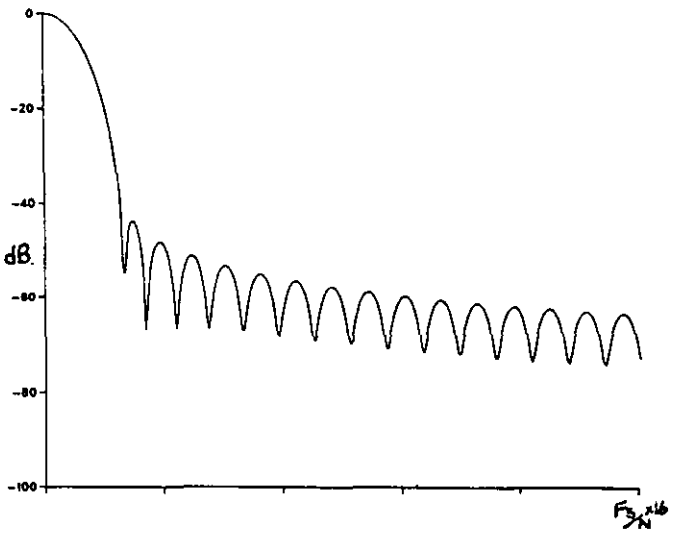
Graph 3.1.1 - 64 point DFT, rectangular window, sinusoids at harmonics 10 and 16.



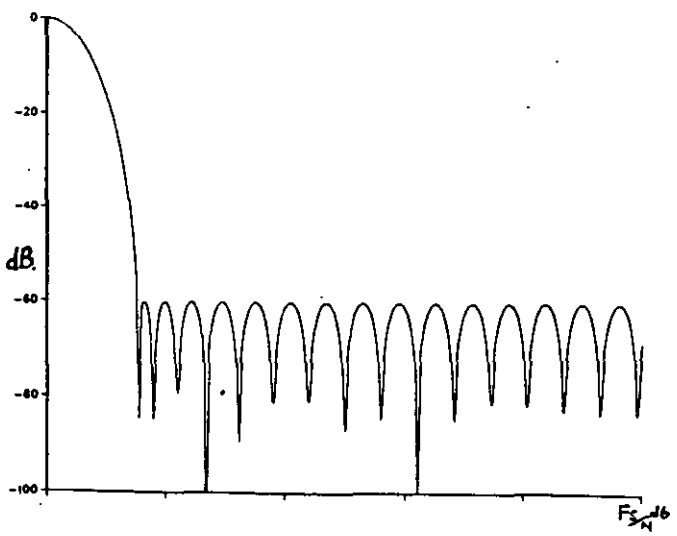
Graph 3.1.2 - 64 point DFT, rectangular window, sinusoids at harmonics 10.5 and 16.



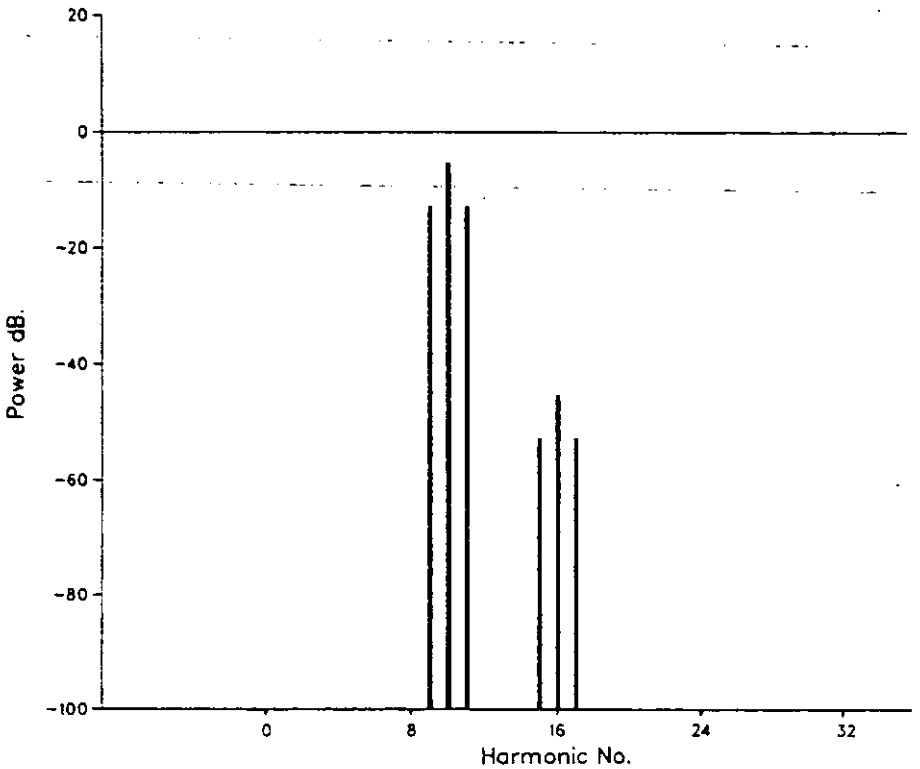
Graph 3.2.1 - Frequency response of Hanning window.



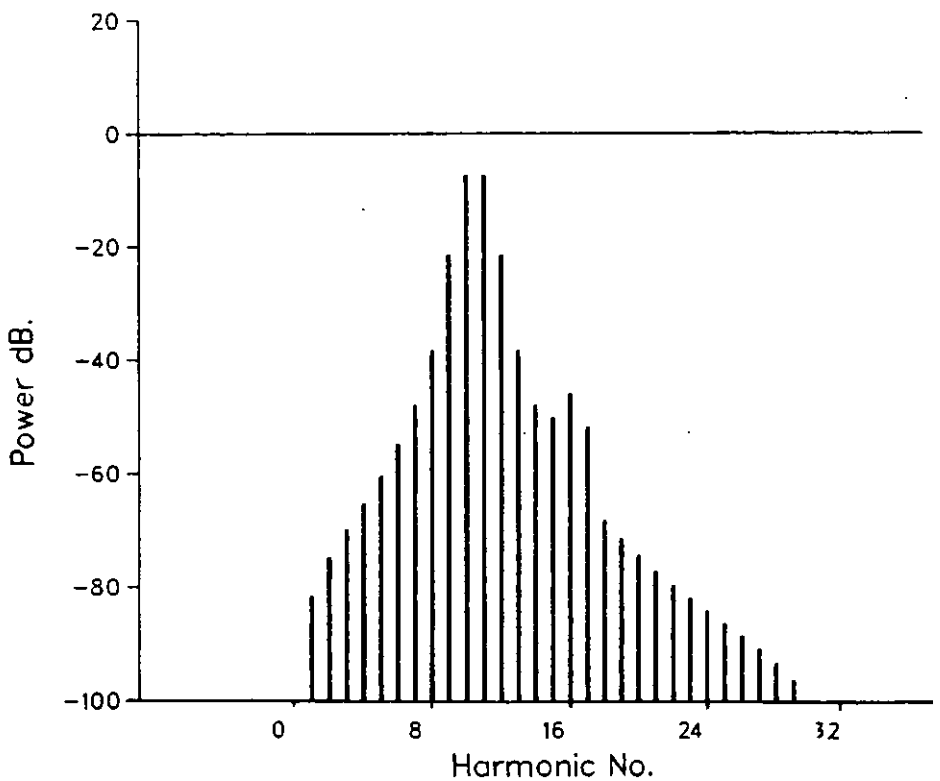
Graph 3.2.2 - Frequency response of Kaiser Bessel window.



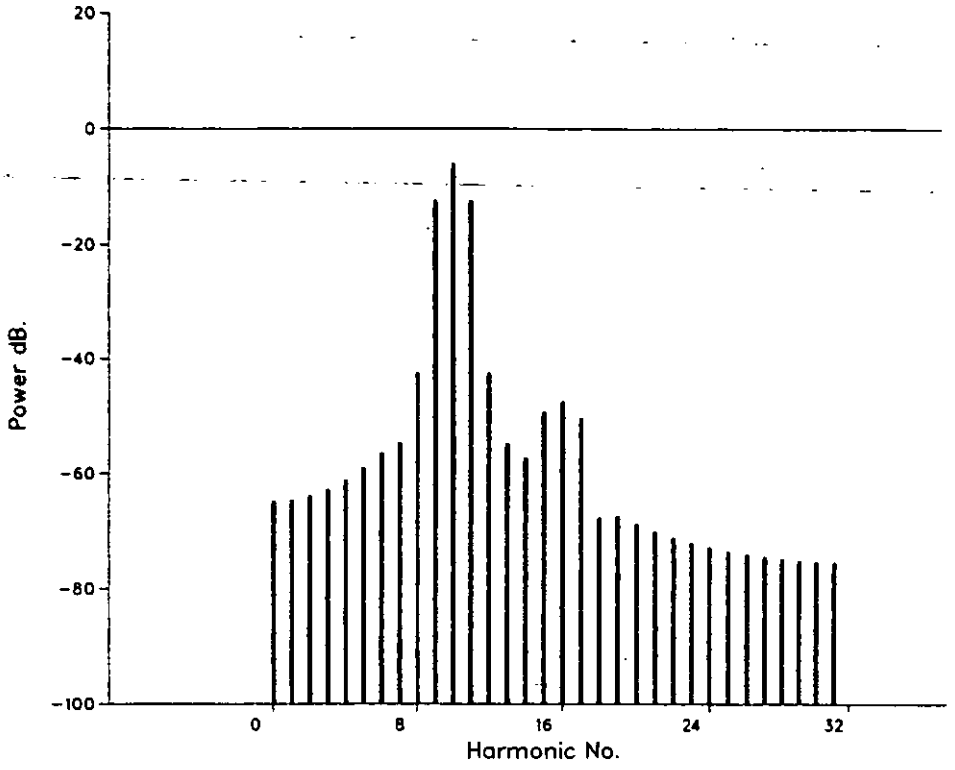
Graph 3.2.3 - Frequency response of Dolph Tchebyshev window.



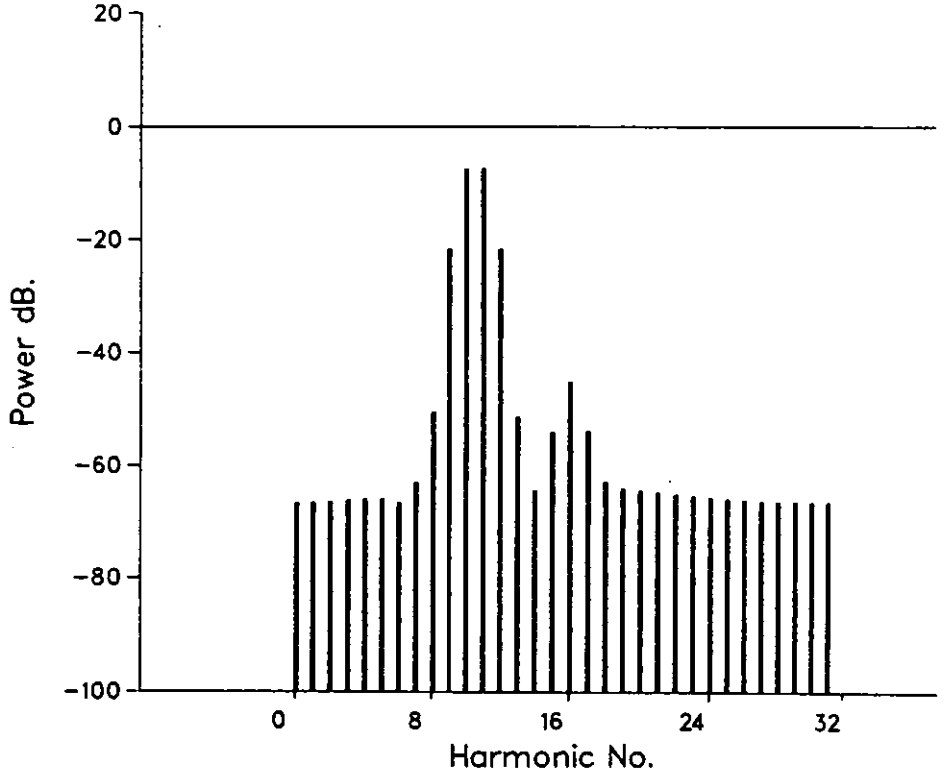
Graph 3.3.1 - 64 point DFT, Hanning window, sinusoids at harmonics 10 and 16.



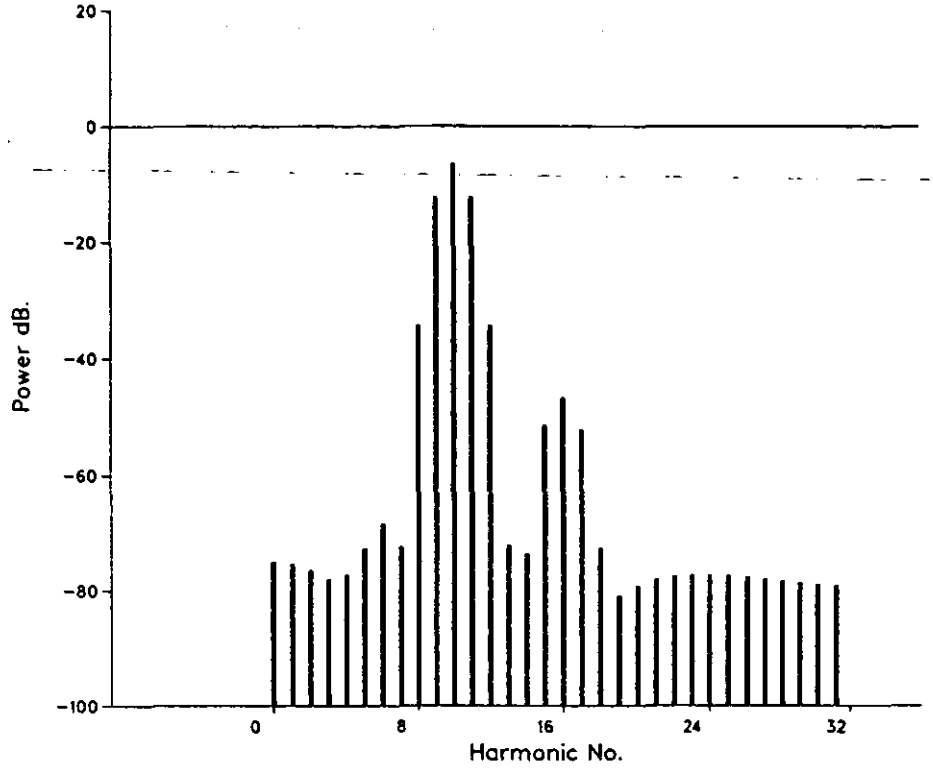
Graph 3.3.2 - 64 point DFT, Hanning window, sinusoids at harmonics 10.5 and 16.



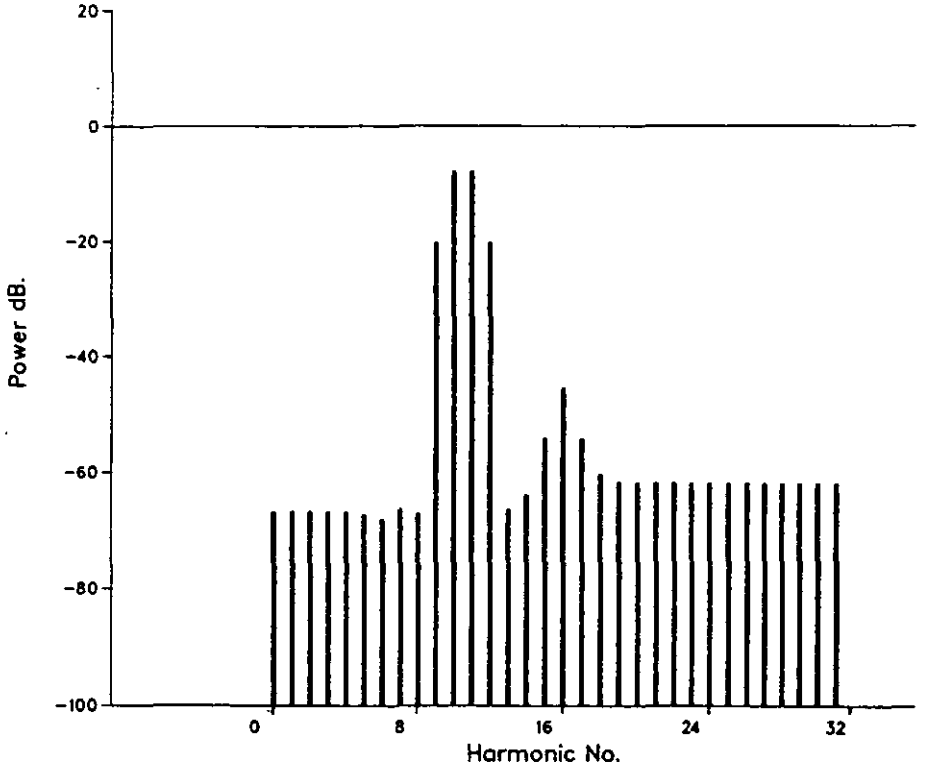
Graph 3.4.1 - 64 point DFT, Kaiser Bessel window, sinusoids at harmonics 10 and 16.



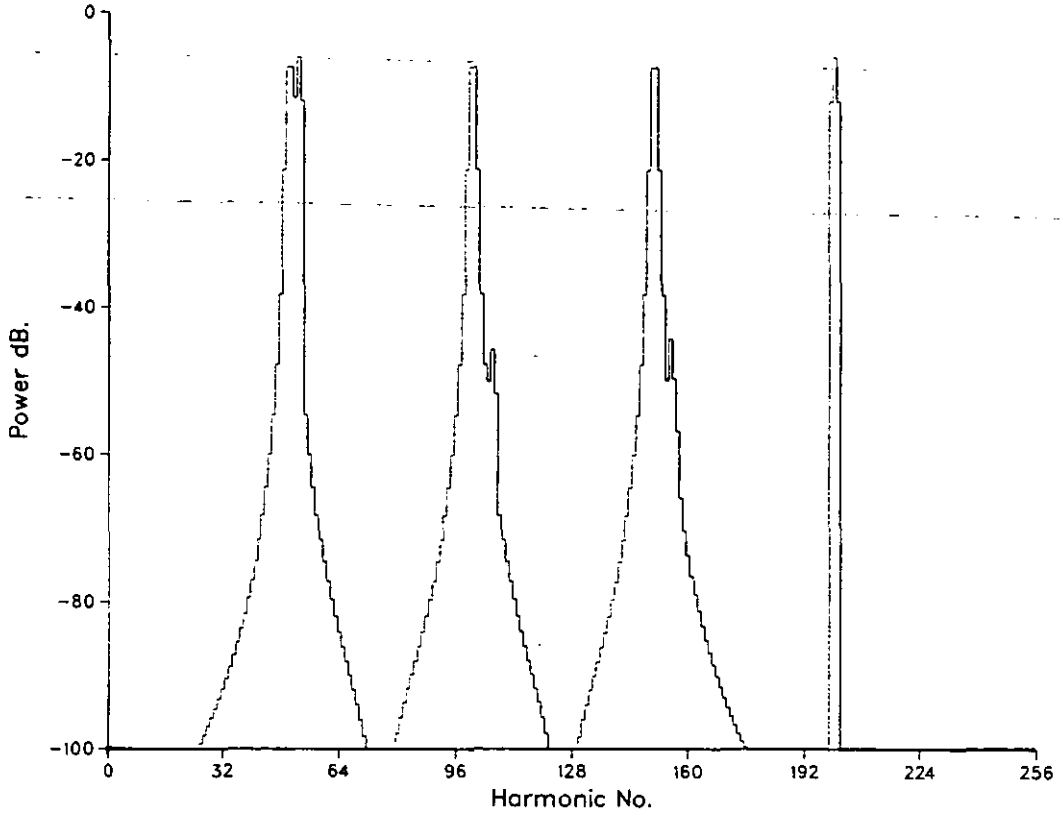
Graph 3.4.2 - 64 point DFT, Kaiser Bessel window, sinusoids at harmonics 10.5 and 16.



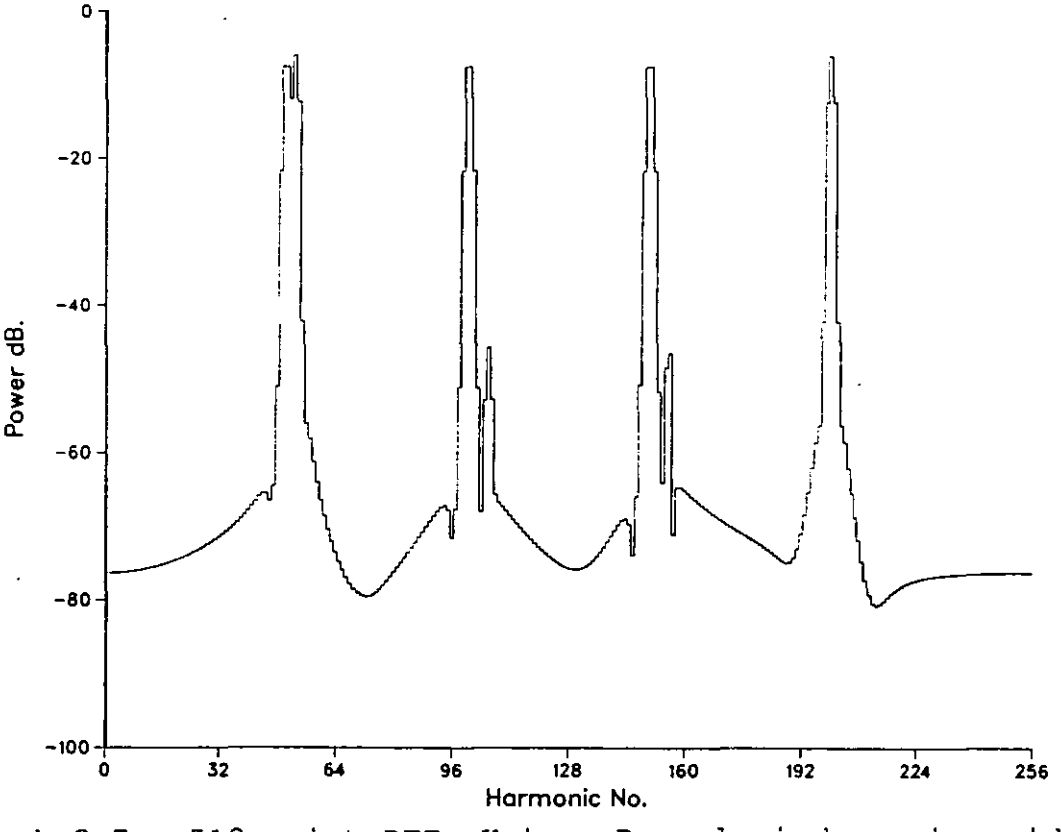
Graph 3.5.1 - 64 point DFT, Dolph Tchebyshev window, sinusoids at harmonics 10 and 16.



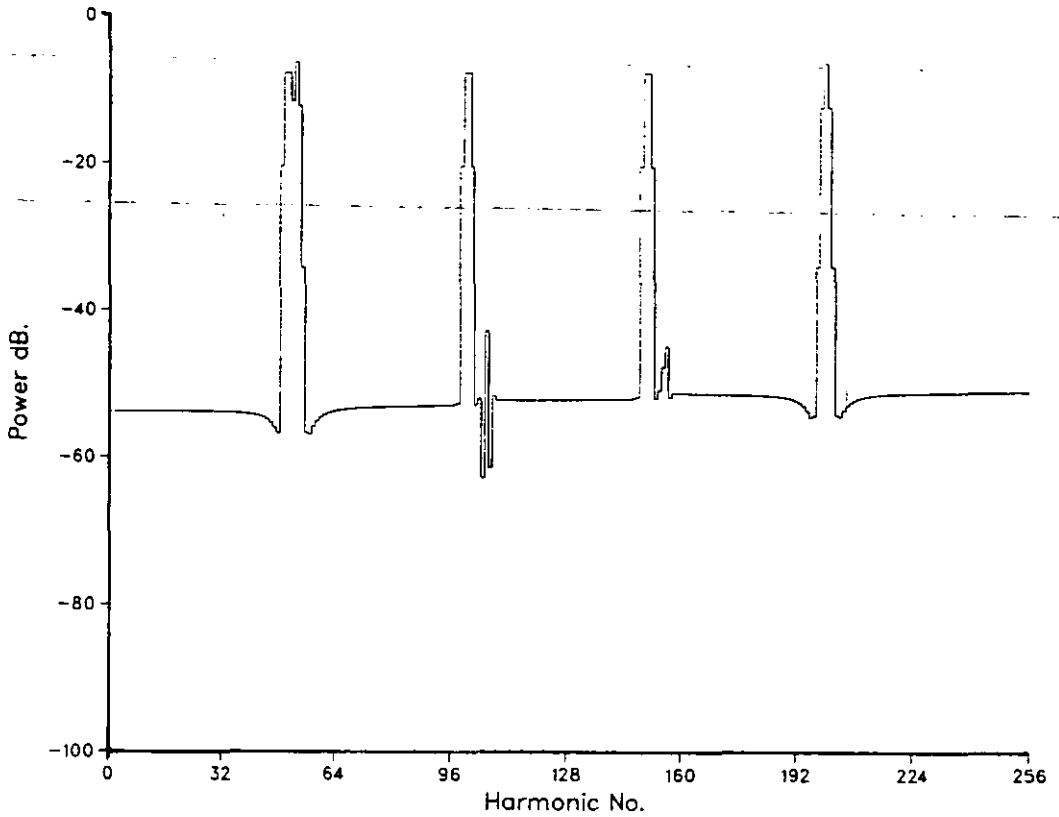
Graph 3.5.2 - 64 point DFT, Dolph Tchebyshev window, sinusoids at harmonics 10.5 and 16.



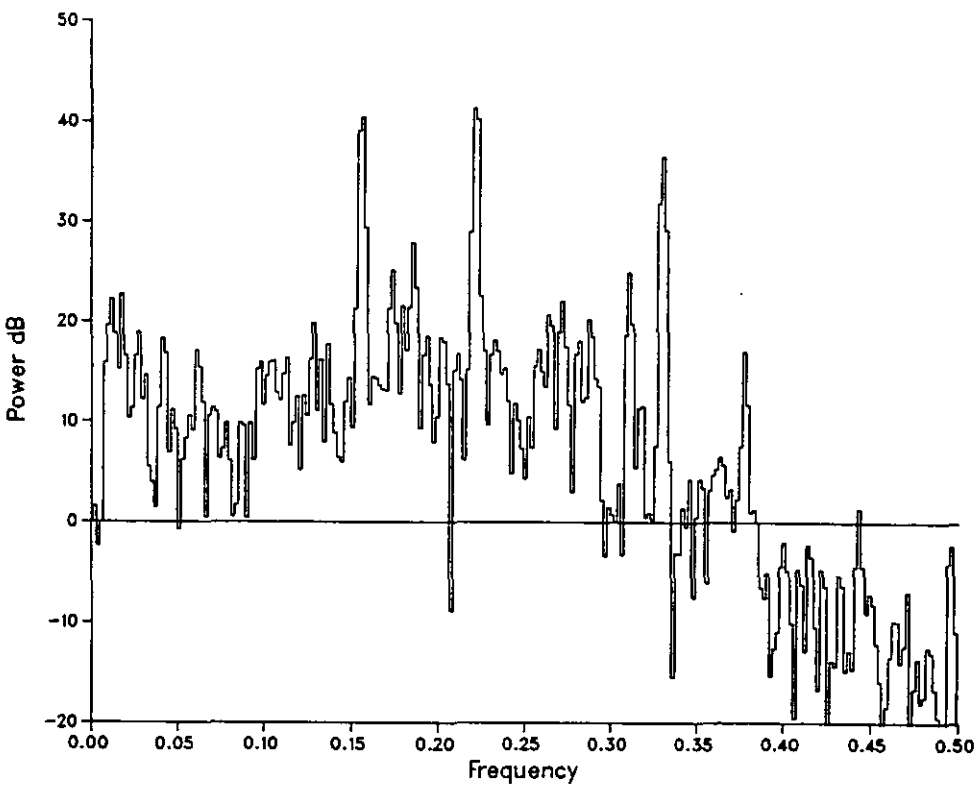
Graph 3.6 - 512 point DFT, Hanning window, sinusoids at harmonics 50.5, 55.5, 100.5, 106, 150.5, 155.5, 106.



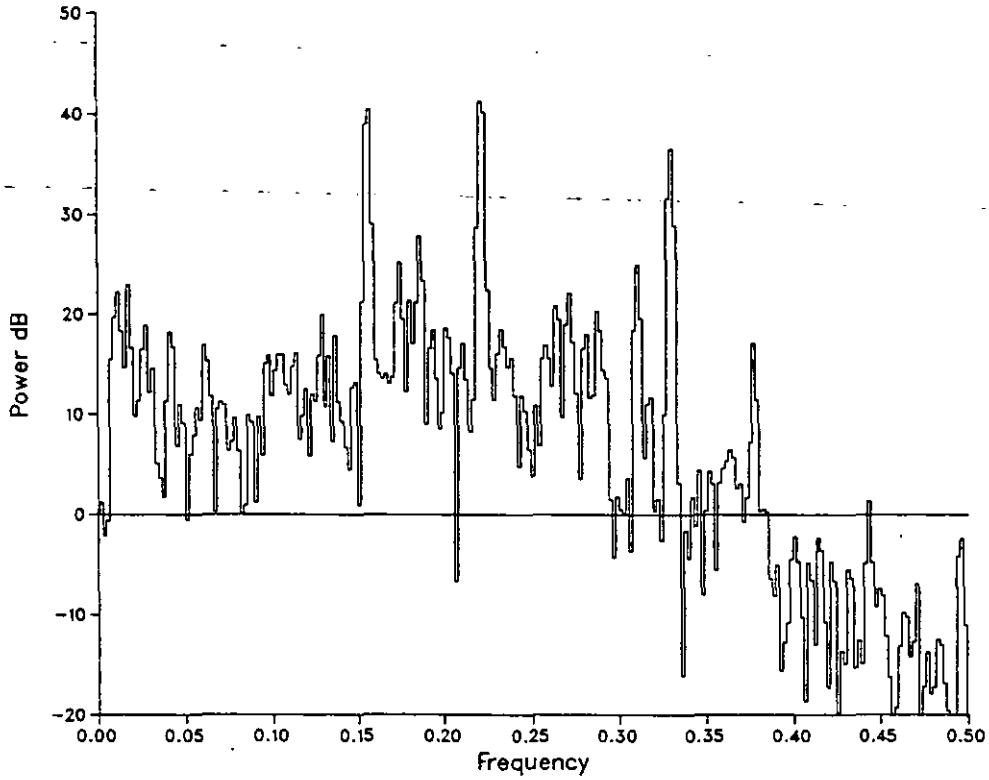
Graph 3.7 - 512 point DFT, Kaiser Bessel window, sinusoids at harmonics 50.5, 55.5, 100.5, 106, 150.5, 155.5, 106.



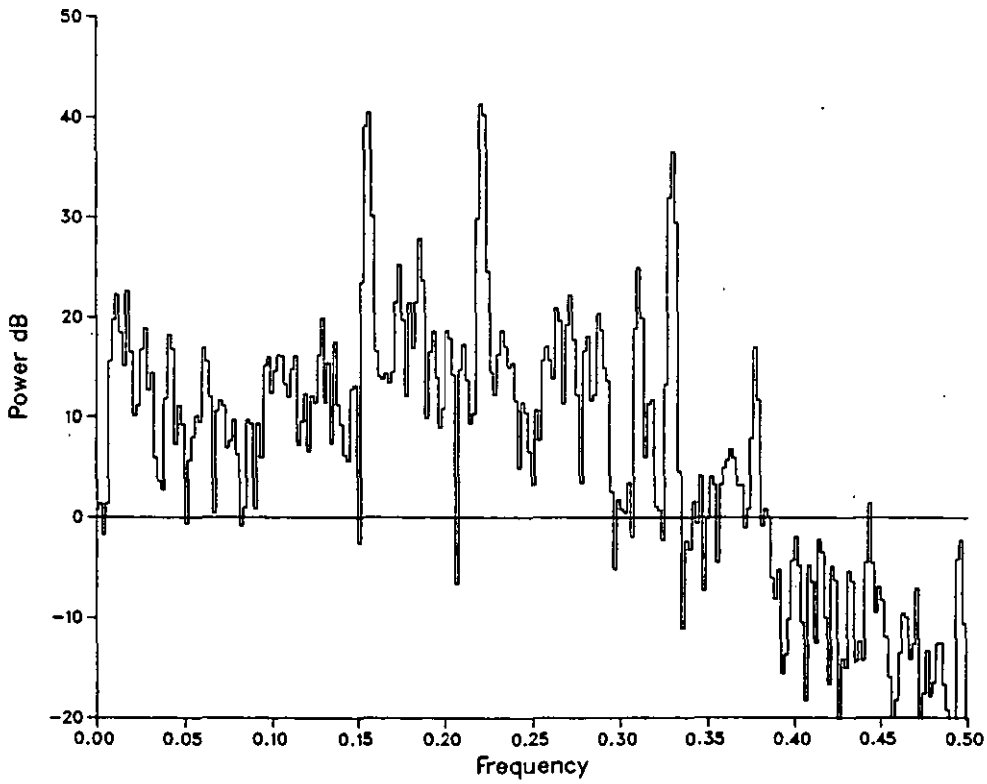
Graph 3.8 - 512 point DFT, Dolph Tchebyshev window, sinusoids at harmonics 50.5, 55.5, 100.5, 106, 150.5, 155.5, 106.



Graph 3.9.1 - 512 point DFT, Hanning window, aero engine vibration data.



Graph 3.9.2 - 512 point DFT, Kaiser Bessel window, aero engine vibration data.



Graph 3.9.3 - 512 point DFT, Dolph Tchebyshev window, aero engine vibration data.

CHAPTER 4

TMS32010 architecture and FFT routines

4.1 Why the TMS32010 micro-processor.

One of the fundamental requirements for a real time dynamic data analysis system is a fast hardware arithmetic unit to enable fast execution of digital signal processing algorithms, especially the fast Fourier transform, and also to perform data reduction algorithms at high speed on the resulting spectra. Basically there are two ways to achieve this, either by using bit slice hardware custom built for the application, or by using a fast micro processor.

The first option would enable a very fast arithmetic unit to be designed. However this type of hardware requires a substantial amount of circuit board area, and is not very flexible in terms of reconfiguration for different algorithms. It also requires a great deal of expensive development equipment which has to be tailored to each application.

The second option allows a much more flexible approach to be taken, as a circuit design for the micro processor and its peripherals can be almost completely done irrespective of the actual application. All subsequent design work can then be directed at writing application code using a standard and well defined set of instructions. The cost of development equipment for this option is also significantly less. However there is a trade off for this flexibility, and this is of course execution speed.

The second option was deemed to be the best choice as flexibility was of prime importance, especially as there were

many unknowns in the actual tasks that this system would eventually be applied to. However the over-riding incentive for choosing this option, was the recent arrival of very fast micro processors designed specifically with signal processing in mind.

The speed at which computationally intense programs can be executed was initially enhanced by the introduction of arithmetic co-processors such as the AM9511 floating point processor. This was then taken a step further by integrating the co-processor and the CPU within the same device. This was achieved by DEC when they literally grafted their arithmetic unit and their 11/70 CPU together to form a very powerful, although somewhat bulky, processor known as the J-11. It was achieved differently by NEC, who basically incorporated a few general purpose micro processor instructions and a small amount of memory capacity into an arithmetic unit. This was called the NEC 7720 and was perhaps the first true signal processing micro processor. However it was in 1982 that the first really practical and fast signal processing chip came onto the market, this being the Texas TMS32010 micro processor. This CPU encompasses a flexible and general purpose instruction set, a fast arithmetic unit, and a unique architecture designed with signal processing algorithms in mind, and this all in a 40 pin package.

The TMS32010 software development system originally existed in the form of an assembler and linker for the Texas 9900 work station and the IBM PC. To develop hardware Texas provided a very powerful in circuit emulator, the XDS/22. Having chosen the TMS32010 as the best signal processing micro, this development equipment was acquired. The emulator cost approximately £4,000 which although not particularly cheap, did enable fast and easy development of hardware and software. The cost of a TMS32010 chip in 1983 was approximately £120.

4.2 TMS32010 architecture.

The Texas TMS32010 micro processor is fabricated in NMOS technology and can operate with a 20 MHz crystal giving a bus speed of 5 MHz. The internal architecture is significantly different from most micro processors. The usual design involves a Von-Neuman approach where the instruction and data memory both sit on the same bus structure. The TMS32010 however employs a Harvard architecture in which the instruction and data memory both sit on separate and distinct buses, thus allowing them to be addressed simultaneously. The hardware capabilities are summarised below.

144 words of on chip ram
4 Kw of external program rom
1.5 Kw of internal masked program rom
16 bit data and instruction buses
200 ns instruction cycle
Signed 2's complement arithmetic
32 bit arithmetic accumulator
200 ns 16x16 bit multiply
0-15 bit barrel shifter (no time overhead)
Eight 16 bit inputs and outputs
Interrupt with context save
Single 5V supply.

It also features two auto increment/decrement indirection registers, a 4x12 bit stack, a single bit input line (BIO) and an on chip oscillator. Note that the internal and external program memories are hardware selectable and that only the external memory was ever used in the author's applications.

In general, arithmetic instructions access a word in the data ram and pass it through the barrel shifter, which can left-shift by between 0 and 15 bits (depending upon the instruction), to the ALU where it is either loaded into,

subtracted from, or added to the accumulator. After a result has been found it can then be stored back in the data ram, this is usually in two parts as the accumulator is 32 bits. The result can also be left shifted as it is stored, as an aid to scaling. A schematic of this architecture is shown in figure 4.1. It should be noticed that there is a link between the program and data buses (contrary to the strict harvard architecture) to allow program constants to be loaded into data ram.

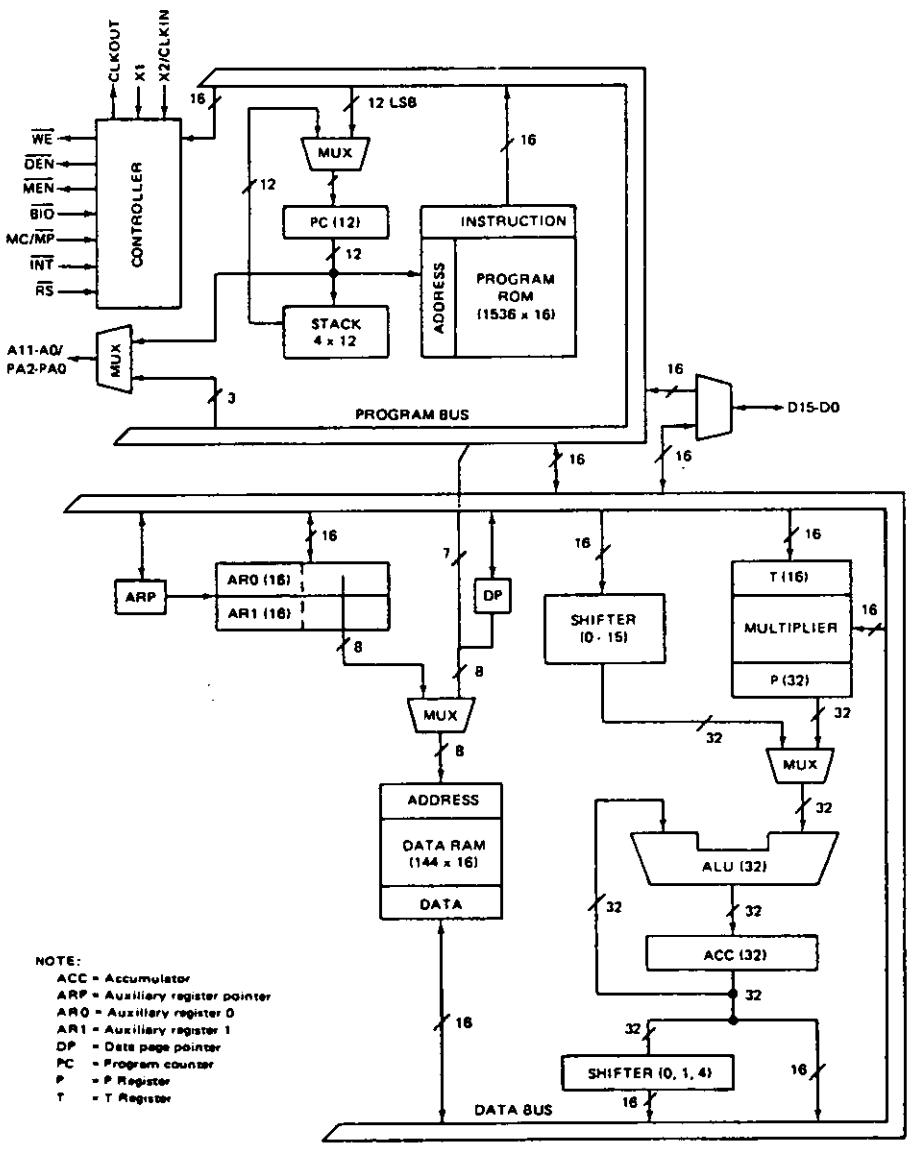


Figure 4.1 - Schematic of TMS32010 architecture

4.3 TMS32010 instruction set.

As stated before, the TMS32010 is not just a pure signal processor but has a number of general purpose instructions such as loop on condition operations, which allow considerable flexibility and program control. Almost all of the instructions are single cycle allowing execution rates of up to 5 million a second. Only the infrequently used control and I/O instructions are multicycle.

Of all signal processing operations, the multiply and accumulate must be the widest and most often used, as employed in correlation and convolution. The TMS32010 has been designed with this type of operation in mind and has subsequently been given a special instruction to allow fast execution of a contiguous number of these operations. This instruction nmemonic is LTD and is used with the multiply instruction (MPY) as follows

```
LTD  AR0- ,ARP=AR1
MPY  AR1-, ARP=AR0
LTD  AR0- ,ARP=AR1
MPY  AR1-, ARP=AR0
LTD  AR0- ,ARP=AR1
MPY  AR1-, ARP=AR0
                    etc.
```

The two indirection registers are called AR0 and AR1, and the indirection register pointer is called ARP. It is assumed that AR0 points to an input data table and that AR1 points to a constant table, and that both are in data ram. In the above sequence, each pair of instructions performs a 16x16 bit multiply, addition of the result to the accumulator, auto-decrement of both the indirection registers and a single positive address move of the data.

By using this instruction pair it is possible to perform a 64 point correlation or convolution in just 25.6 micro-seconds,

and also be immediately ready to start the process again. For more information on the architecture and instruction set of the TMS32010 micro processor refer to the Texas Instruments "TMS32010 User's Guide" [46].

4.4 FFT optimally coded for the TMS32010.

When programming an FFT into an integer machine with limited memory, such as the TMS32010, considerations such as scaling, data storage, constants storage, bit reversal, butterfly efficiency, and output formats must be tackled. The following two sections provide a basic description of how these were overcome in hardware and in software, together with some relevant examples. The two sections are split into radix 2 and radix 4 FFT development. The radix 2 version was the first to be implemented and accounts for most of the hardware and software design, development and debugging time. However it is the real input radix 4 implementation which has been finely tuned and installed in IDDAS.

4.5 Radix 2 FFT.

4.5.1 Data scaling.

The scaling of data as it passes through an FFT must be considered carefully because of the integer operation of the TMS32010. This machine stores its data to 16 bit resolution, and effort must be made to keep the numeric values as high as possible without causing an overflow.

Due to the Fourier transform being a data independent process, the highest possible gain that can occur in a butterfly can be predicted exactly, three cases exist

1) The first column of butterflies involves no complex multiplies and hence no phase shifting. If the inputs to

the butterfly are bounded by ±1 on each axis then the output will be bounded by ±2.

2) The second column of butterflies can only induce a phase shift of 90 degrees, thus if the inputs are bounded by ±1 then the output will again be bounded by ±2.

3) All successive butterflies can have a multitude of phase shifts and in particular 45 degrees, if the inputs are again bounded by ±1, then in this case the output is be bounded by ±2.414. This is shown in figure 4.2.

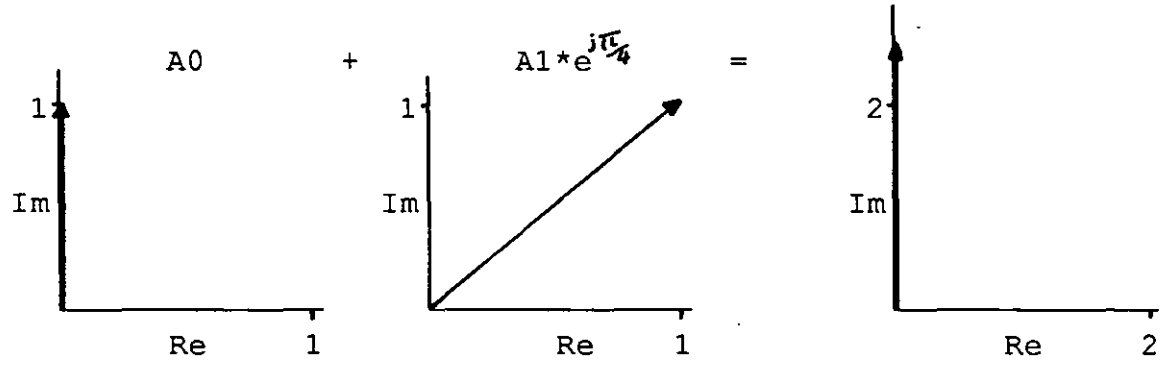


Figure 4.2 - Maximum gain from a radix 2 butterfly

To ensure that no overflow occurs within a butterfly, the inputs to it must not exceed the value of ±2**13. Note that the maximum that can be represented is ±2**15 (2's complement arithmetic). Although the maximum gain possible in some of the butterflies is 2.414, there is an overall gain limit of 2 per stage. If input data comes from a 12 bit bipolar ADC (as described in chapter 5) then it has a maximum value of ±2**11, and only allows a total gain of 16 before overflow occurs. Transforms of 1024 points have a potential gain of 1024 which is obviously far too high, to overcome this problem an attenuation of two is needed at the end of each butterfly. However to make the best use of the slight amount of gain which can be endured, the first set of "special butterflies" (which have no multiplies) can be left with no

attenuation.

4.5.2 Storage of constants.

Two sets of constants are required for an FFT, these are for 1) the window coefficients and 2) the exponentials, note that both need to be stored in program memory.

1) As explained in chapter 3, each input must be multiplied by a window coefficient before an FFT starts. Windows in general are symmetrical about the central point and hence N/2+1 coefficients are required for an N point FFT, i.e. 513 coefficients for a 1024 point FFT. Apart from the central coefficient, which is usually equal to unity, all the coefficients are less than one, thus to enable them to be stored as an integer they must all be increased in value. To maximise precision they are all multiplied by 2**15.

2) An FFT requires exponential coefficients for every one of its butterflies, many of the butterflies use the same coefficients, however it can be seen that the last column of an FFT uses a different exponential for each butterfly. Thus for an N point FFT where there are N/2 butterflies in each column, N/2 exponentials are required, i.e. N coefficients. Exponential coefficients also remain at or below unity and again must be multiplied by 2**15.

As can be seen, for a 1024 point FFT, 1.5 Kw of the TMS32010 4 Kw program memory has already been taken up by constants.

4.5.3 Input data storage.

The TMS32010 operates most efficiently when all data values are stored in internal ram. However it is only possible to

achieve this with an FFT of 64 or less input points as the internal memory is only 144 words in length and each point is complex and thus requires two locations. Extra ram can be implemented in two manners

1) Part of the 4 Kw program memory can be portioned off for data ram. This would make use of the TBLR and TBLW instructions which allow transfers between the program and data buses. These instructions do however require three clock cycles to execute.

2) Up to 64 Kw of external ram can be mapped into the I/O. This requires two ports, a unidirectional port for the memory address and a bidirectional port for the data. To keep I/O accesses to a minimum (note that each I/O transfer requires two clock cycles) a hardware auto-increment register can be implemented on the address I/O port. Hence memory accesses on large arrays require a once only initial write to this register.

The initial development board designed and built by the author, used the first method of external ram implementation. A 1024 point, complex input, radix 2 FFT was successfully executed on this hardware, it was however rather slow, taking approximately 90 ms to complete. Further development boards and IDDAS itself use the second method of implementation as this results in faster operation and allows full use of program memory and of larger data memories.

4.5.4 Bit reversal.

As demonstrated earlier, to achieve the symmetrically arranged butterflies and to keep the data values in the same position throughout an FFT, the input data must be initially rearranged into bit reversed address locations. Note that for this input and butterfly arrangement, the FFT is refered to as a decimation in time algorithm. It is possible,

although not shown here, to breakdown the DFT equation and arrange the butterflies in such a manner that it is the ouputs which must be rearranged via address bit reversal, this method is refered to as decimation in frequency.

Decimation in time is by far the most common method used, and in real time applications has an advantage over the other method. This is because during data initialisation prior to performing an FFT, windowing and address reversal can be performed on each input sample as and when it is presented to the TMS32010 micro processor, rather than inputting a complete array of raw data, then windowing and bit reversing it. There will inevitably be some delay between input samples, especially at low sample rates, hence by windowing and bit reversing as above, their execution times are effectively lost in the acquisition period.

Address bit reversal can be performed quite simply in the TMS32010 software due to the processors barrel shifter. However for a 1024 point FFT, where 10 bits must be reversed, this operation takes 40 cycles to execute, and together with the windowing which takes 10 cycles, the whole operation lasts 10 μs. At the highest required bandwith of 50 KHz the sample rate will be 131 KHz (as will be explained in chapter 5), giving only 7.6 μs between input samples, making software address bit reversal too slow. This problem is easily and elegantly solved by utilising a hardware bit reverser. This is achieved by connecting one of the TMS32010 bidirectional ports to the input and output registers of a ten bit D-type latch in which the TMS32010 data lines for the inputs and outputs are connected in opposite orientations. The bit reversal operation is now reduced to an "OUT" and an "IN" instruction, and takes only 0.8 μs. to execute.

4.5.5 Arrangement of butterflies and data.

The FFT derived earlier is known as an "in place" algorithm.

It is called this because the final spectrum data occupies the same memory locations as the initial address bit reversed input data, i.e. apart from manipulation within the butterflies, data effectively stay in their original locations throughout an FFT. This means that the columns of an FFT must be evaluated in order from left to right, although the butterflies in any particular column can be evaluated in any order.

In a 1024 point FFT where the data is stored in external memory, data pairs are brought into internal memory, operated on by an exponential constant and then outputted back to the external memory. Some exponential constants are applied to many butterfly data pairs (particularly in the early columns), thus to keep data transfer to a minimum, the data inputs can be arranged such that each exponential constant is only loaded once per column.

It will be shown in section 4.6 how the efficiency of the data transfers is improved for a radix 4 FFT, and in particular for IDDAS which uses I/O mapped external ram.

4.5.6 Radix 2 butterfly

The centre piece of an FFT is its butterfly routine, for example in a 1024 point FFT this routine is executed 5120 times. Hence the execution speed of the whole transform is almost totally dependent upon the efficiency of the butterfly. The efficiency of this routine is largely determined by the architecture of the processor on which it is performed. The butterfly requires four multiplies, six additions/subtractions and a certain amount of normalisation by shifting. The TMS32010 excells at both multiplication and shifting, and is certainly not slow at addition and subtraction.

The butterfly algorithm can be generalised as follows

$$F'(x) = F(x) + W[n].F(x+N/2)$$
$$F'(x+N/2) = F(x) - W[n].F(x+N/2)$$

where F and F' are complex data arrays and W is a complex constant array. The TMS32010 assembler routine which was written to perform this is shown below, note the following operand names and definitions -

F(x) = XRP + jXIP F(x+N/2) = XRQ + jXIQ
COS = 2**15.Cos(2πx/N) SIN = 2**15.Sin(2πx/N)

```
; Real component of complex multiply
ZAC                     ; zero accumulator
LT      XRQ             ;
MPY     COS             ; = XRQ.COS
LTA     XIQ             ;
MPY     SIN             ; = XIQ.SIN
APAC                    ; = (XRQ.COS+XIQ.SIN)/2**15
SACH    TEMP1,1         ; stored in temp1
;
; Imaginary component of complex multiply
ZAC                     ; zero accumulator
MPY     COS             ; XIQ.COS
LTA     XRQ             ;
MPY     SIN             ; XRQ.SIN
SPAC                    ; (XIQ.COS+XRQ.SIN)/2**15
SACH    TEMP2,1         ; stored in temp2
;
; Real components of F'
LAC     XRP,15          ;
ADD     TEMP1,15        ;
SACH    XRP             ; XRP+(XRQ.COS+XIQ.SIN)/2**15
SUBH    TEMP1           ;
SACH    XRQ             ; XRP-(XRQ.COS+XIQ.SIN)/2**15
;
; Imaginary components of F'
LAC     XIP,15          ;
ADD     TEMP2,15        ;
SACH    XIP             ; XIP+(XIQ.COS+XRQ.SIN)/2**15
SUBH    TEMP2           ;
SACH    XIQ             ; XIP-(XIQ.COS+XRQ.SIN)/2**15
```

Each of the above instructions only requires a single clock cycle to execute, thus, including a subroutine call and a return (which take a total of 3 cycles) the above routine takes a mere 5.2 micro-seconds to complete. An estimation for the total butterfly execution-time for a 1024 point FFT comes to 26.6 milliseconds. The actual FFT execution time is of course greater than this as no consideration has been given to the loading of constants and the loading and unloading of data to the butterfly.

4.5.7 Output format.

After an FFT has been executed, a double sided complex spectrum is left in memory, from this magnitude and phase information can be extracted. In almost all cases (99.99%) of aero engine analysis only magnitude is required, and is even more so the case for real time applications, subsequently phase is not dealt with at all. Magnitude can be represented either in linear or logarithmic formats, and it is anticipated that both will be required for analysis/ display purposes. The first case requires squaring, adding and square rooting, and the second requires squaring, adding and logging. These functions are tackled as follows

1) Linear output: Squaring two 16 bit values and adding them together could not be simpler for the TMS32010, however performing a 32 bit square root is another story. Square rooting on any integer machine is difficult, but is usually tackled by employing Newtons square root algorithm. This algorithm was successfully implemented in TMS32010 software but was found to be excessively time comsuming. The reason for this is that this algorithm uses a divide as part of its iterative process. This is only indirectly supported by the TMS32010 instruction code (nmemonic "SUBC") and subsequently takes 32 instruction cycles (6.4 μs) to perform a 32/16 bit divide.

The Newton square root algorithm was replaced by one of the author's design. This alternative algorithm uses the multiply, which is 32 times faster than a divide, as the major arithmetic operation within the iterative part of the routine. The algorithm starts by making a square root guess equal to half the largest possible answer, i.e. $4000. This guess is then squared and compared with the input value, if it is larger, $2000 is subtracted from the guess and if smaller $2000 is added. The new guess is squared and the process repeated with either $1000 added or subtracted, and then again with $0800. Eventually the exact square root will be guessed or, as more often is the case, the iterative process will come to a completion after adding or subtracting $0001. This routine only uses multiplies, shifts, additions and subtraction, all of which the TMS32010 executes at high speed. An assembled printout of this routine is shown in appendix A.

2) Logarithmic output : In this case a logarithmic value must be evaluated from the 32 bit result of the squaring and addition operation. No existing integer logarithm routines could be found, so again a routine was designed by the author.

The actual base to which the logarithm is taken is of no real consequence because as stated before, the data is in "banana" units prior to calibration. It was thus decided to use base two, as binary representations lend themselves to convertion in this base. The largest magnitude squared value that can result from an FFT is 2**31 ($2x[2^{**15}]^2$), and hence the largest integer part of a base two logarithm is 31. The integer part can thus be represented by 5 bits and is evaluated by determining the bit field of the most significant "1" in the input value.

The fractional part of the logarithm depends on the value of the remainder of the squared value following the top

most "1", noting that the actual position of this remainder is irrelevant. It is difficult to apply a formula to calculate the fractional part but can be readily and quickly found by using the remainder as a pointer to a look up table of fractions. In the original application the logarithmic output was primarily intended for use with a 12 bit DAC so that spectra could be displayed on an oscilloscope, thus only another 7 bits are required for the full logarithm. Putting the 5 bit integer part above the 7 bit fraction part effectively multiplies the base two logarithm by 128. An example of this procedure is shown below;

Squared value X = 00000000000001010101010101010 b.
= 174762 d.

Bit field of most significant "1" = 17.

Lower seven bits (rounded up) = 0101011 = 43

Lookup table :-

Lower 7 bits		Decimal value of the Log2 fraction		Fraction *128
00	-	0.0000000	-	00
01	-	0.0011227	-	01
42	-	0.4093909	-	52
43	-	0.4178525	-	53
44	-	0.4262648	-	55
127	-	0.9943534	-	127
128	-	1.0000000	-	128

The logarithm is thus equal to : (17*128) + 53 = 2229
(The true value of 128.Log2(174762) is 2229.15)

The output from this routine has a precision of ± half a

bit in eight (i.e. ±0.2%) over a 70 dB range, this is adequate for most applications but can easily be improved by extending the lookup table, this will of course require more memory but will take no longer to execute. An assembled printout of this routine is shown in appendix B.

4.5.8 Radix 2 FFT development.

To build up confidence and experience with TMS32010 assembler programs, and with the concept of the FFT, the first programs written were for 16 and 64 point transforms. These programs both used internal ram for data storage. Having built the necessary hardware for external ram the transforms were extended from 64 points, in factors of two, until a 1024 point transform was reached. It was found that the 64 point FFT could be executed in internal ram in 870 μs., and a 1024 FFT in external ram in 90 ms.

4.6 Radix 4 implementation.

In a 1024 point radix 4 FFT the radix 4 butterfly is performed 1280 times, so again the execution speed of the whole FFT is largely determined by the efficiency of the butterfly and of the time required to get data into and out of it. The radix 4 butterfly is considerably more complex than the radix 2 butterfly having three times the number of exponential coefficients and twice the number of complex input data. As with the radix 2 case the TMS32010 assembler program was developed from a 16 point transform, through a 64 point transform, to the full 1024 point FFT, so as to iron out any unforseen problems and errors at a simple stage. The radix 4 FFT was finally tuned for utmost speed by modifying it for real input data only. This modification was applied directly to the 1024 point transform. Similar consideration to those made for the radix 2 case are now covered for the radix 4 FFT.

4.6.1 Storage of constants.

The radix 4 FFT uses exponential coefficients from W[0] to W[3((N/4)-1)], this is demonstrated in figure 2.5 where N=16 and the highest exponential is W[9]. In the case where N=1024, the highest exponential is W[765], this means that 254 more complex constants have to be stored in the radix 4 FFT than the radix 2 FFT, and thus of course use 1/2 Kw more of program space.

Although the highest exponential required for the 1024 point transform is W[765] not all of the exponential up to this point are actually required. Refering to figure 2.5, exponentials W[5] and W[8] are not used, thus demonstrating that it is not neccesary to store all 766 exponentials. However, the TMS32010 FFT program accesses this large array of exponential constants via indexing pointers, and to subsequently make the pointers take into account the missing constants is a relatively complex and inevitably time consuming task. Another solution to reduce the number of stored constants is by taking into account the fact that when any x in the exponential term W[$\hat{x}$] is greater than N/2, the exponential is then equal to -W[(x-N/2)], thus bringing the highest required exponential back to W[N/2]. However this again involves some time consuming checks and arithmetic which is to be avoided if at all possible. For these two reasons it was deemed necessary to store all 766 complex constants in the radix 4 FFT program, even though some of them would never be referenced.

The same window as used for the radix 2 FFT is employed in the radix 4 FFT, this being the Kaiser-Bessel (beta=6) window. This requires 513 constants to be stored. A total of 2045 constants are thus required for a radix 4 FFT. and leaves only 2 Kw of program memory left for the FFT program.

4.6.2 Memory Transfers.

As with the radix 2 case, all of the data samples (stored in external ram) and all of the program constants have to be transfered to and from internal memory as each butterfly is performed. As explained earlier, the external ram in the hardware used to develop this algorithm and also that of IDDAS employs an I/O mapped address latch to point to I/O mapped ram locations. This latch auto increments when an external ram location is read from or written to. Hence to make the best use and efficiency of these memory accesses, four sets of butterfly data are transfered in one go using four sets of contiguous data. Subsequently four butterflies (of four complex points each) are performed at any one time on the data in internal memory.

4.6.3 TMS32010 Radix 4 Butterfly.

A radix 4 butterfly basically comprises one unmodified radix 2 butterfly, one radix 2 butterfly extended to include an exponential multiply of the first input as well as the second, and then a series of complex additions and subtractions.

The code generated to perform this task is shown in figure 4.3, note the following operands and definitions

$$A0(x) = A0R + jA0I \qquad A1(x) = A1R + jA1I$$
$$A2(x) = A2R + jA2I \qquad A3(x) = A3R + jA3I$$

$$W1KCS + jW1KSN = 2^{**}15(\cos(2\pi x/N) + j\sin(2\pi x/N))$$
$$W2KCS + jW2KSN = 2^{**}15(\cos(4\pi x/N) + j\sin(4\pi x/N))$$
$$W3KCS + jW3KSN = 2^{**}15(\cos(6\pi x/N) + j\sin(6\pi x/N))$$

```
;       radix 2 butterfly
;       A2R*Cos+A2I*Sin
ZAC
LT      A2R
MPY     W2KCS
LTA     A2I
MPY     W2KSN
APAC
SACH    TEMP1R,1
;       j(A2I*Cos-A2R*Sin)
ZAC
MPY     W2KCS
LTA     A2R
MPY     W2KSN
SPAC
SACH    TEMP1I,1
;       A0R+(A2*W2K)re
LAC     A0R
ADD     TEMP1R
SACL    TEMP4R
;       A0R-(A2*W2K)re
SUB     TEMP1R,1
SACL    TEMP5R
;       A0I+(A2*W2K)im
LAC     A0I
ADD     TEMP1I
SACL    TEMP4I
;       A0I-(A2*W2K)im
SUB     TEMP1I,1
SACL    TEMP5I
;
;       Modified r2 but.
;       A1R*Cos+A1I*Sin
ZAC
LT      A1R
MPY     W1KCS
LTA     A1I
MPY     W1KSN
APAC
SACH    TEMP2R,1
;       j(A1I*Cos-A1R*Sin)
ZAC
MPY     W1KCS
LTA     A1R
MPY     W1KSN
SPAC
SACH    TEMP2I,1
;       A3R*Cos+A3I*Sin
ZAC
LT      A3R
MPY     W3KCS
LTA     A3I
MPY     W3KSN
APAC
SACH    TEMP3R,1
;       j(A3I*Cos-A3R*Sin)
ZAC
MPY     W3KCS
LTA     A3R
MPY     W3KSN
SPAC
SACH    TEMP3I,1
;       (A1*WK)re+(A3*W3K)re
LAC     TEMP2R
ADD     TEMP3R
SACL    TEMP6R
;       (A1*WK)re-(A3*W3K)re
SUB     TEMP3R,1
SACL    TEMP7R
;       (A1*WK)im+(A3*W3K)im
LAC     TEMP2I
ADD     TEMP3I
SACL    TEMP6I
;       (A1*WK)im-(A3*W3K)im
SUB     TEMP3I,1
SACL    TEMP7I
;
;       Second stage of but.
;       A0R and A1R
LAC     TEMP4R,15
ADD     TEMP6R,15
SACH    A0R
SUBH    TEMP6R
SACH    A1R
;       A0I and A1I
LAC     TEMP4I,15
ADD     TEMP6I,15
SACH    A0I
SUBH    TEMP6I
SACH    A1I
;       A2R and A3R
LAC     TEMP5R,15
ADD     TEMP7I,15
SACH    A2R
SUBH    TEMP7I
SACH    A3R
;       A2I and A3I
LAC     TEMP5I,15
ADD     TEMP7R,15
SACH    A2I
SUBH    TEMP7R
SACH    A3I
```

Figure 4.3 - TMS32010 radix 4 butterfly

The number of instruction cycles required to perform the radix 4 butterfly is 69, this compares with 92 cycles required to perform four radix 2 butterflies which would do the same amount of computation. This in itself gives a significant saving in execution time, but to add to this, the data and exponential constants only have to be moved to and from internal ram half as many times. To fetch and store the data for four radix 2 butterflies requires 15 I/O instruction per butterfly (I/O transfers require 2 instruction cycles), giving a total of 120 instuction cycles. As compared to this the radix 4 butterfly requires 33 I/O fetch and store operations giving a total of 66 instruction cycles. The approximate overall total execution time for the two methods is thus : radix 2 - 212 cycles, radix 4-135 cycles, i.e. a 36% reduction in execution time.

Note that the first stage of the radix 4 FFT requires no multiplies at all and thus a considerably simpler butterfly to that shown in figure 4.3 is utilised. The butterfly used in the final stage of the FFT is also simplified to produce just the first two complex outputs rather than four, as the other two form part of the duplicated double sided spectrum.

4.6.4 The overall complex input radix 4 program.

A high level flow diagram of the radix 4 FFT is shown in figure 4.4. The computation time of this program in TMS32010 assembler code is 27mS and the total program space is $E90. This includes 2 Kw of constants, and code to input, bit reverse and window the inputs. It does not include the output conversion routines but does in fact leave enough space to include both. Not that the input and output routines for the radix 2 and the radix 4 FFT's are identical. As will be seen in the next chapter there is no need for the output routines to be included with the FFT.

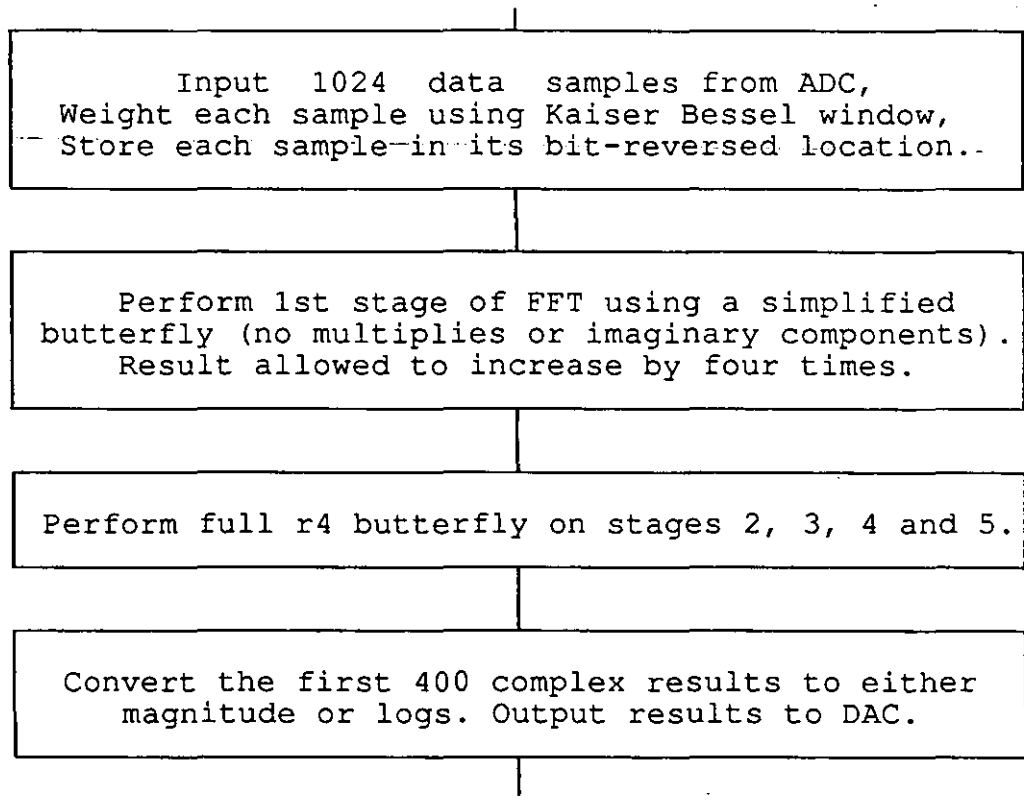


Figure 4.4 - Flow diagram of a radix 4 FFT

4.7 Real input radix 4 FFT.

In chapter 2 it was shown that the derivation of an FFT assumes the input data to be complex, but that by suitable manipulation real input data can be almost completely analysed by a half sized FFT. This manipulation has been applied to the TMS32010 radix 4 FFT to reduce computational effort even further. To demonstrate this manipulation it has been applied to a Fortran 77 program, this provides a compact example which can be understood more easily than in tens of lines of TMS32010 assembler code. Note that no high level or indeed low level language examples of this technique where found in other papers and so is included in Appendix C.

4.7.1 Fortran implementation.

Implementing the real input FFT is quite straight forward and can be split into three distinct and simple parts. These are as follows—

1) Perform an N/2 point FFT on the bit-reversed complex input array. Note that the bit-reversal is performed on consecutive input pairs (the first going in the real component and the second in the imaginary component) and that it is an N/2 point bit-reversal.

2) Separate the two resulting N/2 point spectra (including transformation of DFT(j.V) to DFT(V)) and create the final stage array of an N point FFT.

3) Perform the final stage of the N point FFT, the resulting spectrum being the same as if an N point FFT had been performed throughout.

4.7.2 TMS32010 Implementation.

The process used for the implementation of a real input FFT in TMS32010 code is exactly the same as that described above for Fortran. The starting block for this implementation was the radix 4 complex FFT, as described earlier. The modifications performed on this program were as following

1) The 1024 point radix 4 program performs five columns of radix 4 butterflies. To be able to split the FFT into two 512 point FFT's, the last column of the radix 4 FFT has to be split into two radix 2 columns. Thus the radix 4 program effectively becomes four columns of radix 4 butterflies followed by two columns of radix 2 butterflies.

2) The 1024 point input array was reduced to 512 points in which each complex point contained a pair of consecutive real samples. The address of each pair or complex point being calculated via a 9-bit "bit-reversal".

3) The code for the first four radix 4 columns plus the first radix 2 column reduced from 1024 points to 512 points.

4) A spectrum separation routine was created to split the two spectra (DFT(U) and DFT(j.V)) contained within the output of the 512 point complex FFT. The DFT(j.V) spectrum was also transformed to DFT(V) and joined onto the end of DFT(U) to form a 1024 point complex array.

5) The final radix 2 column (of modification 1) can then be applied to the resulting 1024 point complex array of above, without further modification.

No examples of the above TMS32010 code are given as it is all quite straight forward code. The only really new piece being the spectrum splitting, and this is just an exact translation of the equivalent code in the Fortran example.

4.8 Final FFT program as implememted in IDDAS.

The program coded for the TMS32010 as used in IDDAS and employing all the optimal features as described above is a real input, in place, decimation in time, radix 4, 1024 point algorithm, and takes only 17 ms to execute. Remembering that the complex input, radix 4 FFT took 27 ms to execute and the complex input, radix 2 FFT running on unoptimised hardware took 90 ms, it is apparent that very significant savings in computational effort have been made. This final program allows the overall throughput of data across IDDAS to be 1024 points in 25 ms, i.e. in excess of 40 KHz, thus giving a real

time bandwidth of 16 KHz, this is described in the next chapter.

CHAPTER 5

IDDAS concepts and description

5.1 Introduction to IDDAS.

The primary purpose of the Intelligent Dynamic Data Acquisition System (IDDAS) is to perform fast real time spectral analysis on signals obtained from engine/rig mounted transducers, and supply information in engineering units to engine drivers and development/measurement engineers to enable them to make decisions on engine health and test rescheduling. When IDDAS is combined with a host computer such as the PDP-11/73, together with a proprietary graphics card, a very powerful data acquisition, analysis and display system can be built, as will be shown in chapter 6.

IDDAS is comprised of an arithmetic card and an acquisition card linked together via two 40 way ribbon cables. This card system allows a multitude of signal processing techniques to be realised, such as IIR/FIR filters, convolution, correlation, and FFT analysis, together with a post processing facility to enable data reduction of spectra, correlations, etc.

The arithmetic card can perform a real 1024 point FFT in 17 ms, then perform data reduction algorithms on the results and finally pass the information onto a host computer. The acquisition card can filter an analogue input (51 KHz maximum), digitise the input to 12-bits at up to 131 KHz, and store up to 4096 current samples in a ring buffer. It can also count three clock (tacho) inputs. The sample rate, filter cut-off and tacho counter period are all programmable from the arithmetic card. The system has been designed such that other acquisition cards can easily be designed and

interfaced to the arithmetic card when specialised inputs are required.

This chapter provides a technical description of the more interesting and unique design features of IDDAS. This will hopefully give the reader a good understanding of the system's configuration and operation, without actually involving the mathematics of FFT analysis.

The design and development of IDDAS is split into the following stages;

1) Addition of a second TMS32010 processor to perform averaging, data reduction, etc.

2) Development of a programmable acquisition interface.

3) Acquisition of engine tacho signals for use in data reduction of spectra.

4) Addition of a third TMS32010 to act as a ring buffer of sampled data to allow overlap of transformed blocks.

5) Enable multi-channel input.

6) Develop interface between IDDAS and a supervisory host computer.

These stages are roughly in chronological order and cover a period of approximately one and a half years development (not including the TMS32010 FFT development). Finally a description is given of how IDDAS can be used with a host computer.

5.2 Transfer of spectra to a second processor.

As described in chapter 4, the TMS32010 can perform an FFT in 17 ms, however this is still in a complex form which must be

converted to magnitude to allow for further analysis. This operation takes between 10 and 20 ms (for 400 points) depending on the nature of the data and whether the output is linear or logarithmic.

As earlier explained, it is not the spectrum that the host requires but the information within it, thus further analysis is still needed. The resulting information must also be passed onto a host computer for calibration and display purposes. In a system where several tasks are being performed, throughput can be increased by using a multi-processing architecture, in general this can be arranged either for

1) parallel processing, each processor performing near identical tasks on a multiplexed input and output, or

2) serial (pipeline) processing, splitting the overall task into smaller tasks for each processor.

It was decided that a second processor configured in the serial or pipeline arrangement would be of most benefit for the following reasons

1) There would be no duplication of programs. This is important because the TMS32010 can only access 4Kw of program memory. The radix-4 FFT itself requires very nearly 4Kw of memory which does not leave much left for further data reduction algorithms. Thus, with a pipelined system a further 4Kw of memory becomes available for further analysis algorithms

2) Less hardware. If two processors work in parallel then multiplexing hardware is needed for the processors to interface with the input device (ADC) and the host computer. Timing circuitry would also be required to ensure that the two processors worked on different input data. The only hardware required for pipelined

processors is a simple latch and handshake facility between the processors.

3) Simpler management of the board. In a pipeline system specific tasks can be allocated to each processor eg. one processor can take care of setting sample rates and cut off frequencies, communication with the host, dealing with error conditions, etc, without risk of confliction with a parallel processor.

4) Different processors. As different tasks are being performed by the processors in a pipeline structure, the processors can be chosen optimally to suit their tasks. For example, if the code in the second processor needs to be changed fairly often by someone not experienced in TMS 32010 assembler code then a processor for which a high level language is available could be used (eg. a 68000 running 'C').

It was decided to use another TMS32010 as the second processor as this gives a high level of compatibility, no learning curve (as might be required for an unfamiliar processor), and most importantly, high speed for further data processing. It was also decided to split the overall task after the FFT algorithm.

Thus the first processor (referred to as the FFT processor) gets the input data, performs windowing, bit reversal, an FFT algorithm and then passes the resulting complex spectrum onto the second processor.

The second processor (referred to as the Data Reduction (D.R.) processor) receives the complex spectrum, converts 400 complex points to magnitude, performs ensemble averaging, performs some form of information extraction or data reduction algorithm specific to an application, and then finally passes the information onto a host computer. It can also read three engine speeds, set the sample rate and the

filter cut-off frequency.

The only disadvantage with the pipelined structure is that data must be passed from one processor to the next which obviously reduces data throughput. However, if the transfer is tightly controlled then handshaking delays can be kept to a minimum. Hand-shaking with the TMS32010 is limited to two asynchronous single line inputs;

1) BIO, and 2) Interrupt.

It was considered preferable to leave the D.R. processor interrupt line for use by the host computer, thus the transfer mechanism had to make use of the BIO line. To achieve the fastest possible transfer rates the following design was conceived, reference should be made to figure 5.1

A simple latch with tri-state outputs (74LS374) is used as an intermediate store between processors, a D-type latch is used to clock out a '1' to the BIO of the D.R. processor when data is written to the latches by the FFT processor and cleared when data is read from the latch by the D.R. processor. Assuming that the FFT processor always transfers a spectrum after each FFT, and that the D.R. processor is polling the BIO line waiting for the first data transfer of a spectrum, then it is guaranteed that the D.R. processor will respond to a change of the BIO line by reading the latch within a period of 0.8 to 1.2 μs. If the FFT processor then continues to output to the latch at regular periods, the D.R. processor will be able to read this latch at these regular periods without getting out of step or receiving corrupt data. This does of course rely upon the two processors using the same clock period, they do not have to use the same physical clock although this does add an extra level of reliability. Thus the handshake facility has only really been used in the first data transfer and the actual transfer rate can occur at 1 word per 1.2 μs (6 instruction cycles) as will be demonstrated.

As extra level of integrity can be incorporated which allows the D.R. processor to operate totally asynchronous to the operation of the FFT processor. This is achieved by including a small amount of protocol within the initial part of the data block to allow the D.R. processor to synchronise to the start of a transfer. This process allows the D.R. processor to start polling its BIO line whenever it requires new data and guarantees that the data will be good.

This protocol is in the form of a code which is sent directly before the spectrum and must of course be distinguishable from the data within the spectrum. When transferring spectra there is one value guaranteed not to appear within the transfer, this being negative full scale ($8000). If this is thus transferred by the FFT processor directly before the spectrum, then the D.R. processor can check for it before accepting any further data.

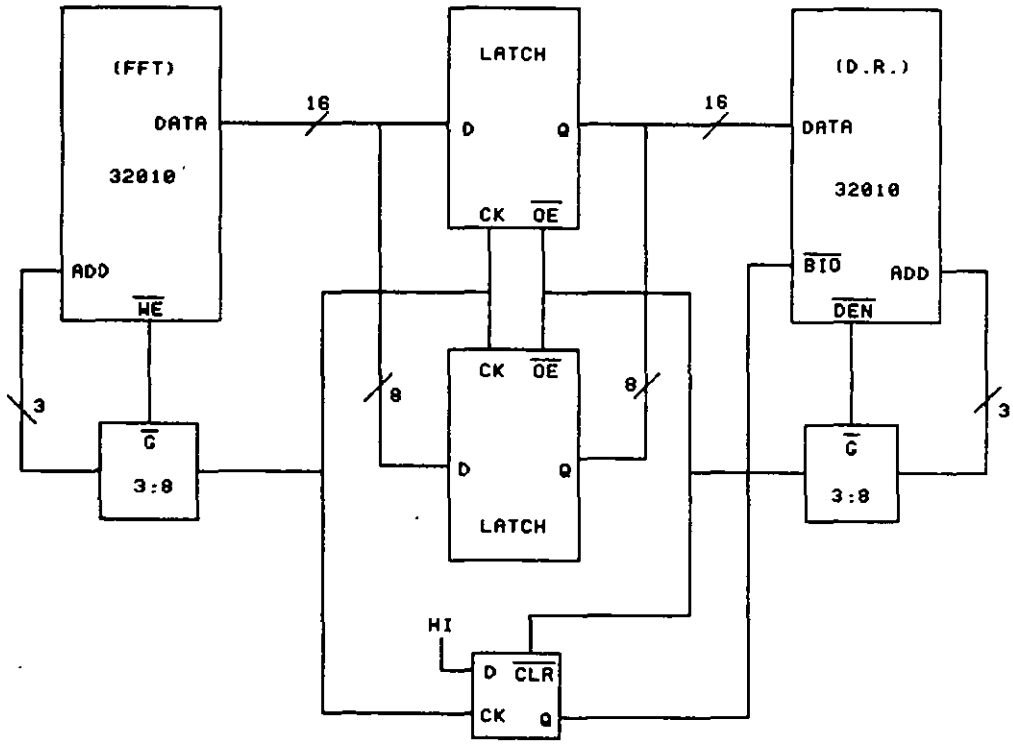


Figure 5.1 - FFT to D.R. processor transfer logic

The timing diagram for the transfer is shown in figure 5.2, the actual code required in each processor is as follows

FFT Processor

```
        THRE99  EQU  399
        STXFER  EQU  $8000
        ;
        ;
        LARP 0
        LAR  0,THRE99
        OUT  SPECTRUM,RAMADD
        ;
        OUT  STXFER,LATCH
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        ;
LOOP1:  IN   TEMP,RAMDATA
        OUT  TEMP,LATCH
        NOP
        NOP
        IN   TEMP,RAMDATA
        OUT  TEMP,LATCH
        BANZ LOOP1
```

D.R Processor

```
        THRE99  EQU  399
        STXFER  EQU  $8000
        ;
        LARP 0
        LAR  0,THRE99
        OUT  ZERO,RAMADD
        IN   TEMP,LATCH
        ;
LOOP1:  BIOZ LOOP1
        IN   TEMP,LATCH
        LAC  TEMP
        ADD  ONE,15
        BNZ  LOOP1
        NOP
        NOP
        ;
LOOP2:  IN   TEMP,LATCH
        OUT  TEMP,RAMDATA
        NOP
        NOP
        IN   TEMP,LATCH
        OUT  TEMP,RAMDATA
        BANZ LOOP2
```

There are four things to notice about this section of code :

1) The first "IN TEMP,LATCH" instruction is necessary in cases where transfers have already occurred from the FFT processor, this will clear the BIO line and allow the following "BIOZ" instruction to synchronise correctly.

2) The loop counter is initialised to 399 (giving 400 loops) not 799, due to the auxiliary register only being modulo 512 when counting. Hence the loop does everything twice.

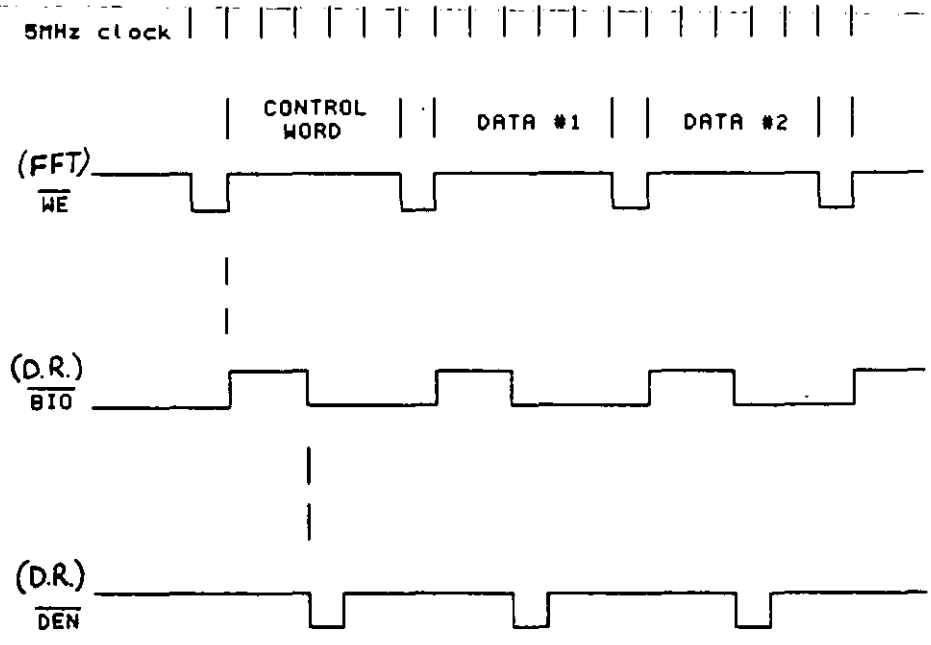


Figure 5.2 - Timing for FFT to D.R. data transfer

3) The delay between the first and second output of data from the FFT processor is 2 instruction cycles longer than that for the inputs to the D.R. processor, this is to change the initial output to input delay across the two processor from 0.8-1.2 μs to 0.4-0.8 μs.

4) The two apparently spare NOP's in the D.R. code are there for multiplexing reasons, as will be described in 5.5.

A spectrum block transfer between the processors can thus occur in just 960 μs, which means that it only takes 18 ms from the FFT processor acquiring a full set of input samples, to 400 complex points of the resulting spectrum residing within the D.R processor memory. This is very much closer to a real time process than achieved by most systems!.

5.3 Programmable acquisition interface.

To provide flexible data acquisition, the interface to the analogue world must allow variable sample rates with corresponding variable anti-aliasing filter cut-off frequencies. Most spectrum analysers provide a frequency scaling as follows

Bandwidth (Hz)	Frequency resolution
200	0.5
500	1.25
1000	2.5
2000	5.0
5000	12.5
10000	25.0
20000	50.0
50000	125.0

Note that the bandwidth is limited to the 400th FFT filter.

This type of scaling is not particularly optimal for engine analysis, or the TMS32010, as almost all data reduction algorithms relate the signal spectrum to engine speed and use the following type of equation to locate where the various engine orders are within the spectrum.

$$\text{spectral filter} = \frac{(\text{engine speed} * \text{engine order})}{\text{frequency resolution}}$$

where engine speed is in hertz. The above division performed in the TMS32010 micro processor is relatively slow (6.4 μs) and could be executed more efficiently if the denominator were a power of two thus allowing it to be replaced by right shifts. Also, ergonomic considerations have been made in selecting the above scale, in that humans prefer ranges such as 1000, 2000, 5000, 10000, this of course does not matter to a machine. Thus, taking into consideration the above points

it was decided to use the ranges shown below.

Bandwidth (Hz.)	Frequency Resolution
200	0.5
400	1
800	2
1600	4
3200	8
6400	16
12800	32
25600	64
51200	128

This type of binary scaling also makes the hardware very straightforward, all that is required is a clock at the highest sample rate (131.072 KHz) and a simple divide by 2**N, where N is programmable, (this exists as the 74LS294 TTL device). The clock frequency is in fact 40 times 131 KHz because the smallest divide capable by the divider is 4, and a divide by ten is required to generate signals with the correct mark space ratio for the ADC, ie. a clock frequency of 5.243 MHz is used.

As stated earlier, the sampling rate will be programmed by the D.R. processor. To make the operation of the FFT processor handle any sample rates selected by the D.R. processor, it must either be told the sample rate and make certain program adjustments, or simply not care. To reduce software complexity the second option is clearly favourable.

To ensure that the processor can only read one piece of data per conversion and that it does not care about the sample rate, the configuration shown in figure 5.3 has been used. Basically the BIO line is set either when the "end of conversion" is unasserted or when a "read ADC" occurs from the processor, thus the processor will only ever see the BIO line asserted once per conversion (when it will read the

ADC), and never low during a conversion. Note that the BIO line is set a fraction before the next conversion is triggered to allow for latency between the "BIOZ" and "IN ADC" instructions. The timing diagram for this is shown in figure 5.4.

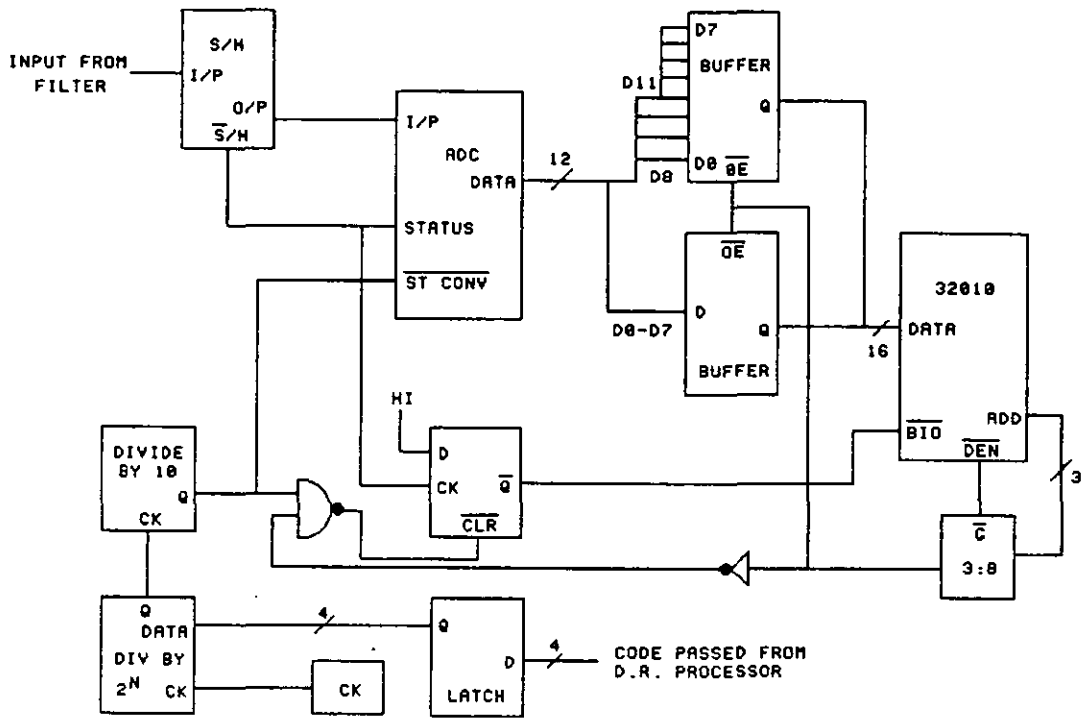


Figure 5.3 - Sampling and digitising logic

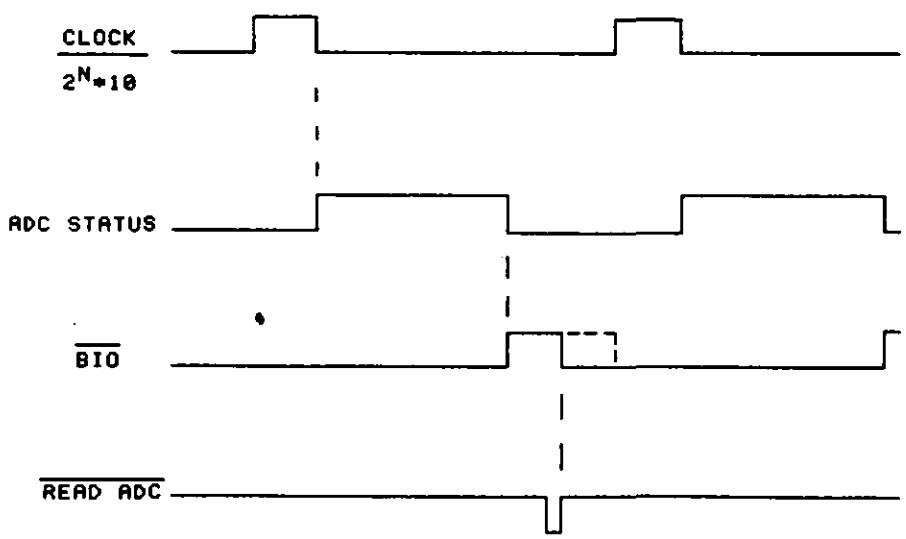


Figure 5.4 - Timing for sampling and digitising logic

Having made the sample rate programmable, the anti-aliasing filter cut-off frequency must also be made programmable. This is not a simple task as the filter must have a roll-off of at least 120 dB per octave and a stopband attenuation of 60 dB. This specification ensures that all aliased signals are attenuated by at least 60 dB between d.c. and the 400th FFT filter. Switched capacitor filters were considered (eg. National Semi MF 10), but these have a maximum cut-off frequency of 30 KHz and also give significant third order harmonic distortion. A 20 pole programmable analogue or digital filter could have been designed in house but this would have presented a significant demand on time and resources. As luck would have it, KEMO Ltd produced a new range of programmable filter board products in mid 1985. One of these was an 8-bit (256 binary step) programmable elliptical filter with any base cut-off frequency up to 200 Hz, a roll off of 135 dB per octave and a stop band attenuation of 80 dB, and all on a standard size euro-card!.

One of these Kemo filter cards (VBF33) with a base frequency of 200 Hz, allowing a top cut-off frequency of 51.0 KHz (255*200) was grafted into IDDAS. [Note that Kemo were not interested in licensing the art work]. The values required to set up the input sampling and corresponding anti-aliasing filtering are shown below

Bandwidth		Code to Divider		Code to Filter
200	....	$A	....	$FE
400	....	$9	....	$FD
800	....	$8	....	$FB
1600	....	$7	....	$E7
3200	....	$6	....	$EF
6400	....	$5	....	$DF
12800	....	$4	....	$BF
25600	....	$3	....	$7F
51200	....	$2	....	$00

Thus control over input acquisition is managed purely by the D.R. processor, which may obtain its commands from a host, and everything has been made completely transparent to the FFT processor.

5.4 Engine speed acquisition.

Nearly all data reduction algorithms require engine speed as one of the parameters, eg. location of fundamental frequency for vibration, or Nth engine order for blade flap analysis. The analogue speed signal from the majority of engines is a 60 pulse per rev output, ie. the output frequency is equal to revs per minute. To measure speed in hertz it is simply a matter of counting pulses (R) for a pre-determined period (T) and then using the equation

$$\text{Speed} = R/(T*60) \text{ Hz}$$

The timing period dictates how precisely speed is measured. If pulses can be counted during the timing period to within ± 1 pulse of the true value (the significance of one pulse is explained in figure 5.5), then the precision of the calculated speed is

$$\text{Precision} = [1/(60.T)] \text{ Hz}$$

If T is large then at constant speeds we obtain high precision, however during engine manoeuvres it will cause significant errors due to the non-stationary signal, in this case a more reliable result will be obtained (at the time of reading the counter) if a smaller value for T is used. For most situations it was considered that 1/4 second would be suitable, this conclusion was reached after considering the following worst case situation.

1) Sample rate = 1024 Hz; Thus resolution = 1 Hz.
2) FFT filter number of 15th engine order required.
3) Algorithm can look either side of filter for peak.

=> Precision of speed-measurement must be to ±1/15 Hz.
(ie. ±1 Hz. at 15th engine order)

Therefore => Counter period (T) = 15/60 = 1/4 second.

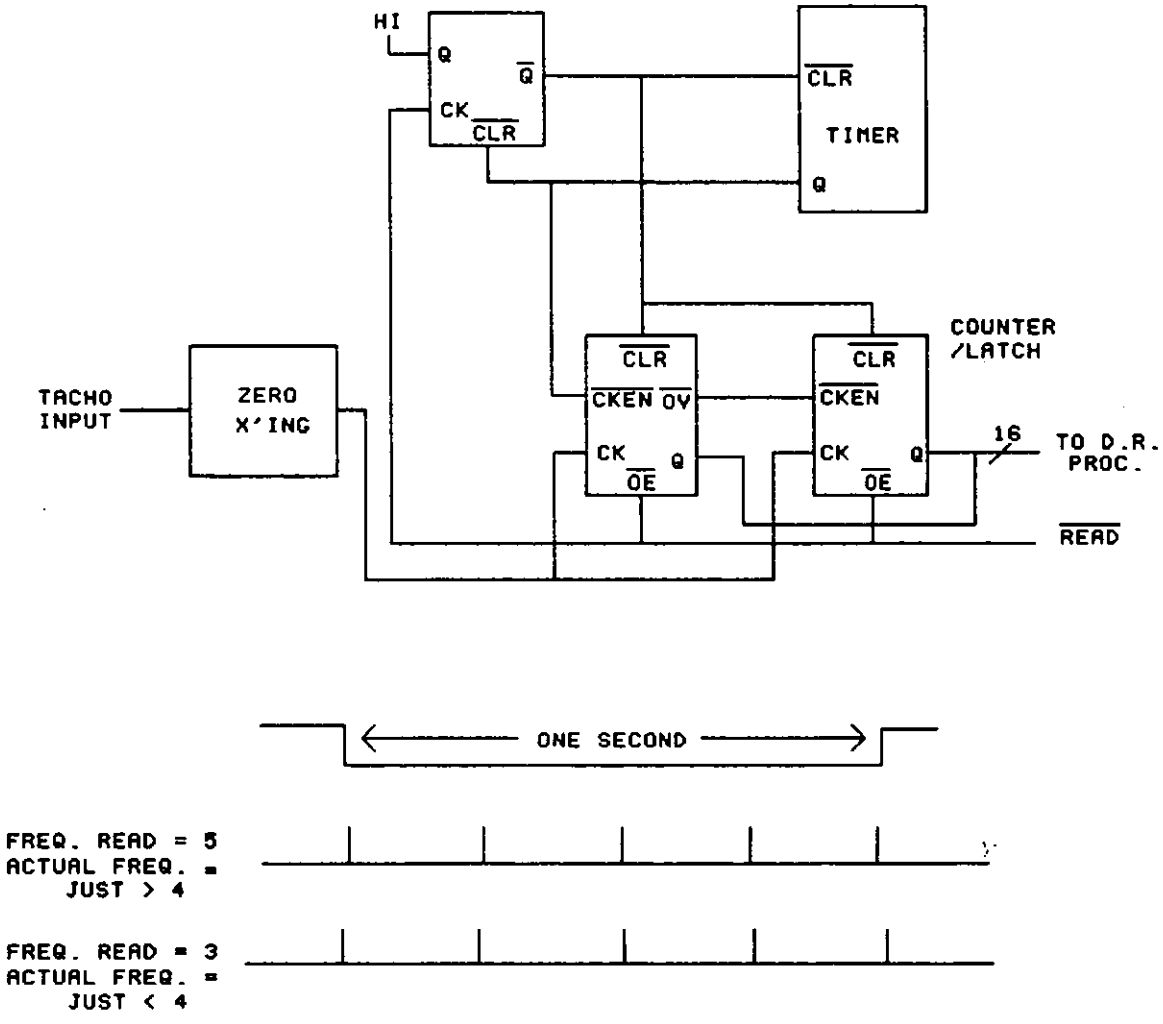


Figure 5.5 - Tacho counter logic.

The circuit used to perform the counting and timing is shown in figure 5.5. The best device found to perform the 1/4 second timing was the PXO-600, this is a programmable

timer/oscillator which will also allow time periods other than 1/4 second to be used. By using three of the top four bits left spare from the sample rate and filter programming word, the timing interval can be selected from one of the following by the D.R. processor

Bit 14	13	12	Timing Period (Sec)
0	0	0	0.0833
1	0	0	0.8333
0	1	0	0.1667
1	1	0	0.2500
0	0	1	0.3333
1	0	1	0.4167
0	1	1	0.5000
1	1	1	1.0000

Note that the D.R. processor does not need to synchronise its speed readings with the elapse of the timing period. The devices used (74LS590), and the special way they are connected, means that they are only updated at the end of each timing period, but that they can be read at any time without upsetting the timing period or the contents of the counters. Quite simply, the speed reading obtained will be that due to the last timing period. It should however be noted that once a timing period has elapsed, then the next period will not start until the counters have been read, i.e. they are not free running.

Analogue tacho signals can vary in amplitude from ± 1 volt peak to $\pm$ 30 volts peak. To cope with this, National Semiconductor LM1815 zero cossing detector chips have been used to buffer the tacho inputs and to trigger the counters. These devices will allow an input of up to 40 volts peak and incorporate adaptive thresholding to reduce triggering on noise.

5.5 Ring buffered input data.

The need for a ring buffer between the FFT processor and the ADC which allows the FFT processor immediate access to 1024 samples of data, came about from the following considerations

1) Although the FFT algorithm has been heavily optimised and tailored to run efficiently on the TMS32010, thus keeping the computation time to a minimum, it is of little consequence if the FFT processor then has to wait for up to a second to obtain the next 1024 sample points directly from an ADC (1.024 KHz sample rate).

2) The integrity and smoothness of a spectrum can be improved by ensemble averaging, as this tends to push down the noise floor and steady the amplitudes of the larger signals. Averaging however, slugs the response of transient changes within the spectrum and could result in averaging time constants of many seconds if spectra are not available fast enough.

3) Statistically, the best results are obtained from FFT's if their input data is overlapped, this is especially true when windowing is used, as is the case. It has been shown by A.H.Nuttal [33] that to obtain 98% of the signal energy density function then the overlap should be at least 60% (window dependent).

Hence faster throughput and overlapping of data is of significant benefit to the system. To enable this a ring buffer is required between the ADC and the FFT processor. Its task is to keep a buffer of the most recent 1024 samples (effectively storing the latest sample by over-writing the 1025th previous sample) and at the same time make the buffer available, on request, to the FFT processor. This is not a trivial task as the input data from the ADC may appear at a rate of up to 131 KHz (one per 7.6 µs) and at the same time a transfer to the FFT processor must occur as fast as possible.

To achieve this task a third TMS32010 has been employed!. The multi-tasking type operation is enabled by inputting data under interrupt control, and outputting data under BIO control. Each task has been made transparent and asynchronous to the other.

The input interrupt service routine simply reads in data from the ADC, increments the ring buffer address and then stores the data at the new address. Note however that this interrupt routine must be as efficient as possible because when an output block transfer is also being performed, the interrupt service routine will steal valuable processor time and cause longer transfer times to the FFT processor.

The output routine, when signalled by the BIO line, outputs the most recent 1024 samples to the FFT processor. The output timing being such that the samples do not appear to the FFT processor faster than it can cope with, noting that windowing and bit reversal is still performed between each sample by the FFT processor. The minimum time in fact being 4.6 μs per sample, hence if no interrupts occur during the block transfer, the transfer will take (1024*4.6) 4.6 ms. Of course when there are interrupts, as there inevitably will be, the transfer takes longer and at a worst case sampling rate of 131 KHz it will take 6.6 ms.

The ring-buffer hardware is shown in figure 5.6 and the software in Appendix D. The ring-buffer operates as follows

1) Input samples are read in from the ADC upon receiving an interrupt generated by a negative edge from the ADC status (end of conversion).

2) The output block transfer starts when the FFT processor latches a '1' onto the ring-buffer BIO line, the block length is not fixed but continues until the FFT processor resets the BIO line. Note that the data transfer starts

with the most recent sample and works backwards through the buffer.

3) The ring buffer size is actually 4096 samples, this is necessarily larger than 1024 to stop input samples from being written over the end of the 1024 point transfer buffer when sampling at high speed, i.e. if the transfer block size is 1024 samples then up to 3 samples can be inputted for one output before overwriting occurs. The large memory size also provides contingency for larger buffers (eg. for 2048 point FFT's).

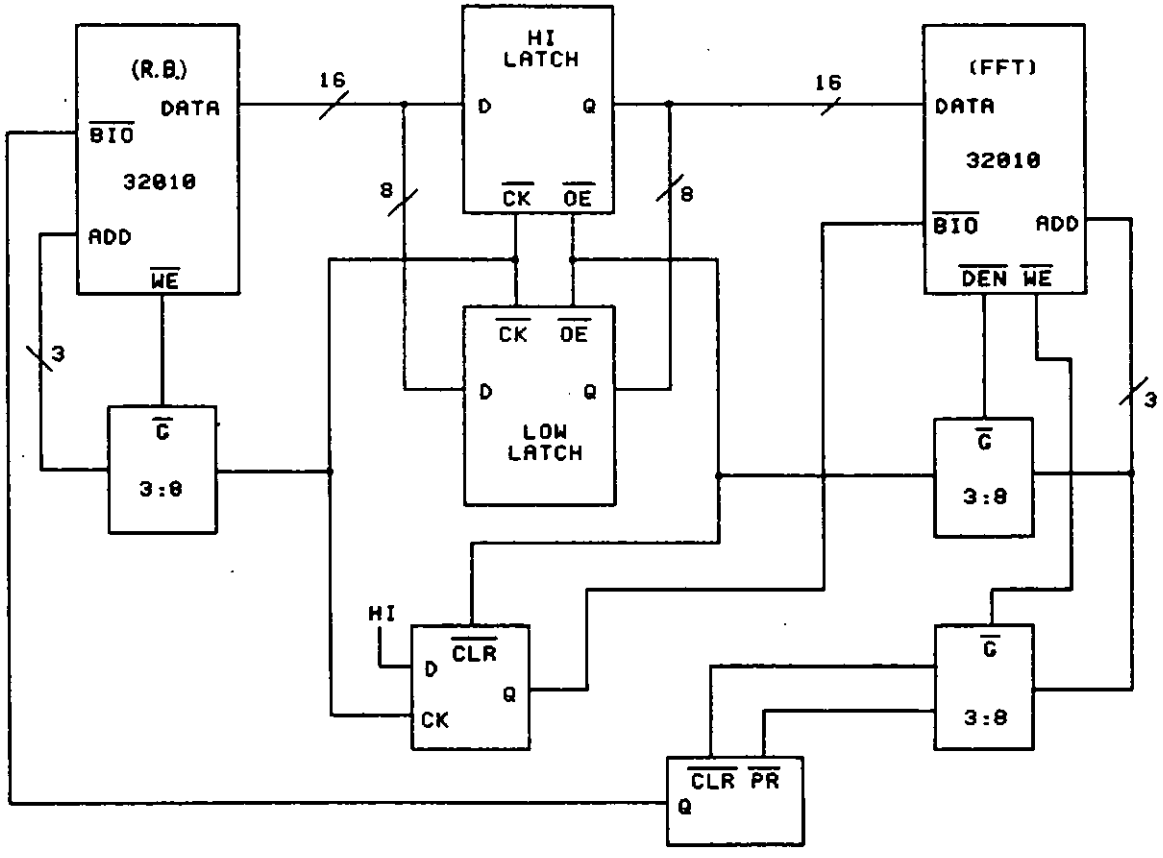


Figure 5.6 - Ring buffer logic

4) Each output to the latch between the ring-buffer and the FFT processor, sets the FFT processor BIO high and each input from the latch to the FFT processor resets the BIO line (this exactly emulates reading the ADC as described in 5.3).

It should be noted that overlap of the sample blocks which are sent to the FFT processor, does not occur for all sampling rates up to 131 KHz, the break point is in fact around 44 KHz (17 KHz bandwidth), as shown below

1) Time for FFT -> 17 ms,
2) Time for transfer from ring buffer to FFT processor -> 5.2 ms,
3) Time for transfer from FFT processor to D.R. processor -> 1 ms.

Total time for throughput of 1024 samples = 23.2 ms.
=> Sample frequency = 1/(0.0232/1024) = 44 KHz.

Hence if we sample faster, the ring buffer will have gathered more than 1024 samples between block transfers and data will be missed, and if we sample slower the ring buffer will receive less than 1024 samples between block transfers and overlapping of the blocks will occur.

The main point to note however, is that no matter what the sample frequency is, there will always be approximately forty spectra per second available to the D.R. processor for averaging and analysis.

5.6 Multi-channel input.

In many situations the signal bandwidth is much lower than the real-time bandwidth capability of the system described so far. Indeed in many cases lost sample data is also of no importance. Under these circumstances the system can be made

to operate more economically if multi-channels are allowed.

The system has thus been designed such that up to sixteen acquisition cards can be paralleled into one arithmetic card. To enable this facility the D.R. processor is able to output in software a four-bit address to the acquisition cards. These cards each then have a four-bit toggle switch to enable any one of sixteen decode addresses to be set up. Only when the addresses match will data be allowed to flow to and from the arithmetic card and an acquisition card. It can be seen that the real-time bandwidth of any one channel is now 16 KHz divided by the number of acquisition cards.

The time at which the D.R. processor changes channel number is very critical, it must be ensured that it does not do so while the FFT processor is part way through an input from a ring-buffer (as both are on different cards). This can be done by outputting the new channel number at the start of a spectrum transfer from the FFT processor to the D.R. processor. At this point the FFT processor will have just finished an FFT and be just starting the 400 complex point transfer, and thus will definitely not be involved with a data transfer from one of the acquisition cards. It should be noted that the FFT processor neither knows nor cares from which acquisition card it receives its input data.

The channel number output instruction replaces the two "NOP's" found in the D.R. processor transfer code (see section 5.2), the code now being as follows

```
LOOP1:  BIOZ    LOOP1
        IN      TEMP,LATCH
        LAC     TEMP
        ADD     ONE,15
        BNZ     LOOP1
        OUT     NEWCHANNEL,CHANNUMBER
LOOP2:  IN      etc
```

Note that the channels can be accessed in any order, or in fact in disproportionate amounts, ie. one channel could be accessed every other time while the others would be accessed in between, thus if there are eight channels, the first would have a real-time bandwidth of 8 KHz while the others would have bandwidths of 1 KHz.

This facility also allows a multi-input interface card to be designed were the channel number actually switches a multiplexor and the FFT processor then directly reads an ADC which is sampling the output of the multiplexor. This makes for a very economical system which is ideally suited to slow scan engine health monitoring type applications.

5.7 IDDAS to host interface.

One of the main objectives behind the IDDAS project was the requirement for a front end signal processing system which could quickly and easily communicate with a central data gathering and administrating host. This host would typically be situated within the test bed control room and could vary in computational power from an 8-bit micro such as the Syntel MC6809 computer to a 32-bit mini such as the Masscomp 5000 series or a Vax, though typically would be a PDP-11.

The following design constraints were imposed on the interface configuration

1) The interface should not use excessive amounts of host memory.

2) It should provide a directly accessible bidirectional control/status register.

3) Interrupts should be available in both directions.

4) The host should be able to transfer arrays of up to 2048 points to and from the D.R. processor.

5) The host should be able to download programs from host memory/disc to the D.R. processor, the ring-buffer and FFT processor programs remaining fixed in EPROM.

When considering computer to computer communication links, generally one of the following types of interfaces is used

1) serial data transfer, e.g. RS232,

2) parallel data transfer, e.g. IEEE 488,

or 3) direct memory to memory transfer, e.g. via globally shared memory or DMA transfers.

The transfer needs to be fast which eliminates serial methods and to some extent parallel methods due to the handshake overheads. Memory to memory transfers under processor control typically allow transfer rates of up to 500 Kb/sec, whereas DMA transfer can be at least twice this speed but does require added hardware and is not particularly flexible. It was considered that a shared global memory approach was best suited to IDDAS due to its ease of implementation, speed and flexibility.

Most data transfers to and from the host computer will consist of an array of data varying in length from perhaps 10 values (eg. 10 largest peaks in a spectrum) to 2048 values (eg. a complete complex spectrum), and the data arrays will almost always be in contiguous memory locations. Under these conditions it would be wasteful and prohibitive to directly map large areas of shared memory into host memory space. By employing an addressing technique similar to that used to map ram into the TMS32010 I/O space (ie. by implementing an auto-incrementing address register) the actual amount of host memory locations required can be kept to a bare minimum.

Infact this approach allows the whole memory block to be accessed via only two memory locations. Including a third location for the status/control register we have

Memory address	Write	Read
0	IDDAS control, reset interrupt, etc	IDDAS status
1	Memory data in	Memory data out
2	Memory address	not used

The main problem with global memory is that only one processor can access it at any one time, also the TMS32010 can not tri-state its data and address buses. Therefore to allow both the TMS32010 and the host access to the same memory, dual port memory is required. Initially this was implemented with standard static ram and multiplexed tri-state transceivers, however in late 1985 Integrated Devices Ltd began marketing 1Kb and 2Kb fully implemented dual port ram devices (this was followed in early 1986 by Advance Micro Devices with pin compatible devices). This enabled a neat and compact way of providing a shared memory between IDDAS and a host. The D.R processor accesses the dual port memory as part of its normal I/O mapped ram and the host accesses it as described above.

The hardware configuration showing memory and status/control implementation is shown in figure 5.7, note that 2Kw of dual port ram has been allocated for bi-directional transfer of variables, spectra, etc, and 2Kw for downloading programs from the host to the D.R processor.

The status/control register becomes very useful during periods of data transfer to stop memory contention occurring, ie. the status/control register can be used for handshaking and allowing only one processor to access the global memory at any one time. A typical sequence of events might be as follows

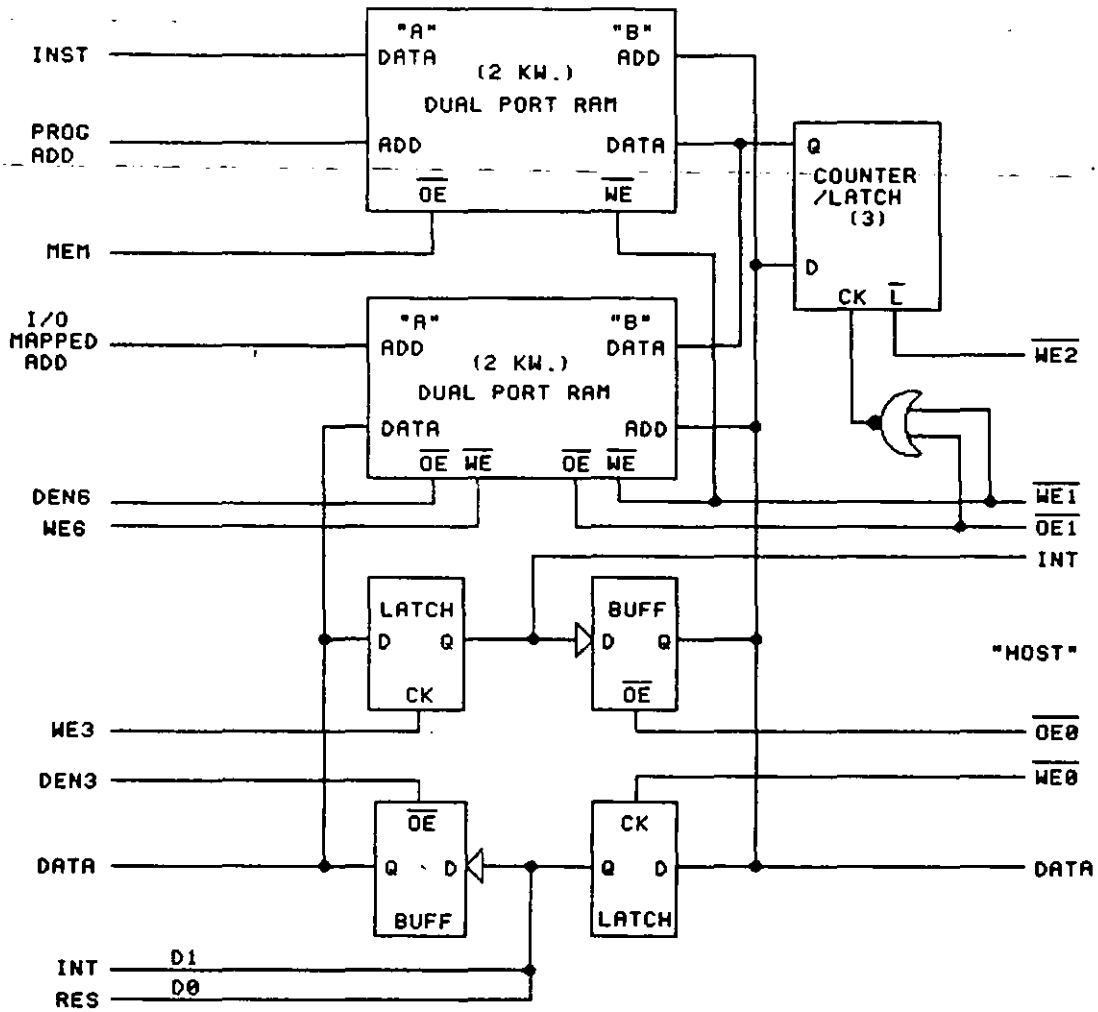


Figure 5.7 - D.R. processor/host computer interface

The TMS32010 D.R. processor performs magnitude conversion, averaging, data reduction, etc, and also periodically polls the host status register and tests the state of a specific bit (e.g. bit-2) to see if it has been set by the host, the state of this bit indicating whether the host is requesting access of the global memory or not. Note that any bit in this register can be used, but that it is not wise to use bit-0 or bit-1 as these are used to reset and interrupt the D.R. processor.

When the D.R processor finds bit-2 of the host status register set, it sets a bit in the host control register

(eg. bit-3) as an acknowledgement, and then either sits in a tight loop continuously polling bit-2 until it is reset, or performs some other task which does not require access to the global memory, while also periodically polling bit-2.

After seeing the D.R. processor acknowledgement, the host can then transfer data from anywhere between $000 and $7FF of the global memory to its own private memory for later analysis, finally resetting control bit-2 on completion.

The D.R. processor, upon seeing this change of bit-2, resets bit-3 of the host control register and continues with its normal tasks.

Note that the status and control registers as seen by each processor are completely separate, so in the above case both the host and the D.R. processor could have used bit-2 for their respective messages. There are of course many other protocol schemes based on this theme where for example the host is master and IDDAS the slave, or where interrupt control is used for faster responses. But the overall idea is that the risk of shared memory corruption by dual accesses upon the same memory locations is negligible.

The host can download programs to the D.R. processor by writing to the dual port ram at address $800 and above. However it should be noted that, although the TMS32010 can access 4Kw of eprom memory, only 2Kw of dual port program memory is available. To download a program, the host must first set bit-0 of the control register to reset the D.R. processor, and then transfer the TMS32010 binary code to the dual port ram from location $800 onward. Bit-0 can then be reset to allow the processor to restart. As with eprom based programs, the first memory location must contain the reset vector. Note that a two position link is provided to swap between an eprom or a dual port ram based program.

5.8 Hardware configuration.

Having explained the concepts behind the design of IDDAS, there now follows a concise description of the hardware configuration, control codes and protocol necessary to drive the system. It should be remembered that the FFT processor, the data reduction processor and their immediate peripherals are housed on the arithmetic card, and the ring-buffer, programmable filter, sample/hold, ADC and the three tacho counters are housed on the acquisition card. Circuit diagrams of the two cards are shown in appendix E, card inter-connections, analogue inputs and outputs, and all switch settings are covered in detail in the author's Rolls Royce report EIR00987 [10]. The two cards are also shown in photographs 5.1 and 5.2.

5.8.1 Memory mapping.

All three IDDAS TMS32010 processors can access the full 4Kw of program instructions in eprom, note that 8Kb eproms with 150ns access times are actually used, as 4Kb eproms with 150ns access times are not commercially available. The D.R processor can alternatively access 2Kw of program instructions in dual port memory (selectable by a link) for which the host computer also has access and is responsible for programming.

Due to the TMS32010 limitation of only having 144 words of ram (all on-chip), all three processors have been given extra I/O mapped ram to accommodate the large arrays which are used. Each processor has one output port interfaced to an auto-incrementing address register, and an input/output port interfaced to the ram data bus, note that reading or writing to the latter increments the address register. Note also that 2Kw of the D.R. processor's I/O mapped ram is dual ported and accessable by the host to facilitate data transfers. External ram for each TMS32010 is mapped as

follows

```
Ring-buffer    :  $0000 -> $0FFF  --  static ram
FFT processor  :  $0000 -> $0FFF  --  static ram
D.R. processor :  $0000 -> $1FFF  --  static ram
                  $2000 -> $27FF  --  dual port ram
```

The host computer addresses the above dual port ram from $000 to $7FF and the program instruction dual port ram from $800 to $FFF. These are also addressed via an auto-incrementing address register.

5.8.2 Input/output port addressing.

The TMS32010 can address up to 8 bidirectional 16-bit ports which it uses to output control codes to, and input data from, the real world. These are configured as follows

RING-BUFFER

Port	Read	Write
0		12-bit DAC
1	ADC	Parallel link to FFT processor
2		Ram address register
3	Ram data	Ram data
4-7	not used	not used

FFT Processor

Port	Read	Write
0		End transfer
1	ADC/Ring buffer	Start transfer
2		
3		12-bit DAC
4		Parallel link to D.R. Processor
5		Ram address register
6	Ram data	Ram data
7	10-Bit reversal in	10-Bit reversal out

A read from the 10-Bit reversal register returns the reversed lower 10-Bits of the last input to the register, the top 6-Bits always return zero, for example

D.R. Output = $0100001110011001
D.R. Input = $0000001001100111

D.R. Processor

Port	Read	Write
0	Counter/speed 1	Counter/Sample/rate Filter cut-off control
1	Counter/speed 2	
2	Counter/speed 3	
3	Host status register	Host control register
4	Parallel link FFT processor	DAC
5		Ram address register
6	Ram data bus	Ram data bus
7		Channel number

The "Counter period/Sample rate/Filter cut-off" control word is configured as follows:-

Bits 0 -> 7 : Filter cut-off frequency.
Bits 8 --> 11 : Sample rate.
Bits 12 -> 14 : Counter period.

The sample rate control bits are shown below together with the filter cut-off control bits necessary to provide anti-alias filtering after the 400th line of a 1024 point FFT.

Sampling frequency (KHz)	Band-width (KHz)	Resolution (Hz.)	Control bits 0->7	8->11
0.512	0.2	0.5	$FE	$A
1.024	0.4	1	$FD	$9
2.048	0.8	2	$FD	$8
4.096	1.6	4	$F7	$7
8.192	3.2	8	$EF	$6
16.384	6.4	16	$DF	$5
32.536	12.8	32	$BF	$4
65.536	25.6	64	$7F	$3
131.072	51.2	128	$00	$2

The counter period control bits are as opposite -

Timing Period (Sec)		Bits 14,13,12
0.0833	---	0 0 0
0.8333	---	1 0 0
0.1667	---	0 1 0
0.2500	---	1 1 0
0.3333	---	0 0 1
0.4167	---	1 0 1
0.5000	---	0 1 1
1.0000	---	1 1 1

The D.R. processor to host computer status/control bits are wired as follows

Host control register:	Bit 0	--- Host interrupt
(D.R. stauts register)	Bit 1-7	--- General purpose
D.R. control register:	Bit 0	--- Reset IDDAS
(Host status register)	Bit 1	--- D.R. interrupt
	Bit 2-7	--- General purpose

Note that a 12-bit DAC has been included as an output for each of the three processors. These are primarily intended as debugging and diagnostic aids but may also have uses in real applications.

5.9 Host interface and addressing.

The majority of, and certainly the initial, applications that IDDAS will be employed in, involve interfacing with a PDP-11 mini computer. For this reason the two cards making up IDDAS have been constructed to Q-bus quad-card dimensions and designed to draw their +5 volt supply directly from the Q-bus. The ±15 volt supply for the analogue circuitry is generated by a dc to dc convertor on the acquisition card. To make the PDP-11 interface compact and reliable but also to make it possible to interface IDDAS to other computers, two versions of the arithmetic card have been designed, as follows

1) This version has an on-board, commercially available standard Q-Bus interface, this provides all the necessary decoding and address selecting for four contiguous 16-bit registers, plus interrupt control and interrupt vector address selection. Two sets of toggle switches allow the interface to be mapped anywhere within the standard PDP-11 I/O page (16000-177770 octal) and the interrupt vector to be anywhere between 000 and 376 octal. The

three interface registers (status/control, address and data) are accessed using the first three of these four decoded addresses. Note also that the fourth location is used to enable/disable the interrupt request onto the Q-bus. This arithmetic card can be plugged directly into a PDP-11 computer with no additional circuitry.

2) This version is exactly the same as the above, except that there is no PDP-11 interface circuitry, instead, the three interface registers are accessed via a 34 way IDC connector. Thus when using this card, an additional interfacing circuit is required between it and the host backplane. A circuit to perform this with the BBC micro-computer has been designed and is in use at R.R. Leavesden.

The arithmetic card shown in photograph 5.1 is intended for use with a PDP-11, i.e. version 1. The four registers described in this version are shown below

Memory address	Write	Read
0	IDDAS control, reset interrupt, etc	IDDAS status
1	Memory data in	Memory data out
2	Memory address	not used
3	Int enable/disable	

The interrupt request signal is enabled by writing "1" to location 3 and is disabled by writing "0".

5.10 IDDAS/PDP-11 interface software.

Having provided an interface between IDDAS and the Q-bus, it was then necessary to provide a set of standard software for the IDDAS D.R. processor and the PDP-11, so that application software can easily and quickly be written. The intention being that Fortran programs within the PDP-11 can set up

IDDAS for selected bandwidths etc, and then call in a spectrum whenever required for manipulation, information extraction, display, etc. Having developed and tested the data reduction algorithms in this high level enviroment, the algorithm can then be reprogrammed for the TMS32010 D.R. processor itself. Note however, that if the high level Fortran program is not short of time, there is no reason why the application program should not remain in Fortran and continue to use the standard software. This standard software is briefly as follows;

IDDAS: The ring buffer and FFT processor software remain as described earlier in this chapter, i.e. 400 spectral complex points are available to the D.R. processor approximately 40 times every second. The D.R. processor software (in eprom) contains routines to convert the 400 complex points to logarithmic or linear magnitude and to perform five different levels of exponential averaging. The selection of one of ten routines, as well as the selection of one of nine bandwidths, is determined by a control code contained within the shared ram which must be set up by the PDP-11. The resulting spectra are then stored in the shared ram also making them available 40 times a second to the PDP-11.

PDP-11: Several routines have been written in PDP-11 assembler which can be called from Fortran programs. These allow such things as, downloading code from a disc resident file to the D.R. processor's instruction dual port ram, uploading or downloading a data array anywhere in the D.R. processor's shared ram, issuing an interrupt to the D.R. processor, enabling/disabling interrupts onto to Q-bus, and setting up the bandwidth/magnitude type/averaging control code.

Note that some of the PDP-11 routines rely on the standard IDDAS software being resident, but others, such as IDDAS reset, can be used with any D.R. processor software. All the above routines are listed and documented in the author's Rolls Royce report EIR01064 [11]. Figure 5.8 shows an overview of the IDDAS hardware and the timing sequence of data through it. A more detailed schematic showing individual parts is shown in figure 5.9.

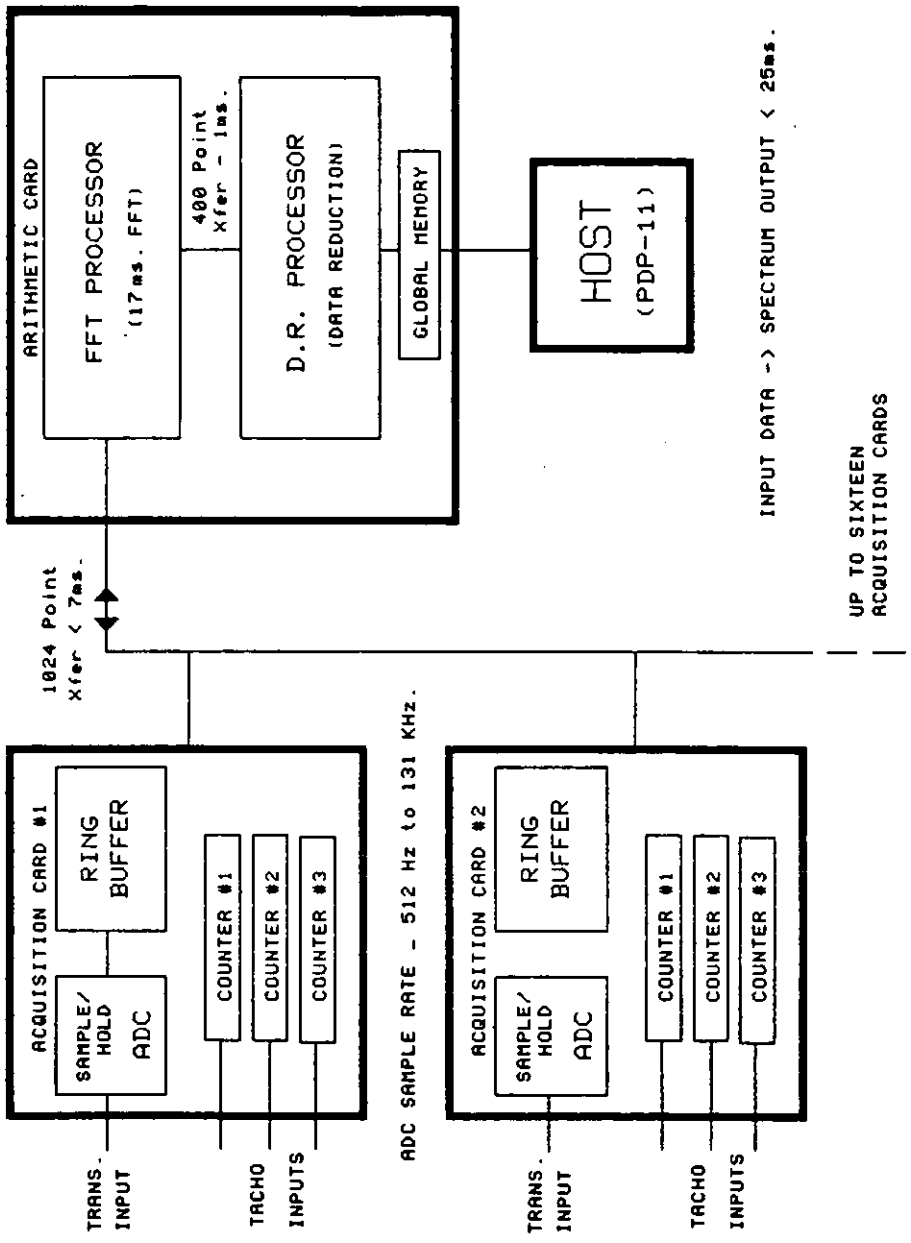


Figure 5.8 - Overview and general timing of IDDAS

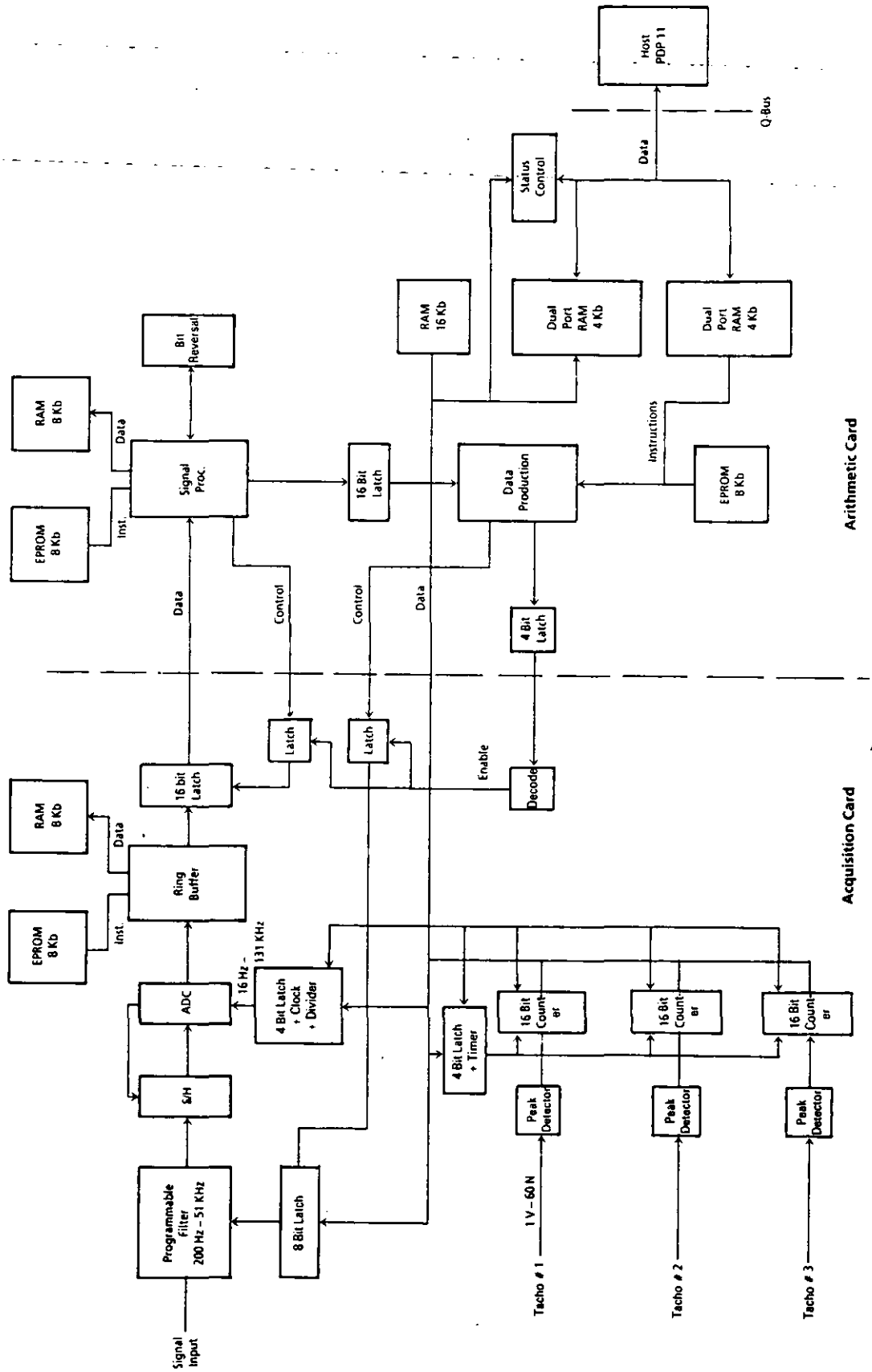


Figure 5.9 - Schematic diagram of IDDAS

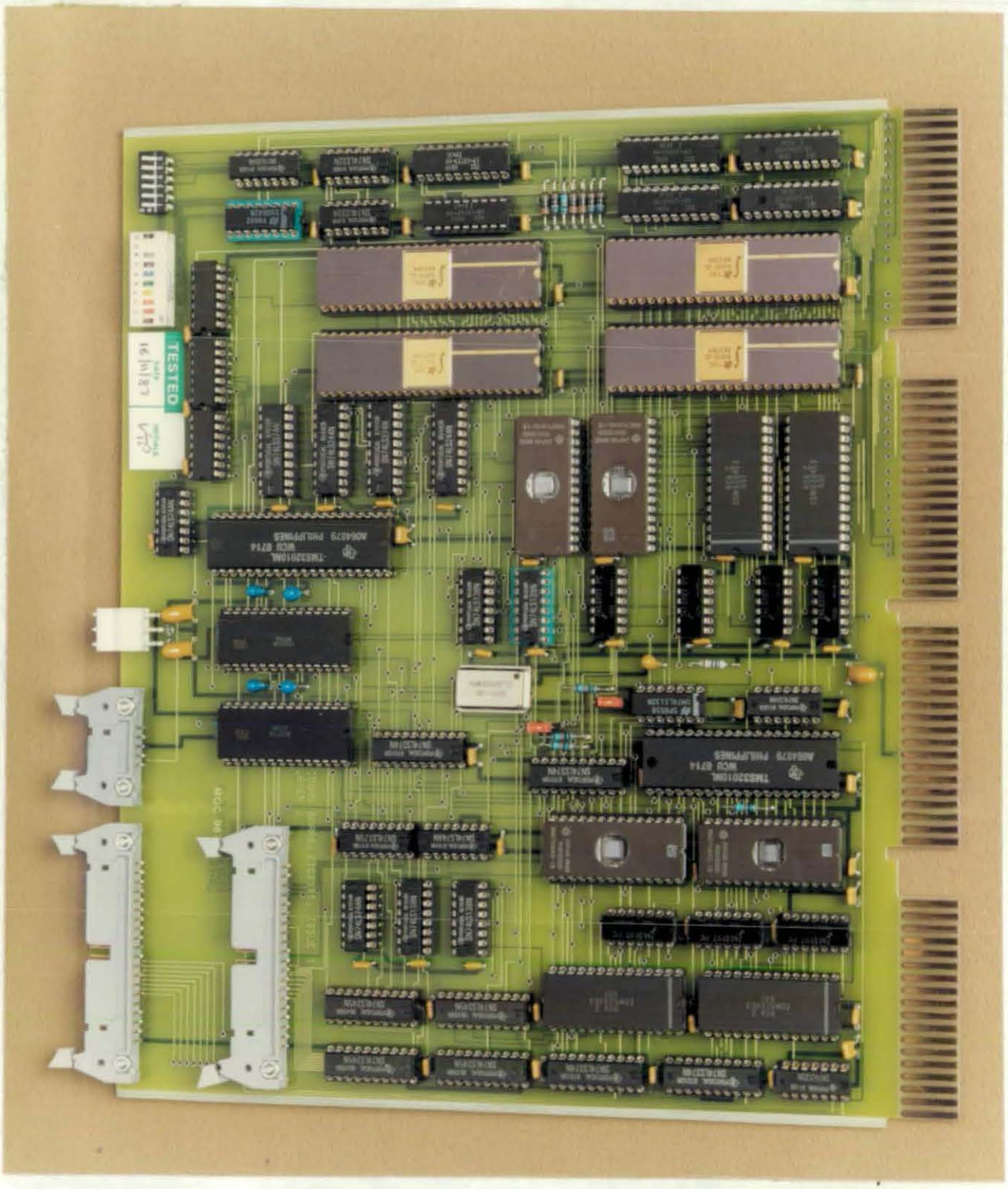


Photo 5.1 - IDDAS Arithmetic card

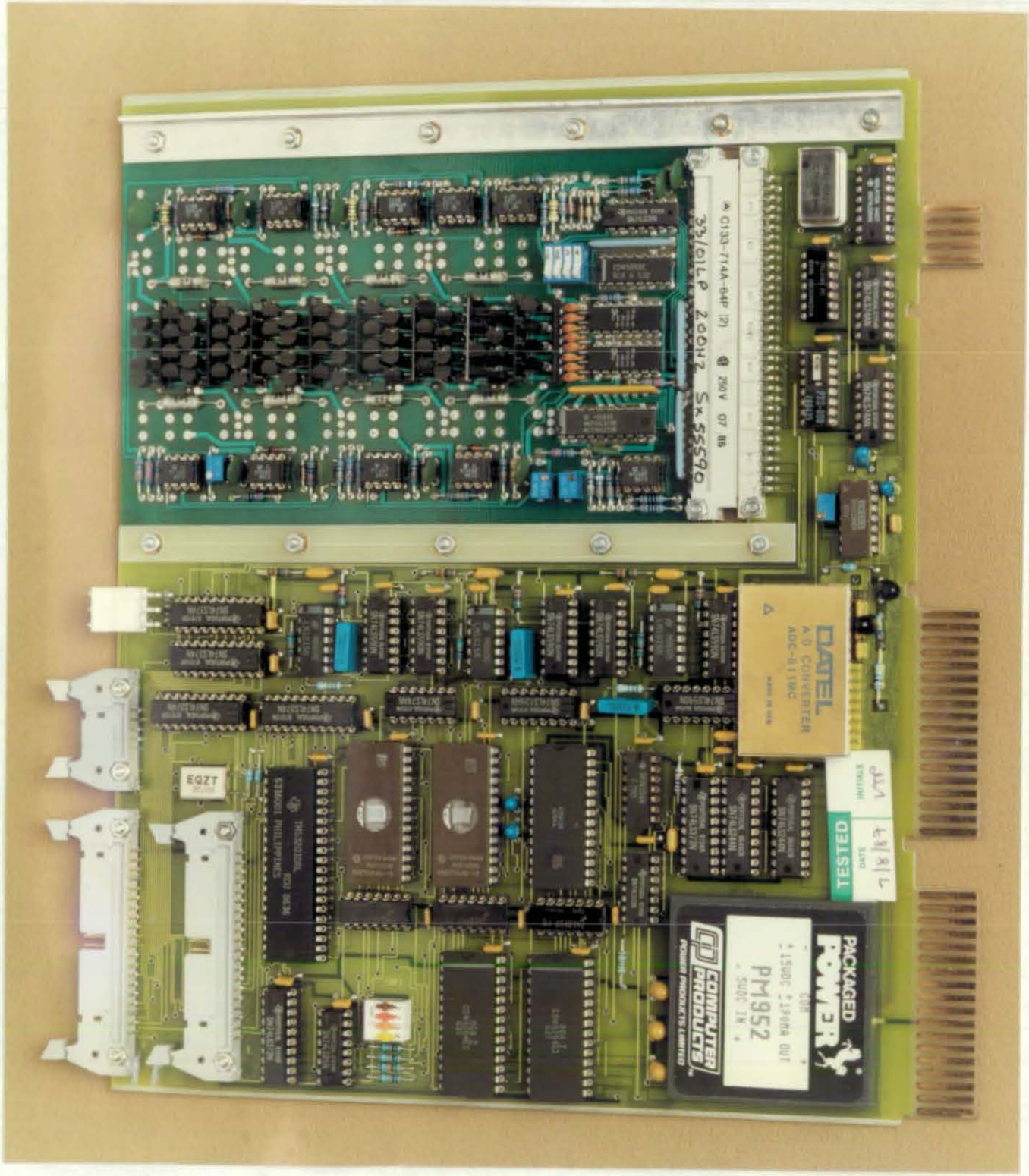


Photo 5.2 - IDDAS Acquisition card

CHAPTER 6

IDDAS Applications

6.1 Applying IDDAS to test facility dynamic signal analysis.

Throughout the design and development of the intelligent dynamic data acquisition system, there were many meetings and discussions held between the author and the measurement engineering, engine development, and data analysis areas of Rolls Royce. These concerned the type of data analysis, information extraction and real time displays that would be most useful and practical within aero engine test facilities. The details obtained from these meetings ensured that the IDDAS hardware covered as many of the potential applications as possible, and in the later stages, enabled basic application programs to be written to demonstrate to potential users the abilities of IDDAS using replayed engine signals.

The demonstrations of the prototype IDDAS unit working with a PDP-11 and a graphics generator on real, although replayed signals, showed its ability to analysis dynamic signals and determine engine health as laid down by certain criteria. These type of demonstrations had previously only been performed with large and expensive computing equipment, and thus served to spur on the use of many IDDAS units in real test facility applications.

The first two real applications of IDDAS are now briefly described. It should be noted how different the two applications are, demonstrating the versatility of both the software and hardware. The first application centres upon the speed at which spectra can be inputted into a PDP-11

computer for calibration, minor analysis, and display. The second centres upon the facility to perform data reduction on multi-channel inputs, prior to transfering extracted information to a PDP-11 computer.

6.2 Aero engine real time health monitor.

There are many occasions when the spectral content of a signal received from an engine transducer is not understood enough to enable data reduction algorithms to be defined. This is particularly the case when new engines or engine parts are being developed. On such occasions it is usually necessary to monitor the health of the engine or engine part during test running. Prior to the development of IDDAS, this was performed by watching the display of a spectrum analyser. The results obtained from this type of analysis are subjective and limited to observing instantaneous amplitudes at particular resonances. No trend analysis is performed and thus changes to the engine test schedule or test guidelines can not be made until a tape recording of the same signal has been passed through the off-line and remote dynamic data analysis and reduction system some time later. In many cases the engine may have already been derigged or worse still damaged during continued running.

Thus a requirement rose for an instrument which could provide a spectrum analyser type display, a history of spectra for trending, and a quick look facility showing such things as engine speeds, and amplitude and frequency of component resonances. It was also a requirement that such things as engineering units, scale ranges, spectrum bandwidths and test remarks/titles be selectable via menu driven tables. During engine running, the spectra also needed storing to enable replays, post analysis and hard copies of results to be produced immediately after an engine manoeuvre.

6.2.1 Hardware.

The standard test bed computer and graphics system used by the author's department (Electronics and Measurement techniques) at the time of this design was the PDP-11/73 with an 80Mb. winchester and a Gresham Lion SBD-B graphics card and high resolution monitor. The graphics card plugs directly into the Q-bus backplane, provides a 574x768 pixel colour display and comes with a suite of powerful Fortran drawing routines. The majority of the above specification can easily be performed by this hardware, except of course for the data acquisition and spectrum analysis. This is where IDDAS becomes vital, being able to transfer up to 40 spectra a second to the PDP-11, and in allowing the PDP-11 to tell it which of the nine possible bandwidths (200 Hz to 52 KHz) to use. Note also that IDDAS can transfer to the PDP-11 three engine shaft speeds.

6.2.2 Application software and operation.

As mentioned in the last chapter, a set of standard development software had already been written for IDDAS and the PDP-11, which allow IDDAS to be set up for bandwidth, and spectra to be inputted into the PDP-11 via high level Fortran calls. By using these standard routines (note that no data reduction is required from the D.R. processor) very little signal processing is required by the Fortran program. In fact the bulk of the application software is concerned with the relatively simple tasks of data manipulation for displaying and storage and with performing minor analysis such as finding the amplitudes and frequencies of the largest resonances. Essentially, the PDP-11 sees the spectra as nothing more than 400 point integer arrays that could have come from any memory resident peripheral device.

Calibration is performed by inputting into IDDAS a sinusoid of amplitude equal to some predefined engineering unit. The

resulting spectra are transfered by the usual means into the PDP-11 and the amplitude of the respective peak (in "banana" units at this stage) is stored for later use. Assuming that IDDAS is set up for linear output, then during normal running the spectrum input value are simply divided by the previously stored calibration figure to convert them into calibrated engineering units. Thus it can be seen that the actual gain applied to data as it passes through IDDAS is of no consequence. A high level and slightly simplified flow diagram of the real time part of this program is shown in figure 6.1.

The display produced by this program is shown in photo 6.1, note that the signals used to generate this were from a signal generator and not an engine transducer. A more realistic picture is shown in figure 6.2, this shows the vibration signature of an RB211-524D4 undergoing a two minute acceleration. Note that this figure has been produced by the replay facility of the machine on an ordinary dot matrix printer (hence being black and white) within the test facility itself.

Other points to notice are;

The two history plots (right hand side of picture) rotate around a four minute axis.

Up to 99 events can be marked anywhere on the time axis to allow post analysis at these points.

A marker can be moved up and down the spectrum plot, amplitude and frequency details at the marker are given in the table.

The relationship of the three largest peak frequencies and of the spectrum marker frequency to any one of the shaft rotation frequencies is also given in the table.

More information on this system can be obtained by refering to the Rolls Royce manual EIR14163. At the time of writing this thesis there were twelve of these systems installed in one of the Rolls Royce Derby site test facilities, enabling

very intensive monitoring of the latest Rolls Royce compressor in the V2500 gas turbine engine.

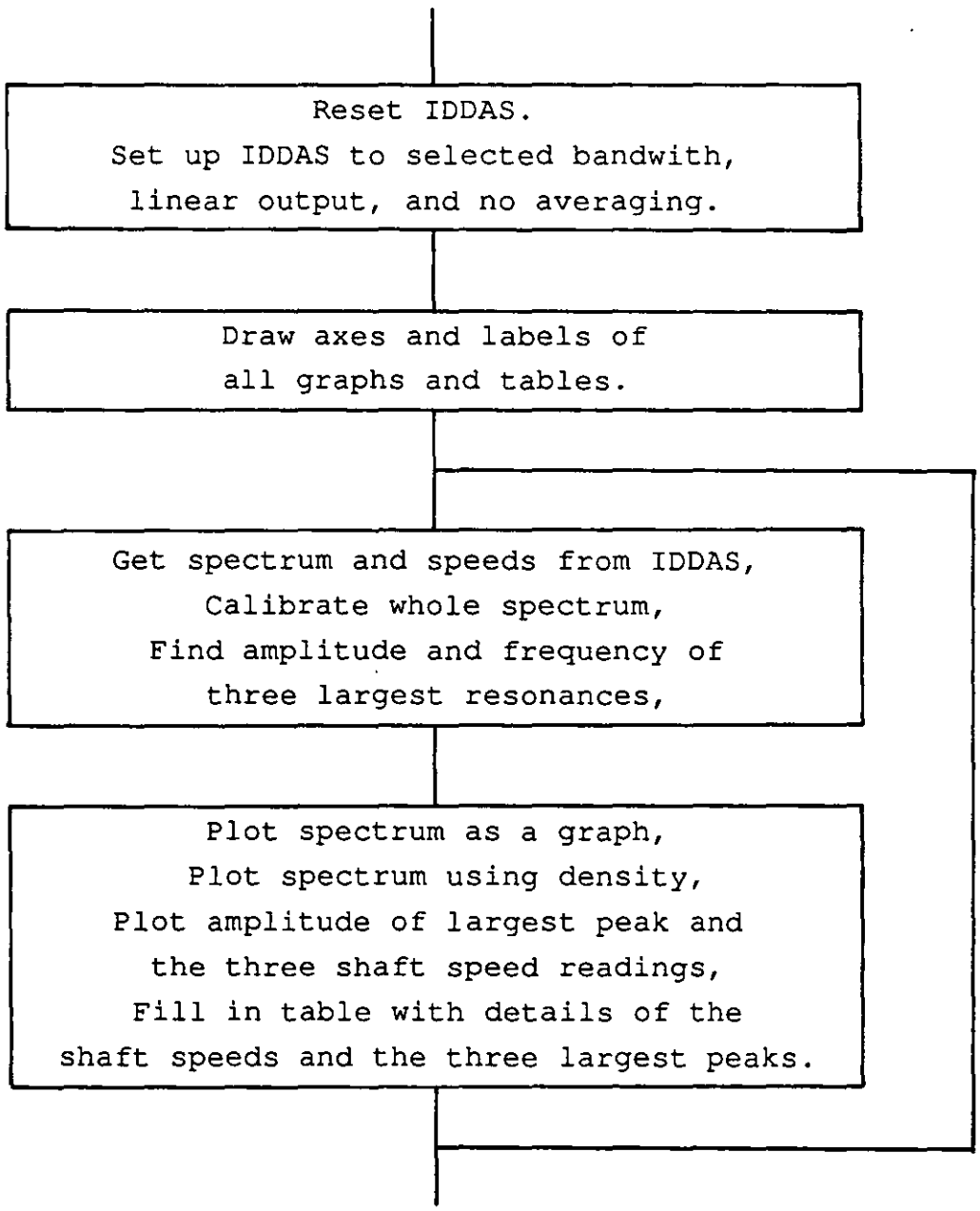


Figure 6.1 - Flow diagram of Real time monitor

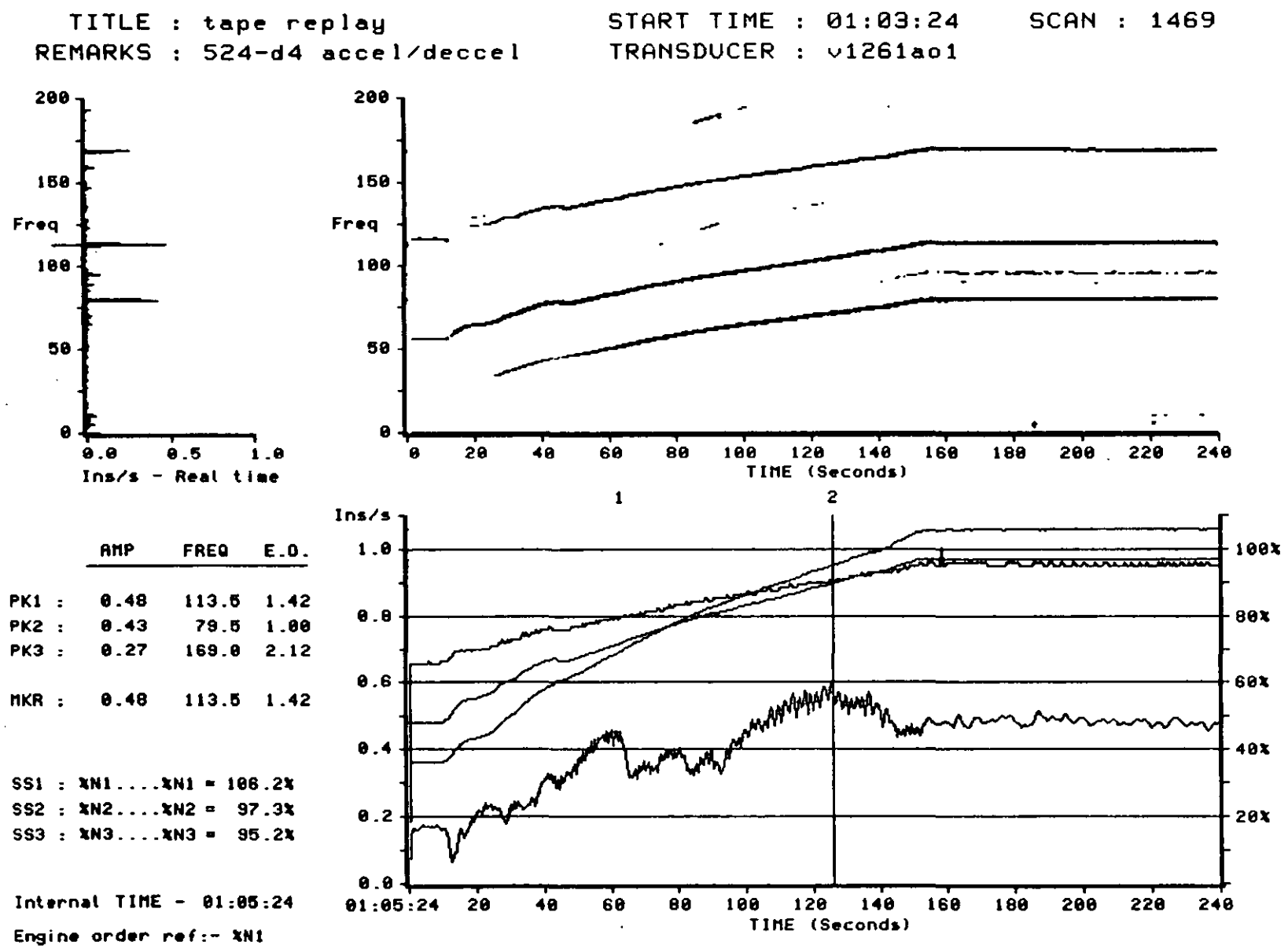


Figure 6.2 - Real time monitor, RB211 acceleration

6.3 Digital vibration system.

Aero engine vibration is found by measuring the conditioned signals obtained from engine casing mounted accelerometers. The conditioning consists of a charge amplifier and an integrator to provide a velocity voltage signal for the analysis. Vibration signals consist mainly of sinusoidal components of relatively high "Q" at the fundamental frequency of the rotating shafts.

For many years vibration has been measured in the test facilities by using band pass filters tracked to the engine speed. However these systems have a number of problems

Being completely analogue, they tend to drift from their operating points, thus calibration must be frequent.

They are tedious to calibrate as both vibration and speed signals with accurately set frequencies need to be injected while potentiometers are adjusted.

Different standards of tracking filters are required for just about every shaft of every engine type. This is because speed signals are usually generated from tachometers which are inevitably geared differently on each engine type. Hence filter modules must be changed everytime a different engine type is tested.

The vibration output (inches/second) is displayed in analogue form on a dial gauge, this means that recording of results is a manual task.

The configuration for this analogue system is shown in figure 6.3, note that broad band vibration is also measured to indicate when there is significant non-shaft related vibration.

As can be seen, vibration measurement in the test facility is

time consuming, operator dependent and produces poor quality results. Infact tape recordings of the vibration signals must also be made so that post analysis and plots of vibration levels can be more accurately produced. For such a straight forward analysis this is obviously wasteful of prime computer time and delays are again inevitable.

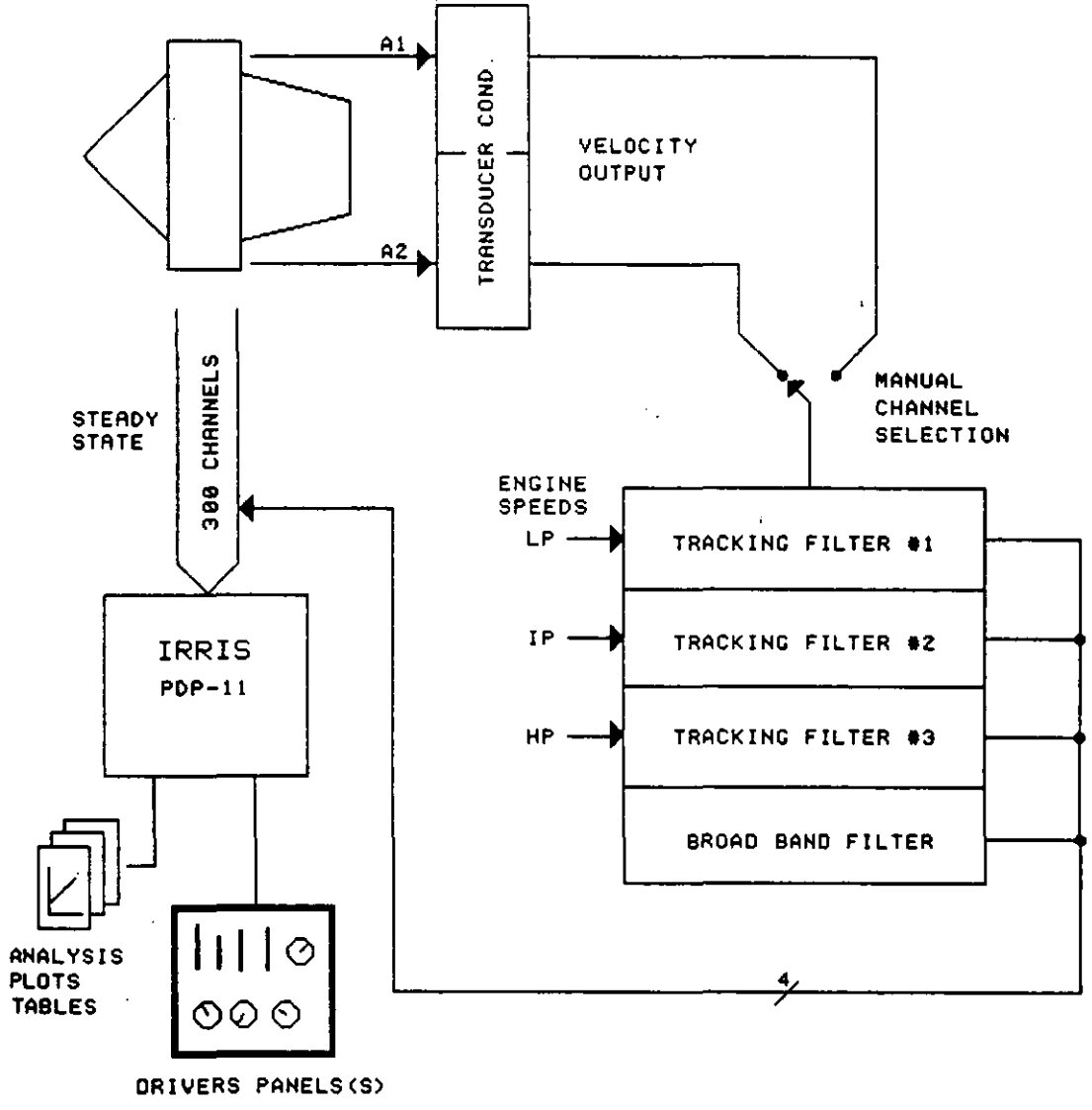


Figure 6.3 - Analogue vibration measurement

This unsatisfactory situation and the production of IDDAS resulted in a real time digital vibration system being designed, the basic outline specification of which follows

The system is capable of analysing three vibration channels simultaneously.

Vibration measurements are given for each shaft (two or three depending upon the engine) and for broad band.

There is no changing of hardware for different engine types, i.e. this is done under software control.

The results are made available to the test bed instrumentation computer (IRRIS) ten times a second for display purposes.

The calibration procedure is straight forward and short.

6.3.1 Hardware.

The test bed instrumentation computer mentioned in the specification (IRRIS) had already been installed in the test facility. This computer takes care of the displays and operator interaction, thus the only extra hardware needed to enable the vibration analysis was IDDAS. To perform this analysis one arithmetic and three acquisition cards are required, this allows three vibration signals to be acquired and stored while the arithmetic card sequences around the acquisition cards, performing a data reduction algorithm on each. All results are then be transfered to the PDP-11 over the Q-bus.

Only one arithmetic card is required because even when accessing three acquisition cards it can still analyse 40 spectra a second. Thus, shared between three input signals, the analysis is performed approximatley 13 times a second on each channel. The configuration of this hardware, as used in Rolls Royce Derby production test facilities is shown in figure 6.4.

6.3.2 Application software and operation.

Unlike the previous example where the standard D.R. processor software remained unchanged, this system required a major modification and addition to the software. In this case the D.R. processor does actually perform data reduction, or information extraction, on the incoming spectra. Note however that the FFT and ring buffer software is not modified.

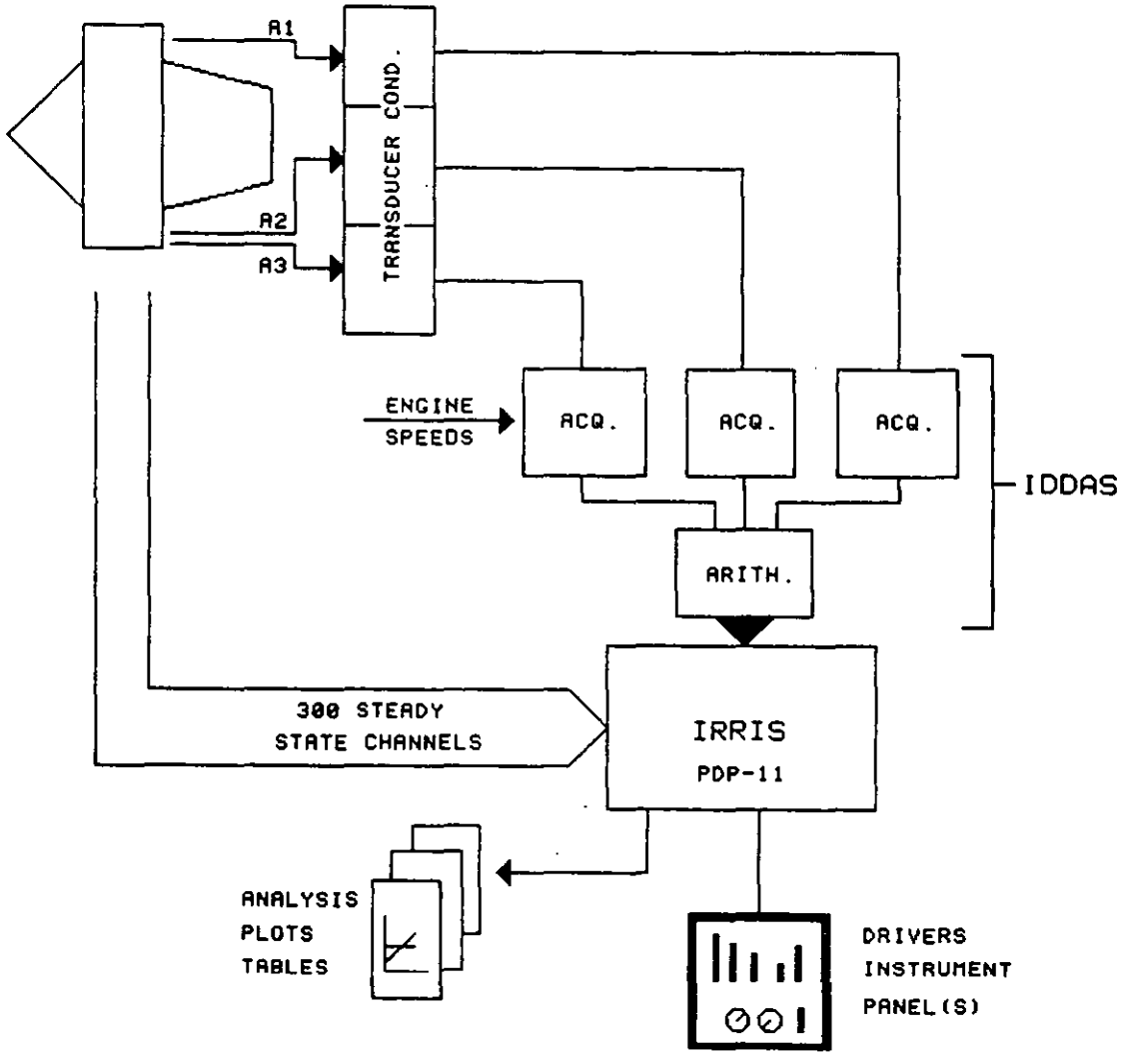


Figure 6.4 - IDDAS vibration system.

To cope with different engines the host computer first initialises IDDAS by passing to it three shaft to tacho speed multiplication factors, this allows the D.R. processor to

convert the speed signal measurement to shaft rotation frequency and allows it to track the shaft fundamental vibration component within the spectra. The host also passes the top and bottom frequencies for the broad band analysis and the actual bandwidth to which the acquisition cards should be set. This set up procedure, controlled from Fortran and host set up tables, allows the system to cope with any engine type including those not yet designed, thus providing a trully universal system. Once initialised, IDDAS continuously sequences around the three acquisition cards, each time storing the four vibration measurments per channel (three shaft related plus broad band) in the shared ram for use by the PDP-11 host.

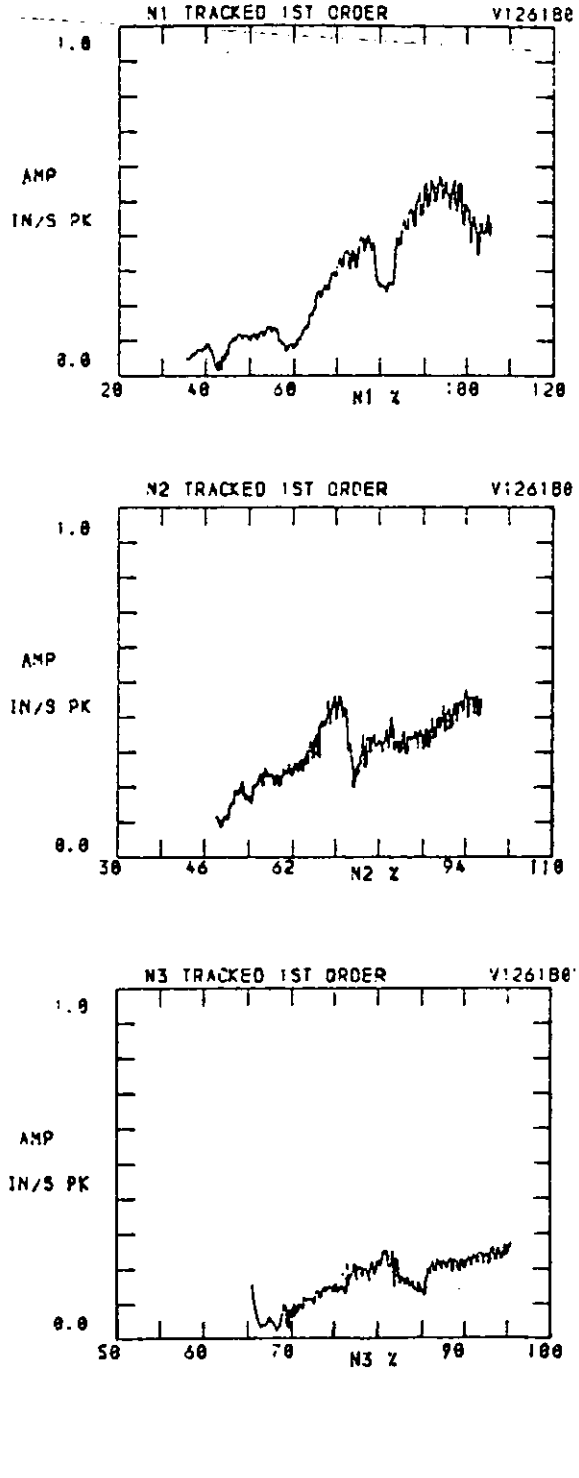
As was mentioned in chapter 3, the amplitude of a sinusoidal component varies by approximately 15% as it moves between Fourier transform filters. This sort of error is unacceptible for vibration measurement, hence the method of amplitude measurement described in chapter 3 has been employed. This involves using the filters each side of the central (or tracked) filter in a squaring, adding and square root procedure, the error in this measurement being significantly less than 1%.

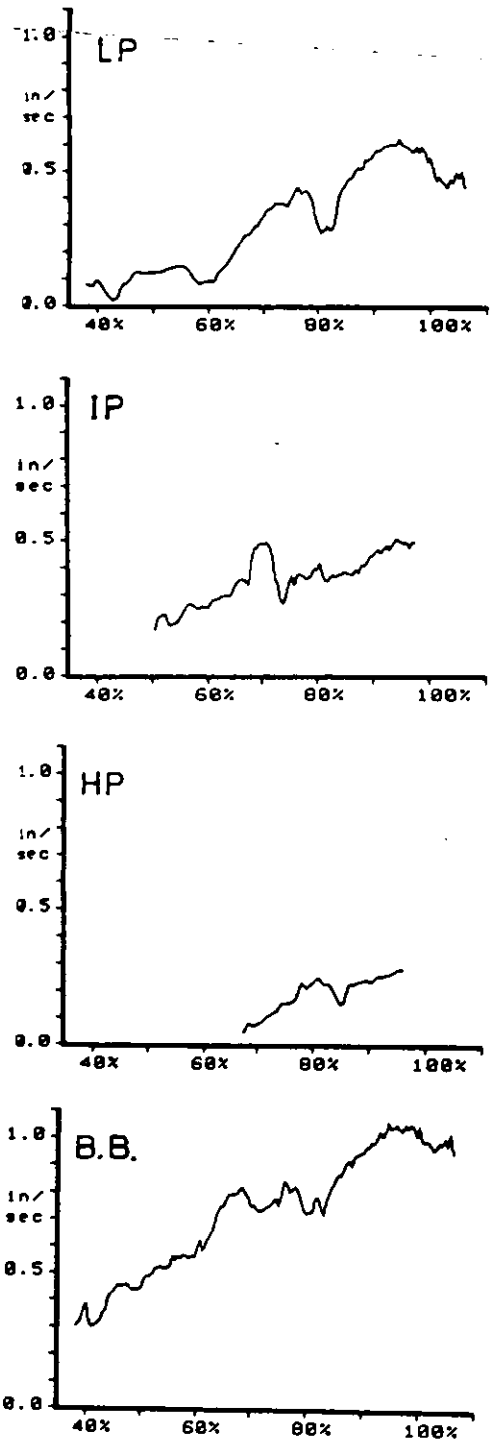
Calibration of this system is extremely straight forward, a routine has been included in the D.R. processor software (entered by issuing an interrupt from the host) which simply finds the largest component in each of the incoming spectra, converts it to an amplitude as described above, and stores it in the shared ram for use by the PDP-11. Calibration thus consists of injecting into each acquisition card a sinusoidal signal representing 1 inch/sec, and then recording the resulting calibration figure found in the shared ram. Note that no speed signals have to be injected and also that the actual frequency of the calibration signal is not critical. During normal vibration analysis the 12 data values read in from IDDAS simply need dividing by the recored calibration figure to form calibrated data. The results can then be

displayed in the same manner as any other engine parameter.

As can be seen, the host computer has to do very little work or computations to obtain tracked and calibrated vibration information. Results obtained by analysing the vibration signals from an RB211 aero engine, using the Rolls Rolls data reduction system (via tape recorded signals), and using the above IDDAS based system, are shown in figure 6.5. The two sets of graphs are almost identical except that the IDDAS plots are slightly smoother in shape. This is not due to lack of resolution or system response, but due to the improved amplitude estimation technique as described above. The data reduction system directly plots the amplitude of the tracked filter, resulting in the picket fence effect being clearly visible and somewhat misleading. Thus is can be concluded that not only does IDDAS produce vibration information in real time and in the test facility, but that the results are also of better quality.

At the time of writing this thesis, this vibration system had been installed into two production test facilities and was to be installed in a further two within the following few months.





Data reduction system IDDAS system

Figure 6.5 - Vibration signature of an RB211 engine

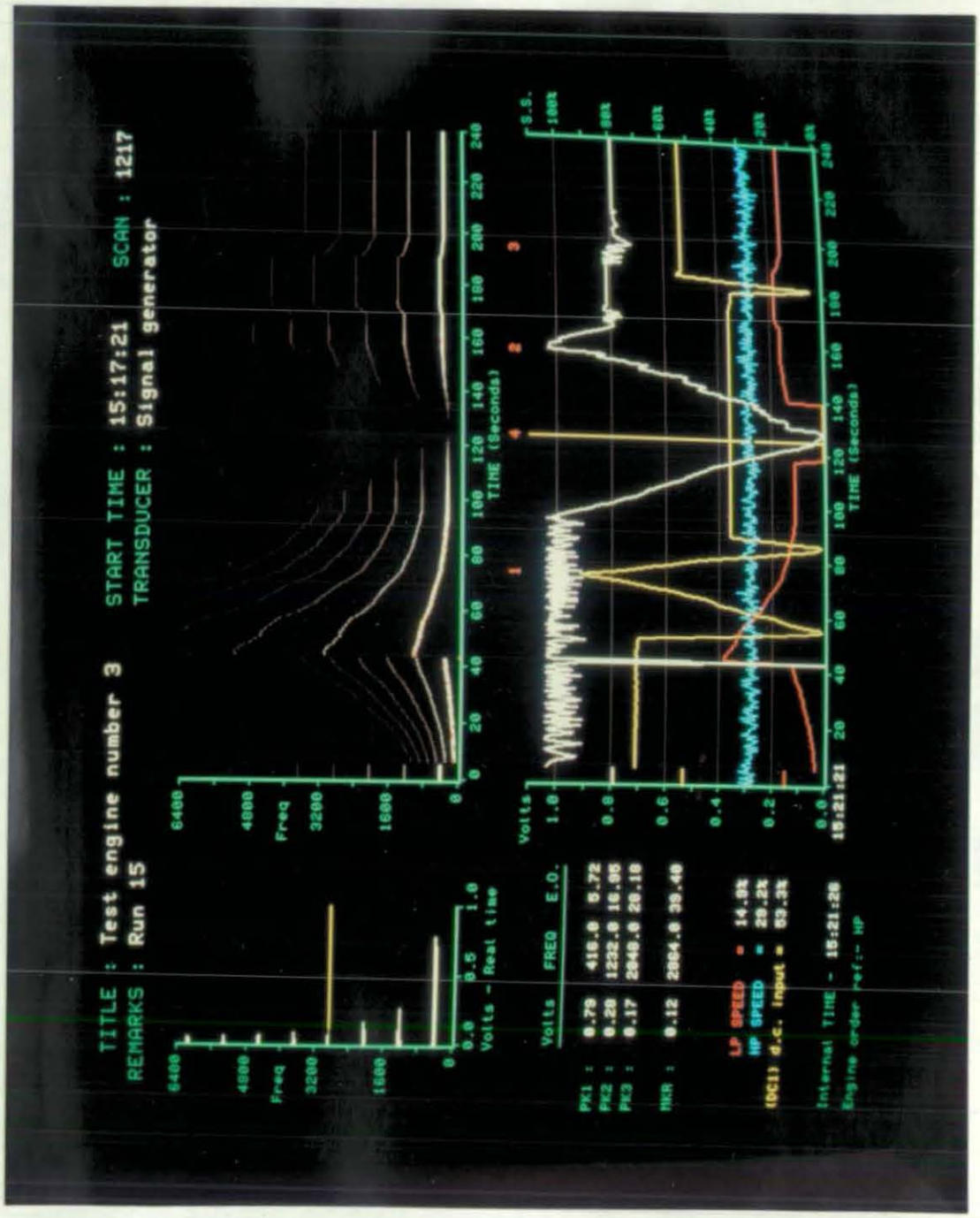


Photo 6.1 - Real time monitor display

CHAPTER 7

Spectrum estimation techniques

7.1 Introduction.

The fast Fourier transform is almost always used to perform spectrum estimation due to its consistent and predictable performance. As has already been shown, the amplitude and frequency estimation is not as precise as is often required, but its ease and speed of execution has ensured its popularity. There are however some occasions when the Fourier transform is unsuitable for spectrum analysis and can not be used. This usually occurs when the input data sequence is too short to provide adequate frequency resolution.

The above problem commonly occurs in geophysics and astrophysics where sample rates can be extremely low and the duration of an event relatively short (e.g. an earthquake). Consequently researchers in these fields have put much work into developing alternative spectrum analysis techniques which can cope with these conditions. Although the original work for many of the techniques dates back to the early part of this century, practical alternative spectrum analysis algorithms have only existed for about 20 years. As will be seen these algorithms are more complex than the FFT and are not straight forward to apply to input data.

A paper produced by Kay and Marple titled "Spectrum analysis - A modern perspective" [27] brings together most of the modern techniques and performs a simple test of their spectrum estimation capabilities. The four best techniques, as found in this paper, have been used by the author as a starting point for an assessment of modern techniques on real data, especially aero engine signals.

The derivation and programming of the four algorithms is briefly covered (there are many references made to more detailed discussions), followed by a fairly involved assessment of each one, together with modification and improvement of the two best techniques. It should be noted that the Fourier transform is also used as a standard in all the assessments.

7.2 Fourier Transform (via FFT).

The Fourier transform can be traced backed over 200 years and Fourier spectrum analysis to nearly 100 years. In 1898/9 Schuster published two papers [40,41] showing how he had attempted to fit "hidden periodices" to variations in sun-spot numbers, he also coined the term "periodogram". Another major step came in 1930 when Norbet Wiener published a paper [45] titled "Generalised Harmonic Analysis". From this developed the Fourier transform relationship between the autocorrelation function of a random process with the power spectral density. In 1959 a major publication from Blackman and Tukey [7] providing a practical implementation of Wieners autocorrelation approach. This implementation requires estimation of the autocorrelation lags, windowing, and a Fourier transform to obtain the psd.

The BT periodogram implementation soon became popular and many analytical computers became heavily loaded with spectrum analysis programs. In 1965 a major break through came with the introduction of the fast Fourier transform by Cooley and Tukey [17]. This new approach reduced the computational effort of the BT implementation proportionally from N**2 to N.log(N). The computational efficiencies obtained from the symmetries of the sine and cosine functions (as used in this new approach) can in fact be traced back to Danielson and Lanczos in 1942 [20], and to Runge and Konig in 1924 [39]. However for some reason, when machines became capable of processing large arrays of data, these earlier efficient

algorithms where overlooked. The fast Fourier transform, in amongst its various forms, e.g. decimation in frequency, decimation in time, prime factors and various radices, is currently the most widely used form of spectral analysis.

The fast Fourier transform is basically just a fast method of performing the discrete Fourier transform, and as such can be derived from the equation for the DFT. This is described in chapter 2. The properties of a spectrum produced by the FFT algorithm are hence the same as those for a DFT spectrum, the three most significant being

1) The resolution of a resulting psd is restricted at best to 1/sample window period.
2) The true signal psd is modified due to the convolution which occurs between the signal psd and the sinc function.
3) The spectrum filters are at fixed and harmonically related frequencies.

The second property, caused by assuming that data is zero outside of the Fourier sample block, is an undesirable feature which can be alleviated by using other shaped windows. Unfortunately these other windows result in a worse resolution, thus there is always a compromise between the effects of convolution and the resolution.

In the real world no signal is totally deterministic and will always be embedded in random noise, ie a stochastic process. Fourier transforms of these processes result in statistical inaccuracies, and thus spectral ensemble averaging must be performed to reduce and smooth out these inaccuracies. The need for this averaging is illustrated by Oppenheim and Schafer [34], and Otes and Enachson [35]. The performance of the DFT is thus still further compromised.

It can be shown that the DFT is equivalent to performing a

least squares fit with sinusoids of the following frequencies

0, 1.Fs/N, 2.Fs/N, ..., (N/2).Fs/N

Fs = sample frequency

This highlights the fact that the DFT assumes that the input function can be represented by a preassigned number of harmonically related sinusoids of fixed frequency. This of course is very rarely the case, causing the algorithm some difficulty in representing sinusoids of frequencies other than the preassigned ones, and also in representing wide-band components. Bearing in mind the above points it becomes obvious that the DFT is not the most ideal techinique for spectrum analysis, as a result much work has been put into researching alternative spectral analysis techniques which do not put such severe restrictions and constraints on the data.

7.3 Autoregressive Decomposition.

Transfer function modelling can be used to determine the psd of many deterministic and stochastic processes. The model is derived from known sampled data, representing an output sequence, and from an assumed function of known psd representing the driving sequence to the model. In this model the two sequences are related by a linear difference equation of the form

$$x(n) = \sum_{m=0}^{q} b(m).x(n-m) - \sum_{K=1}^{p} a(k).x(n-k)$$

this is generally termed as an Auto-Regressive Moving Averaging model. The system function H(z) is given by

$$H(z) = B(z)/A(z)$$

where A(z) is the AR term
and B(z) is the MA term

Having determined the system function, the psd of the sampled data can be found by using the well known relationship

$$P(z) = H(z).H^{*}(1/z^{*}).Pn(z) \qquad \text{where Pn is input noise}$$

Most research has gone into the Auto-Regressive model (where $b(m)=0,\ m>1;\ b(0)=1$) due to its computational efficiency over the full ARMA model. Algorithms developed for this model determine the poles of the AR filter from both the raw data and the estimated auto-correlation function. The most popular and well researched algorithms are Burgs maximum entropy method and the forward-backward least squares fit technique. These two will now be briefly examined.

7.3.1 Burg's Maximum Entropy AR Method.

The maximum entropy method is based upon the prediction of unknown autocorrelation functions using sequences of autocorrelation functions estimated from sampled data. No assumption is made of the data outside of the sample window. There are many ways of predicting the unknown autocorrelation functions, all of which produce different but valid results. Burg argued that the time series values produced by the predicted autocorrelation functions should have maximum randomness (ie maximum entropy). Another way of thinking about this approach, is that the estimated all pole filter, when applied to the sample data, should result in the flattest possible psd.

The autocorrelation function and AR parameters are linked by the Yule-Walker equations [12] shown below

$$Rxx(k) = \begin{cases} -\sum_{m=1}^{p} a(m).Rxx(-m) + \sigma^2 & k = 0 \\ -\sum_{m=1}^{p} a(m).Rxx(k-m) & k > 0 \end{cases}$$

Rxx = autocorrelation function, σ = gaussian noise.

From the solution of p+1 of these equations, a(1) to a(p) can be found directly. However there is a recursive technique known as the Levinson-Durbin algorithm [21, 28] which allows increasing orders to be evaluated starting from one AR parameter.

Burg found that the above solution for the AR parameters did not produce very good resolution and in 1967 he introduced a somewhat different MEM approach to the AR estimation which could be considered as a constrained least squares minimisation. This technique sets out to minimise the sum of the forward and backward prediction error energies

$$E(pn) = \sum_{n=p}^{N-1} |e(pn)|^2 + \sum_{n=p}^{N-1} |b(pn)|^2$$

$$e(pn) = \sum_{K=0}^{p} a(pk).x(n-k) \quad , \quad b(pn) = \sum_{K=0}^{p} a^*(pk).x(n-p+k)$$

$$a(p0) = 1 \qquad n = p,..,N-1$$

The constraint being that the AR parameters must satisfy the Levinson recursion for all orders from 1 to p. The desire for this constraint is to ensure a stable AR filter (i.e. all poles within the unit circle). This is shown in Figure 7.1

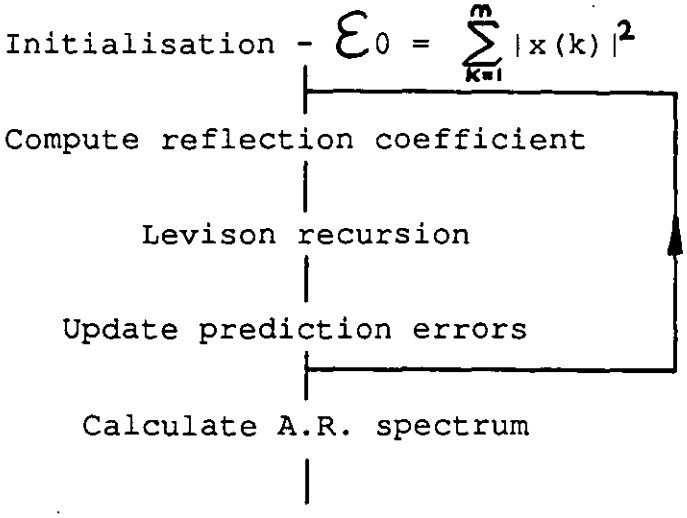


Figure 7.1 - Burgs A.R. spectrum estimation

7.3.2 Forward-backward least squares (FBLS) technique.

The Burg method of least squares minimisation has several problems which can be reduced in severity by removing the Levinson recursion constraint, a method first introduced independently by Ulrych and Clayton [44] and Nuttall [32] in 1976. It also then minimises the forward and backward prediction errors for all of the AR parameters a(k) (k=1,..,p). The forward prediction equation is defined by

$$X.a = y$$

$$\text{where } X = \begin{bmatrix} x(k) & x(k-1) & \dots & x(1) \\ x(k+1) & x(k) & \dots & x(2) \\ . & . & . & . \\ x(N-1) & x(N-2) & \dots & x(N-k) \end{bmatrix}$$

$$a = \begin{bmatrix} a(1) \\ a(2) \\ . \\ a(k) \end{bmatrix} \qquad y = \begin{bmatrix} x(k+1) \\ x(k+2) \\ . \\ x(N) \end{bmatrix}$$

and thus the forward prediction residual error is represented by

$$e(\text{forward}) = y - X.a$$

The least square solution is defined as any set of [a] which minimises the residual sum of squares S (where $S = e^T.e.$), this condition is achieved when its first partial derivative is equal to zero (ie $\partial S/\partial a = 0$). Applying this condition, it follows that S is minimised when

$$X^T.X.a = X^T.y$$

If $R(f) = X^T.X$ and $S(f) = X^T.y$ then the complete forward backward solution is represented by

$$[R(f) + R(b)].a = S(f) + S(b)$$

Thus the solution for a least squares fit is reduced to a straight forward simultaneous linear equation. This approach results in more computation than the MEM but as will be shown, the resulting spectra do not suffer from biasing.

A recursive approach to the FBLS method was published in 1980 by Barrodale and Erickson [4], which significantly reduces the computation effort, particularly when searching for the optimum length filter. Their approach also tackles the problems inherent in the LS method of parameters blowing up, and of inaccurate and sometimes negative residual parameters occurring.

7.4 Pisarenko harmonic decomposition.

The two auto-regressive methods described above can produce spectra with significantly better resolution than the FFT method because they do not assume anything about the data outside of the sample block. They also do not assume the psd to be constructed of fixed and harmonically related sinusoids. However, still better spectrum estimation can be made if it is based on some prior knowledge. The Pisarenko harmonic decomposition (PHD) technique [37, 38] assumes the input sequence to be composed of non-harmonically related sinusoids in white noise.

Now a deterministic process consisting of p real sinusoids of the form $\sin(2\pi ft)$ can be represented by a 2p order difference equation of the form

$$x(n) = -\sum_{m=1}^{2p} a(k).x(n-k) \qquad n=2p,..,N-1$$

where the a(k) parameters are coefficients of the polynomial

$$z^{2p} + a(1).z^{2p-1} + ... + a(p-1).z^{p+1} + a(p).z^{p} + a(p+1).z^{p-1} +$$

$$.... + a(2p-1).z + a(2p) = \prod_{i=1}^{p} (z-z_i)(z-z_i^*) = 0$$

This has unit modulus roots that occur in complex conjugate pairs of the form: $z(i) = \exp(j2\pi f(i).t)$, where the frequencies can be any where between -t/2 and t/2. It can also be shown that a(m) = a(2p-m) and with the addition of noise (w(n)) the difference order equation becomes

$$\sum_{K=0}^{2p} a(k).y(n-k) = \sum_{K=0}^{2p} a(k).w(n-k)$$

$$a(0) = 1$$
$$n=2p,..,N-1$$
$$y(n)=x(n)+w(n)$$

By suitable eigenvalue analysis of the above matrices it can be shown [24] that

$$Ryy.A = \sigma^2.A$$

When the dimension of Ryy is (2p+1) by (2p+1) or greater, σ^2 is equal to the smallest eigenvalue of Ryy, and A is its corresponding eigenvector. The a(i) coefficients can then be used to find the roots of the polynomial which in turn reveal the frequencies of the sinusoids. The power of each sinusoid can then be calculated by solving the simultaneous equation

$$F.P = r$$

$$\text{where } F = \begin{bmatrix} \cos(2\pi f_1 \Delta t) & \cdots & \cos(2\pi f_p \Delta t) \\ \vdots & \vdots & \vdots \\ \cos(2\pi f_1 p\Delta t) & \cdots & \cos(2\pi f_p p\Delta t) \end{bmatrix}$$

$$P = \begin{bmatrix} P(1) \\ \vdots \\ P(p) \end{bmatrix} \qquad r = \begin{bmatrix} Ryy(1) \\ \vdots \\ Ryy(p) \end{bmatrix}$$

A flow diagram is shown in figure 7.2. As with the AR methods the number of sinusoids and therefore the order of the polynomial must be determined for the best results. The PHD technique requires a lot of computational effort, and it was not until Hayes and Clement published an algorithm in 1986 [24] providing an iterative approach to calculating the eigenvalues and associated vectors, that the process order could be more efficiently calculated.

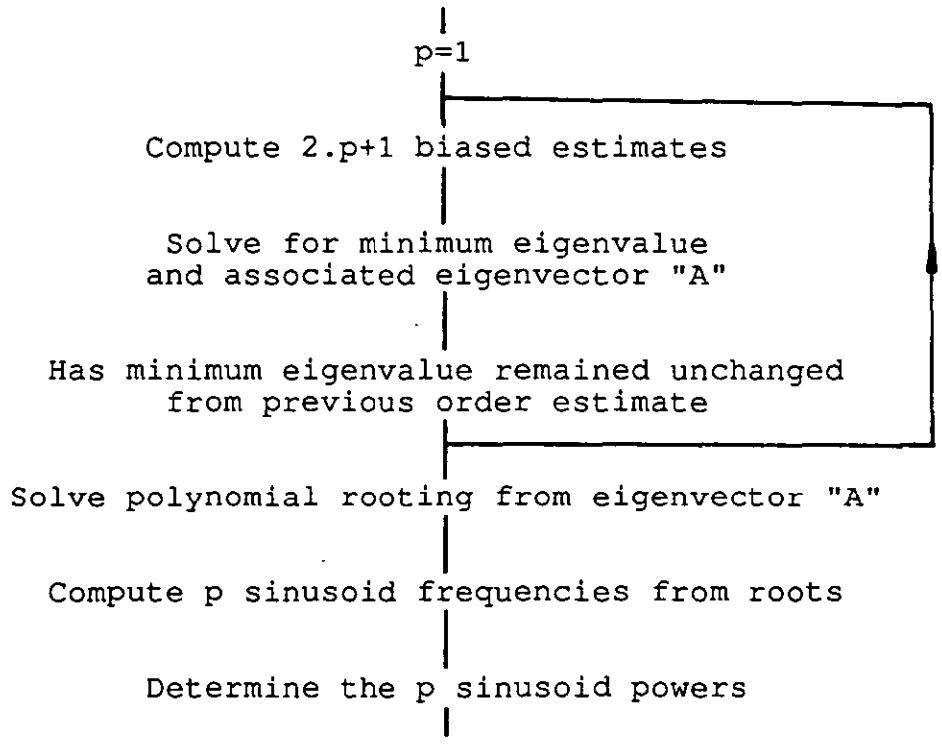


Figure 7.2 - Pisarenko spectral line decomposition

7.5 Prony spectral line estimation (PLSE) technique.

Prony's method of spectral estimation, like Pisarenko's assumes that the sampled process consists of sinusoids in additive noise. The original work by Prony was aimed at representing the expansion of gases by sums of exponentials, but this has undergone major changes over the years resulting in the modern extended Prony technique. This uses a least squares fit approach to estimate exponentials of the form

$$x(n) = \sum_{m=1}^{p} b(m).z(m)^n \qquad n = 0, 1, \ldots, N-1$$

where $b(m) = A(m).e^{j\theta}$ and $z(m) = e^{(\alpha_m + j2\pi f_m)\Delta t}$

however the solution to this equation is a difficult non-linear least squares problem [25].

To analyse a process of p real undamped sinusiods in noise a special variant of Prony's method can be used in which α_m is set to zero, resulting in the roots being complex conjugate pairs of unit modulus. Thus we must solve for the roots of the polynomial

$$\Psi(z) = \prod_{i=1}^{p} (z-z_i)(z-z_i^*) = \sum_{k=0}^{2p} a(k).z^{2p-k} = 0$$

where $a(0) = 1$ and $a(k)$ are real

Due to the roots being of unit modulus and occurring in complex conjugate pairs, it can be shown that $a(k) = a(2p-k)$ $(k=0, 1, .., p)$ and hence that $a(p) = 1$ [27]. The solution is implemented by constraining the polynomial coefficients to be symmetrical about the centre element. Thus the linear prediction error can be written as

$$\varepsilon_n = \sum_{k=0}^{p} a(k).[x(n+k) + x(n-k)] \qquad n=p,..,N-p-1$$

From this point onwards a standard least squares fit algorithm can be applied to the problem, with the data matrix taking the form as shown below, note that X is Toeplitz and Hankel in structure rather than just Toeplitz as in the AR case

$$\begin{bmatrix} \varepsilon_{(p+1)} \\ . \\ \varepsilon_{(N-p)} \end{bmatrix} = X.A$$

$$\text{where } A = \begin{bmatrix} 1/2 \\ a(1) \\ a(p-1) \\ . \\ a(p) \end{bmatrix} \qquad X = T + H$$

$$T = \begin{bmatrix} x(p+1) & \dots & x(1) \\ . & . & . \\ . & . & . \\ x(n-2p) & & x(p+1) \\ . & . & . \\ . & . & . \\ x(p+1) & \dots & x(N-2p) \end{bmatrix} \qquad H = \begin{bmatrix} x(p+1) & \dots & x(2p+1) \\ . & . & . \\ . & . & . \\ x(2p+1) & & x(N-p) \\ . & . & . \\ . & . & . \\ x(N-p) & \dots & x(N) \end{bmatrix}$$

Having determined p coefficients, a polynomial rooting can be performed to determine the complex conjugate root pairs which will then reveal the frequency components of the process. The power and phase information (ie the b(m) term) is then determined from the following equation

$$\phi.B = X$$

$$\phi = \begin{bmatrix} 1 & 1 & .. & 1 \\ z(1) & z(2) & .. & z(p) \\ . & . & .. & . \\ z(1)^{N-1} & z(2)^{N-1} & .. & z(p)^{N-1} \end{bmatrix} \qquad B = [\ b(1) \ \dots \ b(p)\ ]^T$$

$$X = [\ x(0) \ \dots \ x(N-1)\ ]^T$$

The Prony spectral line estimation thus consists of the solution of two simultaneous linear equations (least square fits) and a polynomial rooting. As with the other three methods the order of the process must be determined to obtain optimum results. A flow diagram for this technique is shown in Figure 7.3. This technique requires a great deal of computation especially as Barrodale and Ericksons recursive techniques (as used in the forward-backward LS method) cannot be used here due to the form of the AR filter matrix.

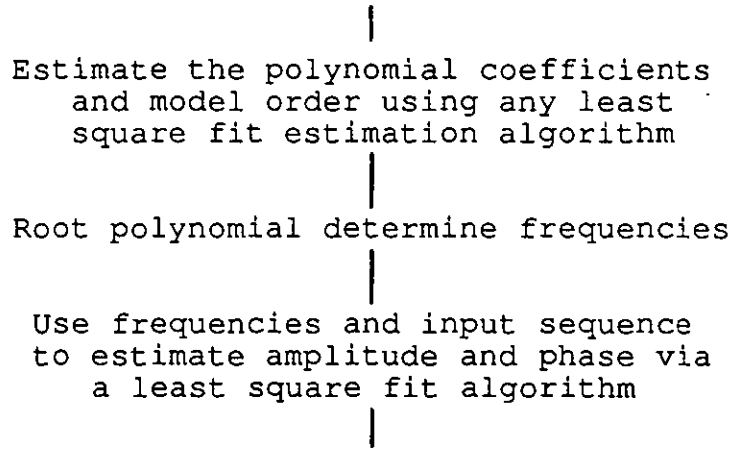


Figure 7.3 - Prony spectral line estimation

CHAPTER 8

Assessment of spectral estimation techniques

8.1 Form of assessment.

In this chapter an assessment is made of the performance of four modern spectrum estimation techniques as described in chapter 7. This assessment is made by comparing the spectra obtained applying each of the techniques to two different data sequences. These data sequences are as follows

1) The data sequence as used by Kay and Marple [27] is used here to enable a cross check of results and provide confidence in the author's programs.

2) A data sequence consisting of pure sinusoids is used to assess the performance of the four techniques with more realistic aero engine type signals.

8.2 Kay and Marple data.

The true power spectral density of the 64 point data sequence used by Kay and Marple, together with the psd's obtained from the DFT and the four the spectrum estimation techniques are shown in graphs 8.1.1 to 8.1.6. This input data sequence does not represent the type of signals expected from engine transducers but provides a useful check of the validity of the author's fortran programs. Note that Kay and Marple use the DFT without windowing, which is not really representative of the way in which the DFT is usually used. They also do not show for the other techniques, to what extent the psd estimates change when the process order is chosen either too high or too low.

8.2.1 Fourier transform.

Kay and Marple have used double zero padding in their DFT analysis (graph 8.1.2), this has the effect of interpolating between filters but does not provide any extra resolution or information. By windowing the input data however, a considerable difference can be made to the clarity and to the amount of information that can be extracted from the psd. Graphs 8.2.1 and 8.2.2 show the psd's obtained when the rectangular window and a Kaiser-Bessel window (beta=6) are used. As can be seen, a much better psd is obtained by using a window, although of course the two close sinusoids (0.20 and 0.21 Fs) are still merged together. Note that the resolution of a 64 point DFT is always worse than 0.016 Fs, so the two close sinusoids can never be singularly identified. Note also that the reduced side lobes of the windowed DFT results in a more accurate psd due to reduced interference between neighbouring filters.

The 0.2 Fs component of the windowed DFT is very close to its correct amplitude of 1, however the amplitude of the 0.1 Fs is approximately 1 dB down on its true value (-13% error). This is a result of the picket fence effect which is inherent in DFT analysis. Note that the Y-axis power scale is referenced to peak amplitude and not rms amplitude (ie a peak amplitude of 1 is equal to 0 dB), and that the window weighting constants have been arranged to give an overall window gain of unity.

The test signal clearly shows the picket fence effect and the lack of resolution which are inevitable in DFT spectrum analysis.

8.2.2 Burg's A.R. method.

A psd plot produced by Burg's method is shown in graph 8.3, this was obtained by estimating 16 filter coefficients and then calculating the power around the unit circle in 0.005-Fs steps from these coefficients. The program used to perform this was taken from a paper by Ulrych and Bishop [43]. Comparing graph 8.3 with graph 8.1.3, there is very little difference to be seen, indicating correct programming of the algorithm. The only small difference between the two, being that the amplitude of the 0.1 Fs component in the authors psd is more accurate than that in Kay and Marple's. Although it is difficult to say exactly why this is, it is most probably due to the different computers and floating point software that have been used. This would almost certainly cause a slight difference between the AR coefficients used to calculate the psd, resulting in components occurring with very similar frequencies but slightly different powers.

The effect of using the Levinson recursion constraint on the AR coefficients results in frequency biasing of the estimated components. This is high-lighted by the estimated frequencies of the 0.20 and 0.21 Fs components which are 0.2025 and 0.215 Fs respectively. Note that the psd has been normalised to the largest component, thus absolute powers are not indicated in the results, the actual power of the largest component before normalisation was 22.3 dB. The peaky nature of the AR filter can be seen in the broad band area of the spectrum, these peaks could easily be interpreted as real sinusoidal components.

Burg's method produces a better result than the DFT when considering resolution of close sinusoidal components, but its power estimation is poor and biasing has occured in its frequency estimates. It must also be remembered that 16 filter coefficients have been chosen by trial and error to produce the best psd. Choosing the wrong number of coefficients produces even worse estimates as is demonstrated

with the next data sequence.

8.2.3 Forward-backward least squares technique.

A psd produced by the forward-backward least squares technique is shown in graph 8.4, again the algorithm was used to estimate 16 AR coefficients and the psd calculated around the unit circle in 0.005 Fs steps. The program used to perform this was taken from a paper by Barrodale and Erickson [4]. Comparing graph 8.4 with graph 8.1.4, the two psd's are practically identical, again the only difference being that the amplitude of the 0.1 Fs component is more accurate in the authors psd than in Kay and Marple's. As with Burg's method this is assumed to be due to slight variations in the AR coefficients, caused by using different computers and software.

In this case the three frequencies of the sinusoidal components have been estimated very accurately without any biasing and the relative power levels are a lot closer to the true values. Note however that the psd has again been normalised to the largest component, the actual estimated value of this component being 21.6 dB. This method produces a peakier response than with Burg's method, and again generates potentially confusing peaks in the broad band area.

As with Burg's method this technique is much better at resolving close sinusoids than the DFT is, but its power estimation is again poor. The performance of this technique is also highly dependent on the correct filter order being chosen, as will also be shown later.

8.2.4 Pisarenko harmonic decomposition.

A psd produced by the PHD algorithm is shown in graph 8.5. The program for this was derived from a non-working program

started by Khaldoon Ghaidan for his speech analysis doctorate [26], the Pisarenko program published in a paper by Hayes and Clement [24] was not available to either Khaldoon or the author when this work was started. Most of the matrix manipulation and eigenvalue analysis has been achieved by using NAG routines supplied as part of the Fortran software library on the Loughborough main frame computer. The program uses no recursive techniques and is computationally intense. The algorithm was used to estimate eight complex conjugate root pairs.

Comparing graph 8.5 with graph 8.1.5, there is one significant difference between the two psd's, this is the amplitude of the component at 0.16 Fs. It would seem unlikely that this is due to a programming error as all of the other components exactly match. As in the earlier cases it is much more likely to be due to differences in floating point and library routines, especially as the power calculations are dependent upon a very small eigenvalue which has been found via several NAG matrix routines. It should be remembered that this component should not be in the spectrum at all and is likely to be highly dependent on small variations.

The frequency and power estimates obtained by this algorithm are very poor, the psd shown in graph 8.5 is again normalised to the largest component, the actual estimated power of this component was -2.6 dB. The frequency estimation is significantly biased due to the use of biased auto correlation functions, the relative power estimates are not at all representative of the true powers, and the component at 0.16 F(s) is spurious and totally misleading. Although this algorithm is better than the DFT in that it manages to resolve the two close sinusoids, it becomes practically useless when it injects components such as the one at 0.16 Fs.

8.2.5 Prony spectral line estimation technique.

A psd produced by the Prony technique is shown in graph 8.6, the algorithm was used to estimate eight complex conjugate root pairs of unit modulus. The program used to perform this is entirely original to the author as no publications of programs already existing could be found, although it is assumed that either Hilderbrand or Kay and Marple have published something along these lines. Note that the authors program makes much use of the NAG library routines to perform simultaneous equations and complex polynomial rootings. A Fortran listing of this program is given in appendix F for reference.

Comparing graph 8.6 and graph 8.1.6, no difference at all can be seen between the two psd's, indicating that the authors program is correct. The Prony method estimates the frequency and power of the three sinusoids extremely well as shown in the table below.

Frequency [fraction of Fs]	Amplitude [peak]	Phase [degrees]
0.100039	0.1033464	125.28
0.200267	1.0255654	156.47
0.209599	1.0396414	170.60
0.270987	0.0564410	36.35
0.309460	0.2108561	65.79
0.358735	0.1186532	174.40
0.402381	0.1971185	95.59
0.450810	0.0278794	32.93

The frequency is particularly well estimated with only a fraction of a percent error. The two close sinusoids are also well resolved with seemingly no interference on each other. As with the Pisarenko technique, which also assumes a sinusoidal model, Prony's technique does not represent the broad-band components particularly well, although it does indicate the presence of power in this region. Again, this

algorithm is computationally intense and no recursive techniques have been used, however compared to the Pisarenko harmonic decomposition it produces significantly better results for a similar amount of computation.

8.2.6 Summary of results using Kay and Marple test data.

Comparing the above results obtained from the four spectrum estimation techniques, the following points can be noted

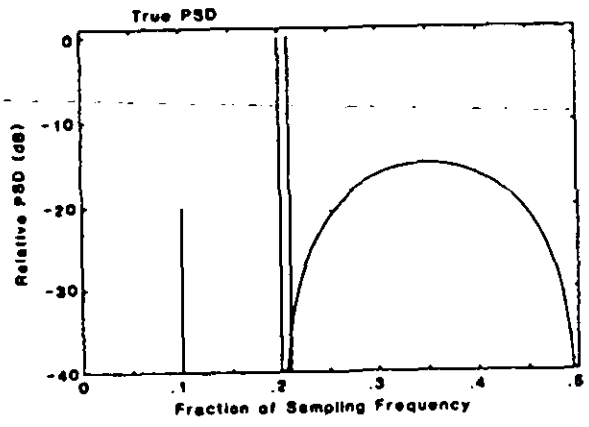
The Burg and Pisarenko techniques both suffer from frequency biasing due to the constraints imposed by their respective algorithms.

Power estimates by Burg's method and the forward backward least square technique, which both rely on accurate AR coefficients and residual power, are very poor.

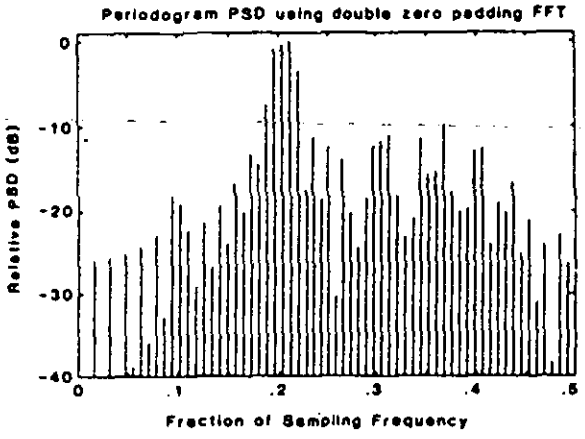
The least square and Prony techniques produce very accurate frequency estimates.

Least square power estimation, as in Prony's technique, produces accurate results.

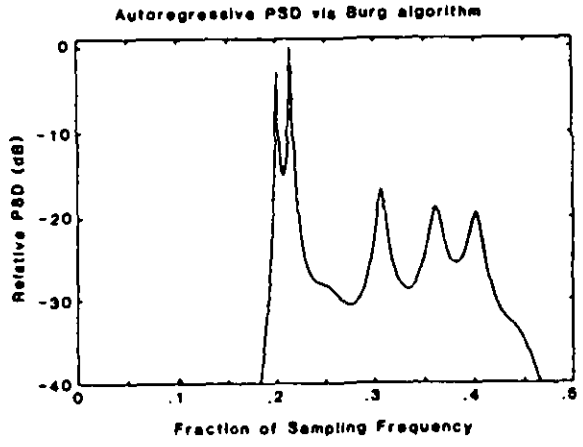
None of the techniques represent the broad band frequency component very well.



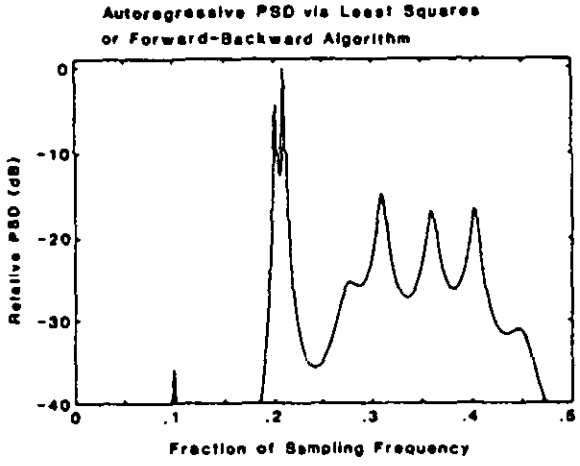
8.1.1



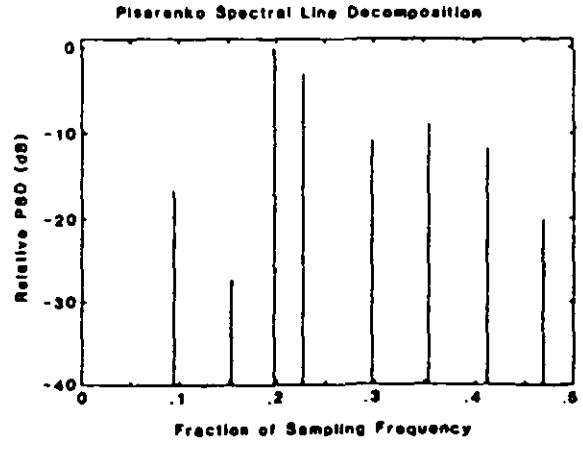
8.1.2



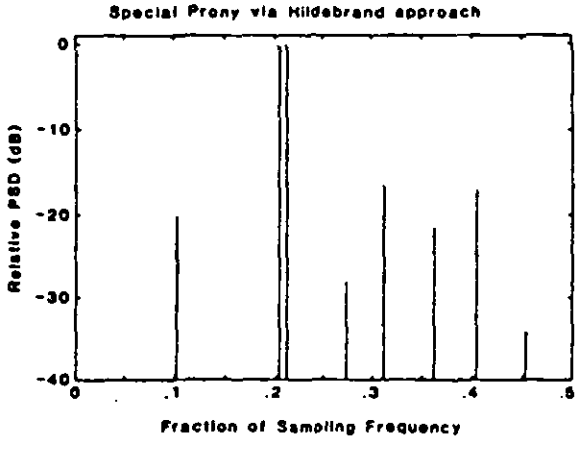
8.1.3



8.1.4

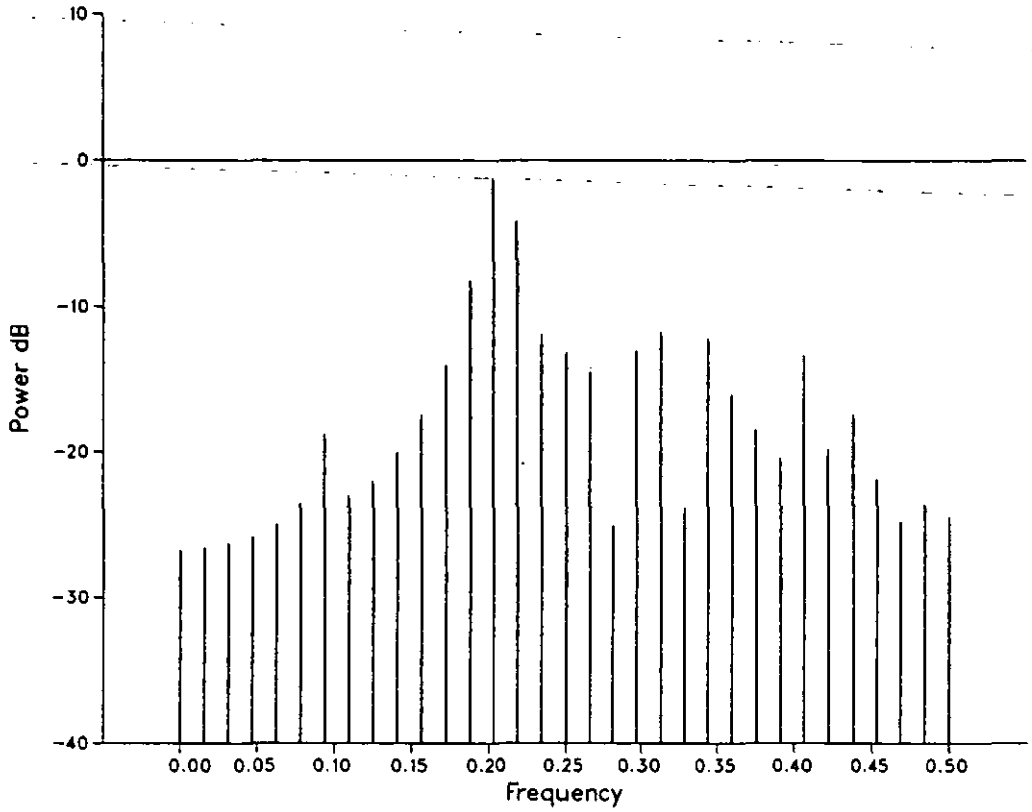


8.1.5

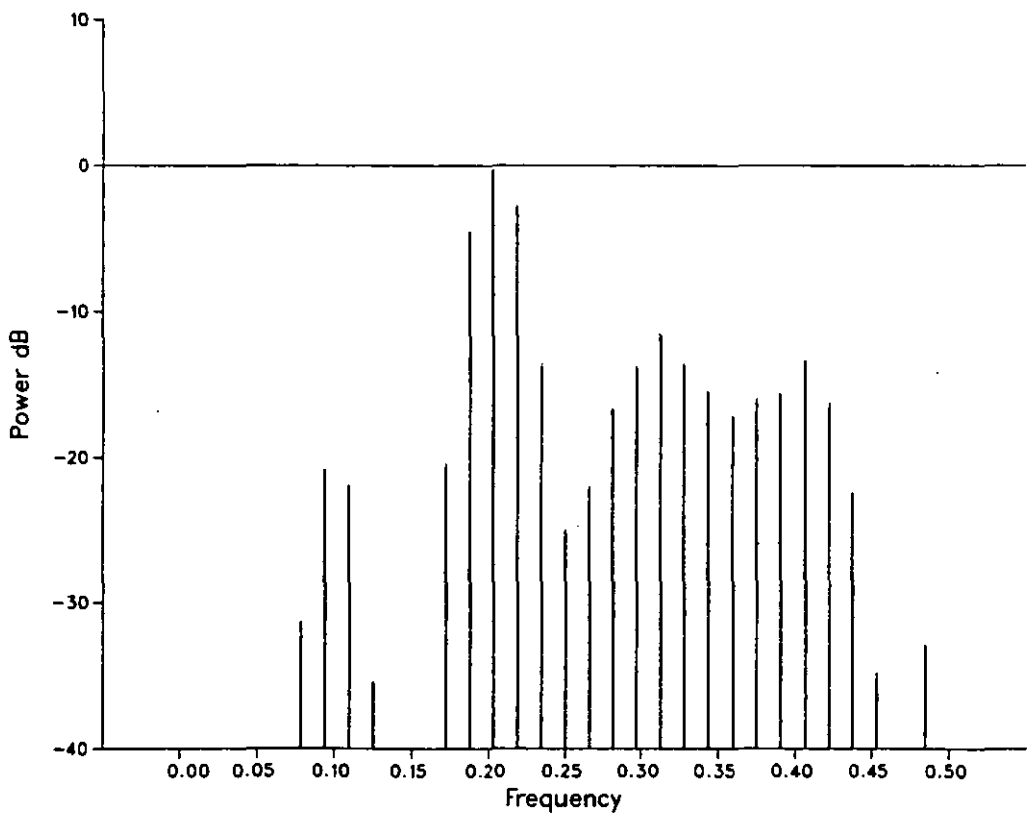


8.1.6

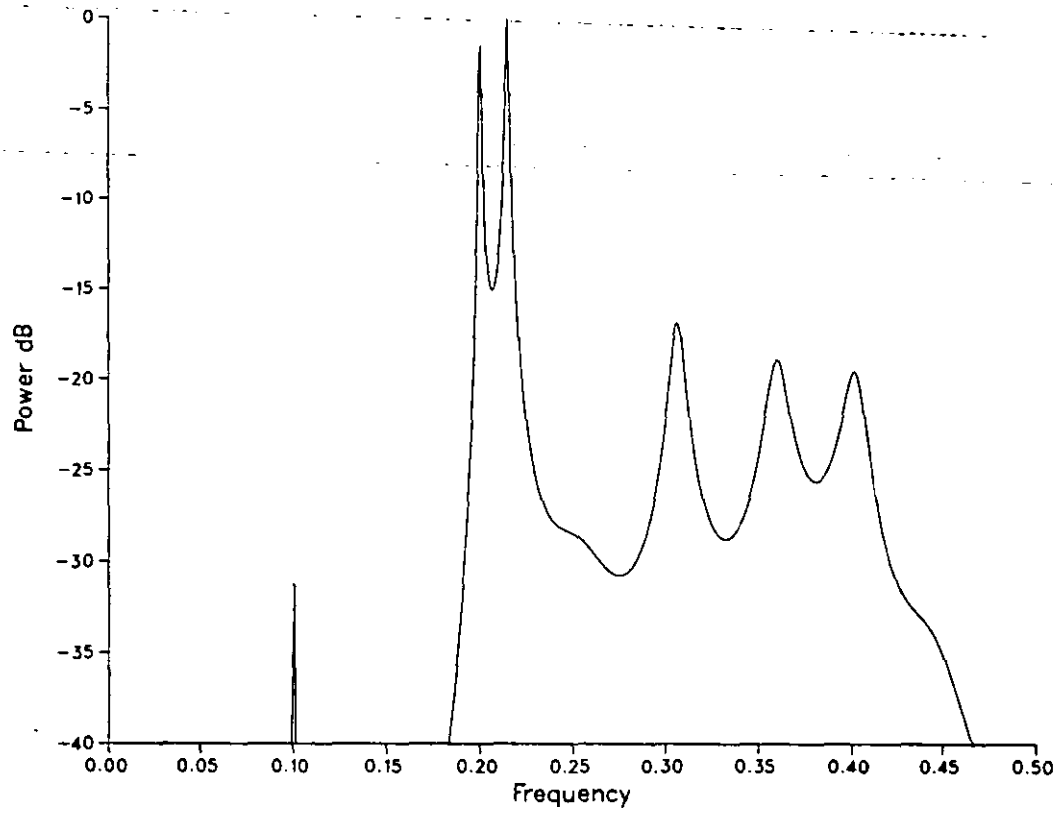
Graph 8.1- Kay and Marple psd's



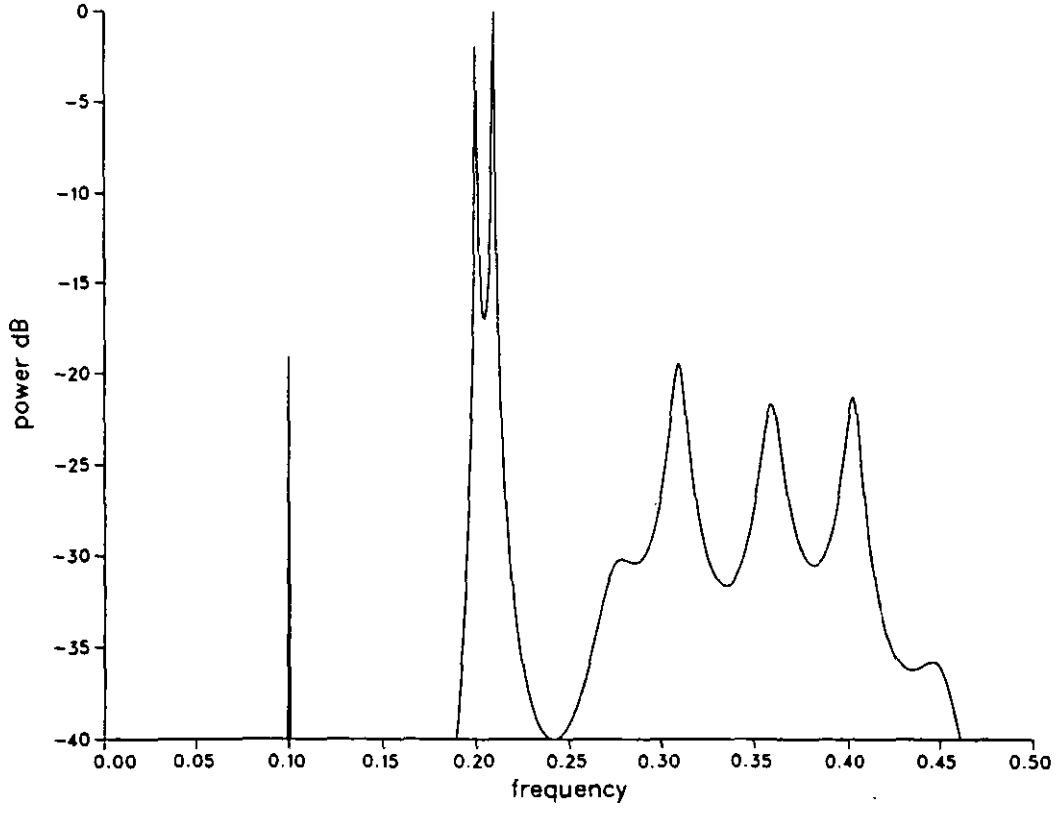
Graph 8.2.1 - DFT, rectangular window; K&M data



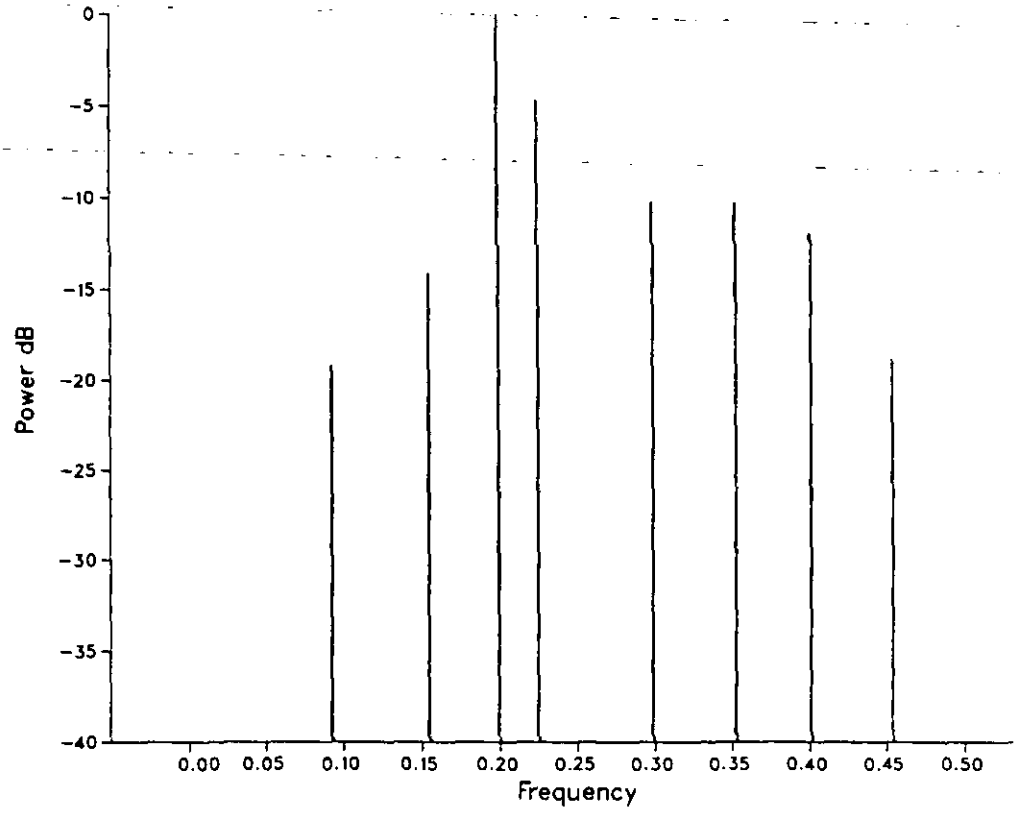
Graph 8.2.2 - DFT, Kaiser-Bessel window; K&M data



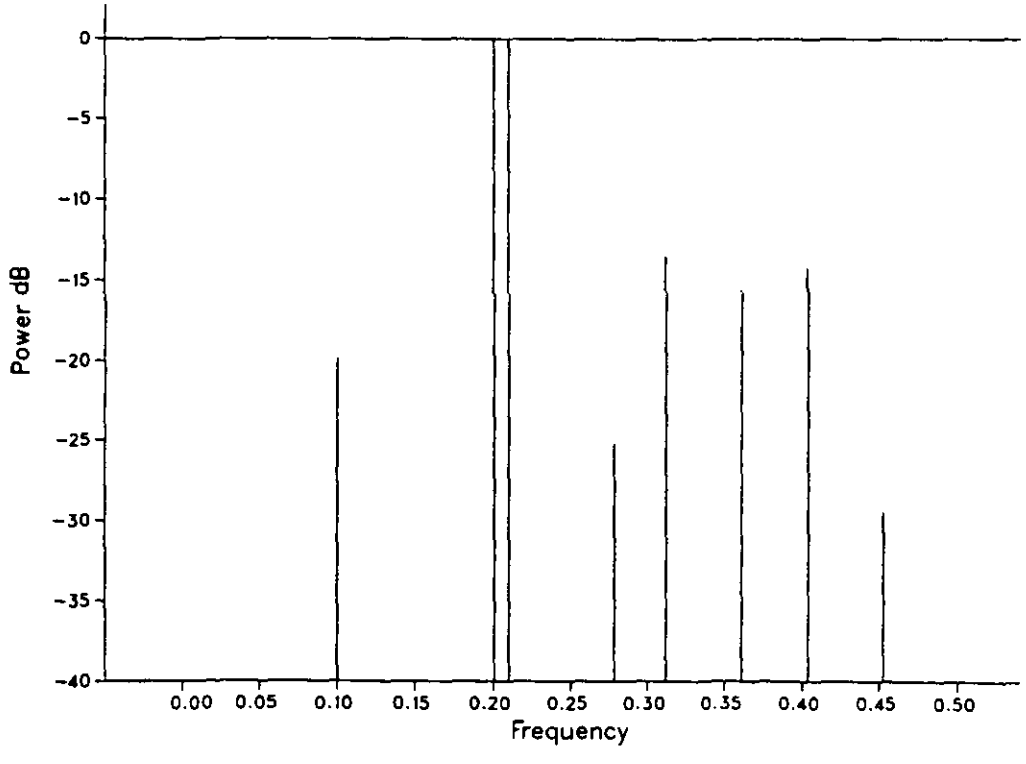
Graph 8.3 - Burg's method, 16 coefficients; K&M data



Graph 8.4 - FBLS technique, 16 coefficients; K&M data



Graph 8.5 - Pisarenko, eight root pairs; K&M data



Graph 8.6 - Prony, eight root pairs; K&M data

8.3 Sinusoidal test signal.

An assessment is now made on the performance of the four spectrum analysis techniques when applied to a signal consisting entirely of sinusoidal components. The 64 point data sequence consisted of six sinusoids, as follows

Frequency [fraction of Fs]	Amplitude [peak]	(dB)	Phase [degrees]
0.05	0.1	(-20)	0
0.16	10.0	(20)	60
0.25	1.0	(0)	45
0.26	10.0	(20)	120
0.35	0.1	(-20)	180
0.425	1.0	(0)	90

Two points to notice about these components

1) There are two close sinusoids of different amplitudes (Kay and Marple's were of the same amplitude),

2) There is a 40 dB difference between largest and smallest sinusoids (Kay and Marple's only differed by 20 dB).

8.3.1 Fourier Transform.

This technique was again applied using the Kaiser Bessel window, the resulting psd is shown in graph 8.7. This psd reveals no real surprises, the close sinusoids are not resolved, energy is spread into side lobes, and where components are not exactly on filters the power is less than the true value.

8.3.2 Burg's AR method.

To demonstrate the need for the correct choice of the number of AR coefficients required to achieve an optimum psd estimate, Burg's method has been tested with four different numbers of AR coefficients, these being 12, 14, 16 and 18. The resulting psd's are shown in graphs 8.8.1 to 8.8.4, note that the power has been normalised to the largest component. As can be seen, with only 12 coefficients only four of the six components have been extracted, with 14 and 16 coefficients six have been extracted, and with 18 coefficients eight components have been extracted. This is a variation from two sinusoids too few, to two sinusoids too many over a change of just six coefficients.

The optimum psd must be obtained somewhere in the relatively small range of 13 to 17 coefficients. The frequencies of the six components extracted in the psd's for 14 and 16 coefficients do not seem to have suffered from biasing and are reasonably accurate. The difference in power between the largest and smallest components is approximately correct at 40 dB, however the actual power estimate of each component before normalisation is very poor. The power of the largest component (0.16 Fs) being 36.9 dB for 14 coefficients, and 41.8 dB for 16 coefficients. There is also clear evidence of line splitting at 0.25 Fs when 18 coefficients are used.

8.3.3 Forward backward least squares technique.

The Least squares technique has been tested in a similar fashion to Burg's method in that psd's have been estimated by using four different numbers of coefficients, in this case 11, 12, 14, and 18. The resulting psd's are shown in graphs 8.9.1 to 8.9.4, and have again been normalised to the largest component.

As can be seen there is a very sharp change in the spectrum response between 11 coefficients and 12 coefficients , where the "Q" of the peaks become very much higher and all six sinusoids are extracted. It is significant that the break point is at 12 coefficients as this is the number of coefficients required to represent the 6 complex conjugate root pairs which match with the 6 sinusoids.

The frequency estimation of this technique is very good, producing an accurate and high "Q" response, and there is also no sign of line splitting. Power estimation however is not so good, as shown by the estimate of the largest component in the 14 coefficient psd, of 44 dB at 0.26 Fs before normalisation.

8.3.4 Pisarenko harmonic decomposition.

The PHD technique has been used to extract six complex conjugate root pairs from the test data, this being the correct number to represent six sinusoids, the resulting psd is shown in graph 8.10. Psd's for other numbers of root pairs are not shown as these result in totally inaccurate estimates. Note that, as with the two previous techniques, the power estimate is referenced to rms amplitude, i.e. a sinusoid of amplitude 10 should be equivalent to 17 dB on the graphs.

As can be seen from the psd, the estimated power of the sinusoids at 0.16, 0.26 and 0.425 Fs are quite accurate, their frequency estimates are also reasonable although they do suffer from some slight biasing. The frequency estimate of the 0.05 Fs component is also not too bad but the power estimate is completely wrong, as is the frequency and power estimates of the two remaining components, thus making this psd practically useless.

8.3.5 Prony spectral line estimation.

As with the PHD technique, only one psd has been produced, again this being for six complex conjugate root pairs, the resulting psd is shown in Figure 8.11. Note that the power estimate, as with the Fourier transform, is referenced to peak rather than rms amplitude. The results are extremely good, both frequency and magnitude have been estimated very accurately as shown below

Frequency [fraction of Fs]	Magnitude [peak]	Phase [degrees]
0.05000000	0.9999926	0.000001
0.16000000	10.000000	90.000165
0.24999999	0.9999866	45.000087
0.26000000	10.000010	119.999993
0.34999999	0.1000010	180.000002
0.42500000	1.0000008	90.000160

This result is not unduly surprising as the model this technique is based on, is identical to the process presented to it in this test.

8.4 Summary of results.

The two tests performed on the four spectrum estimation techniques have produced consistent results which can be summarised as follows

8.4.1 Burg's AR method.

Good points :-

Employs a recursive approach resulting in relatively efficient computation.

High frequency resolution even with small sample blocks.

No assumption is made of the data outside of the sample block and thus there are no leakage effects.
Guaranteed stable linear prediction filter.

Bad points :-

Constraint on AR coefficients causes frequency biasing.
Overdetermination causes line splitting.
Power estimates inaccurate and dependent on filter order.
Must determine order of AR filter.

Burg's AR method works well in that it provides good resolution, a stable output and is computationally efficient. However, for analytical work where accurate results are required this technique does not provide a good solution.

8.4.2 Forward backward least squares technique.

Good points :-

Employs a recursive approach.
Higher frequency resolution than Burg's method.
No line splitting.
No assumption is made of the data outside of the sample block and thus there are no leakage effects.
No frequency biasing.
In general a stable AR filter (although not guaranteed).

Bad points :-

Power estimates inaccurate.
Must determine order of AR filter.

This technique provides good frequency resolution and estimation, the filter response being particularly sharp once the minimum set of coefficients has been calculated. Also, in the tests performed here, no conditions have been found where the algorithm is unstable, although as stated above this is not a guaranteed condition. The only problem with this technique is poor power estimation. As with Burg's method it

calculates the psd from the AR filter residual power. It has become quite obvious that this is not a good way of doing it. Lacoss [29] has suggested that the peaks are proportional to the square of power and that the area under the peaks is proportional to power. This hypothesis was tried on the spectra shown in graphs 8.8 and 8.9, the results showed no real improvement in power estimates. It can be seen that there is a more fundamental problem in the fact that the peaks change in shape and amplitude in respect to each other as different numbers of AR coefficients are used. It is felt that estimation of power via the filter residue, as is the case for all the above estimates, is not the best approach. This problem is discussed further in chapter 9.

8.4.4 Pisarenko harmonic decomposition.

Good points :-

- High frequency resolution.
- No assumption is made of the data outside of the sample block and thus there are no leakage effects.
- Reasonably accurate power estimate of some components.

Bad points :-

- Significant frequency estimation biasing due to auto-correlation constraints.
- Power estimates inaccurate.
- Produces spurious components.
- Must determine order of process.
- High degree of computation.

This technique has not performed well in either of the two tests, showing significant inaccuracies in both frequency and power estimates. It would seem that this is also the opinion of other researchers, as this technique has been extended to what is called the "Music algorithm" [36] in which the results from several eigenvectors are averaged together to reduce the frequency biasing and the effect of spurious

components.

8.4.4 Prony spectral line estimation technique.

Good points :-

High frequency resolution.
No assumption is made of the data outside of the sample block and thus there is no leakage effects.
Accurate frequency estimation of sinusoids.
Accurate amplitude estimation.
Accurate phase estimation.

Bad points :-

High degree of computation.
Must determine order of process.
Not guaranteed stable.

This technique has produced by far the best results, especially for purely sinusoidal signals. As pointed out before, this is not totally surprising as this technique performs a sinusoidal least square fit for frequency estimation and a least squares fit for amplitude estimation, assuming in both cases that the signal consists entirely of undamped sinusoids. There are however two slight problems;

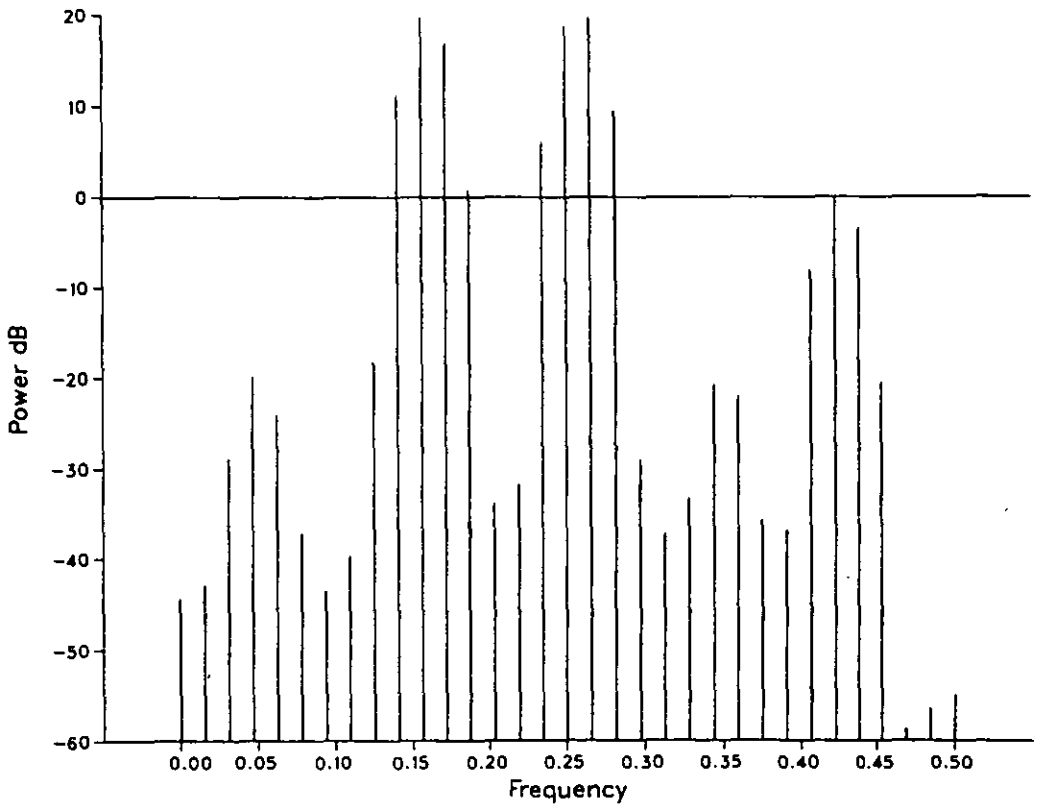
The algorithm is not 100% stable. It has been found that if the wrong number of root pairs is used on any particular signal, then frequencies of 0.0 or 0.5 Fs occasionally occur. This is not too detrimental as the resulting amplitude estimation for these components is always very small. However occasionally a more serious and fatal error occurs, when the Fortran NAG routines exit early due to overflow errors during matrix manipulations.

All of the other spectrum analysis techniques have methods for estimating when the correct order has been

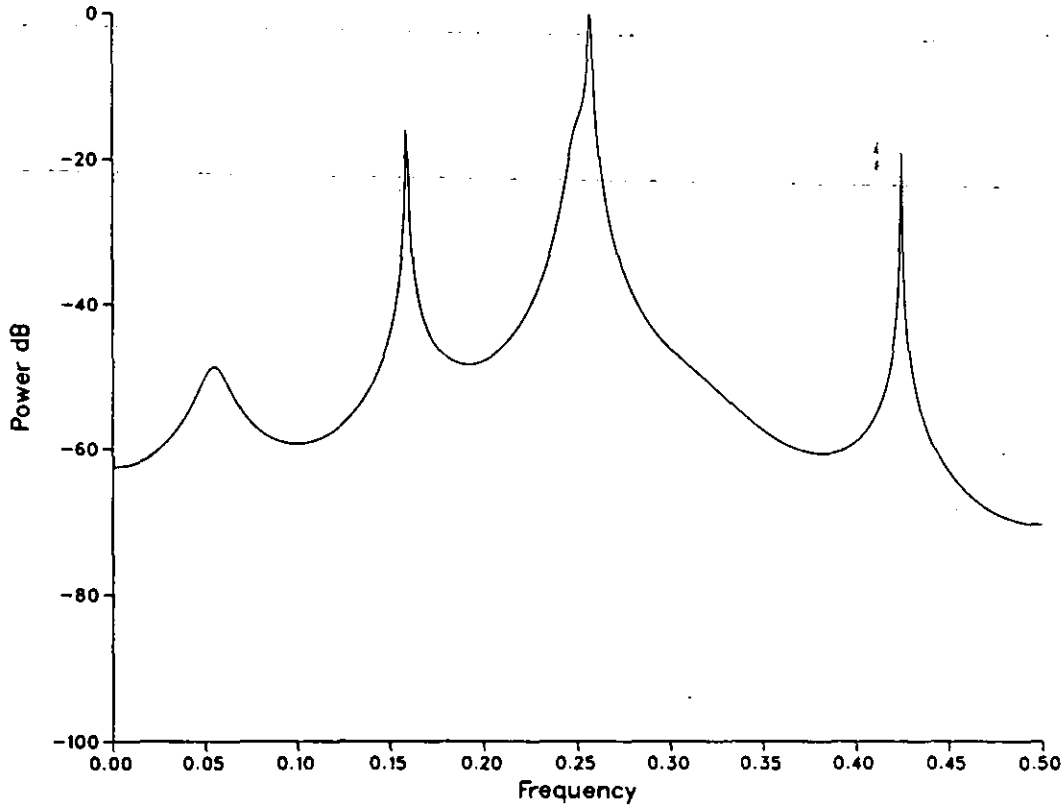
reached, and thus when further recursive computation can stop (eg Akiake's methods for the Burg and LS techniques). No reference has been found for a method of doing this on Prony's technique.

8.5 Further assessment.

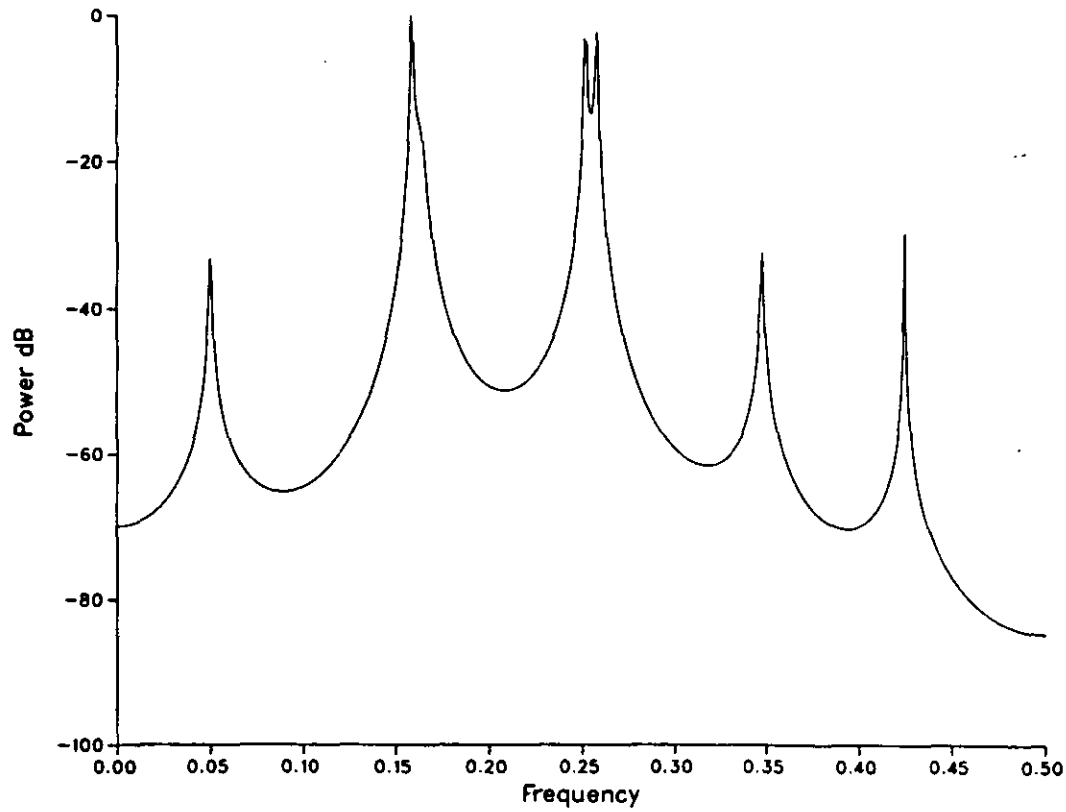
The real test for any spectrum analysis technique comes when it is applied to a signal embedded in noise, such as aero engine transducer signals. The technique must provide accurate results and demonstrate a reasonable tolerance to noise. Based upon the points made in the above summary, only the forward backward least squares technique and Prony's spectral line estimation technique will be assessed further with this type of signal. In chapter 9, not only will resilience to noise be assessed, but also whether reliable order estimation methods can be developed, as will be required for a final solution.



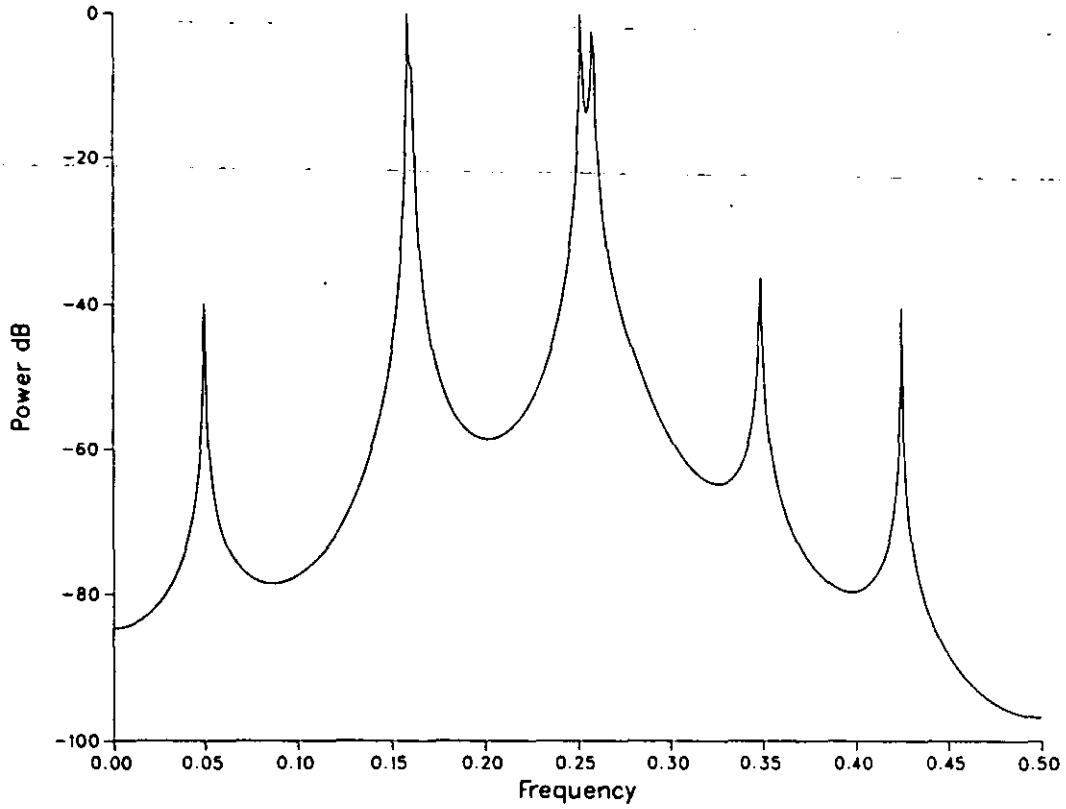
Graph 8.7 - Fourier transform, Kaiser-Bessel window; Six sinusoids



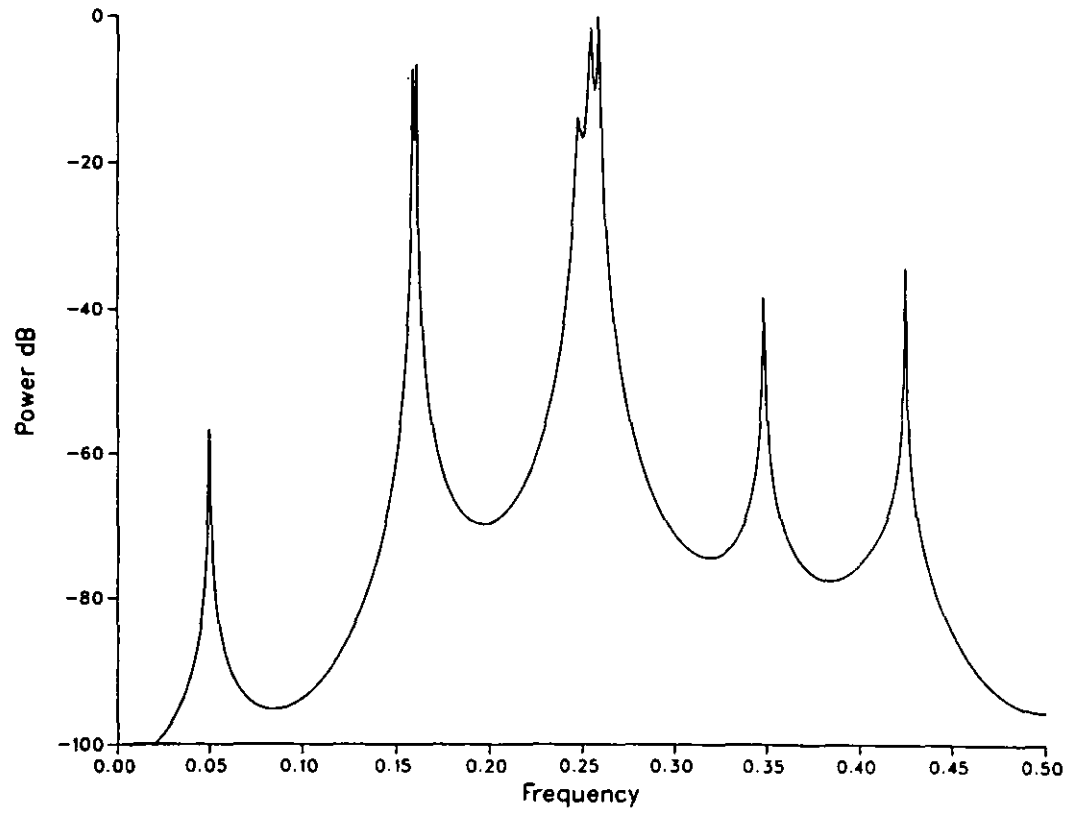
Graph 8.8.1 - Burg's method, 12 coefficients; Six sinusoids



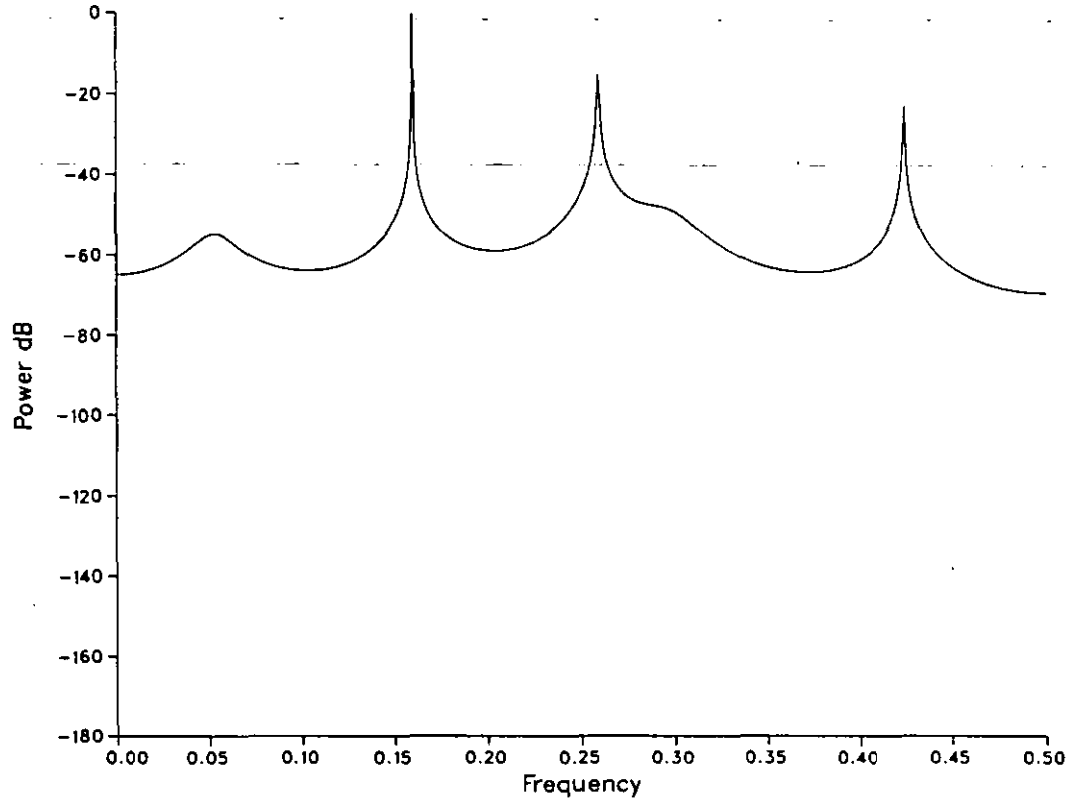
Graph 8.8.2 - Burg's method, 14 coefficients; Six sinusoids



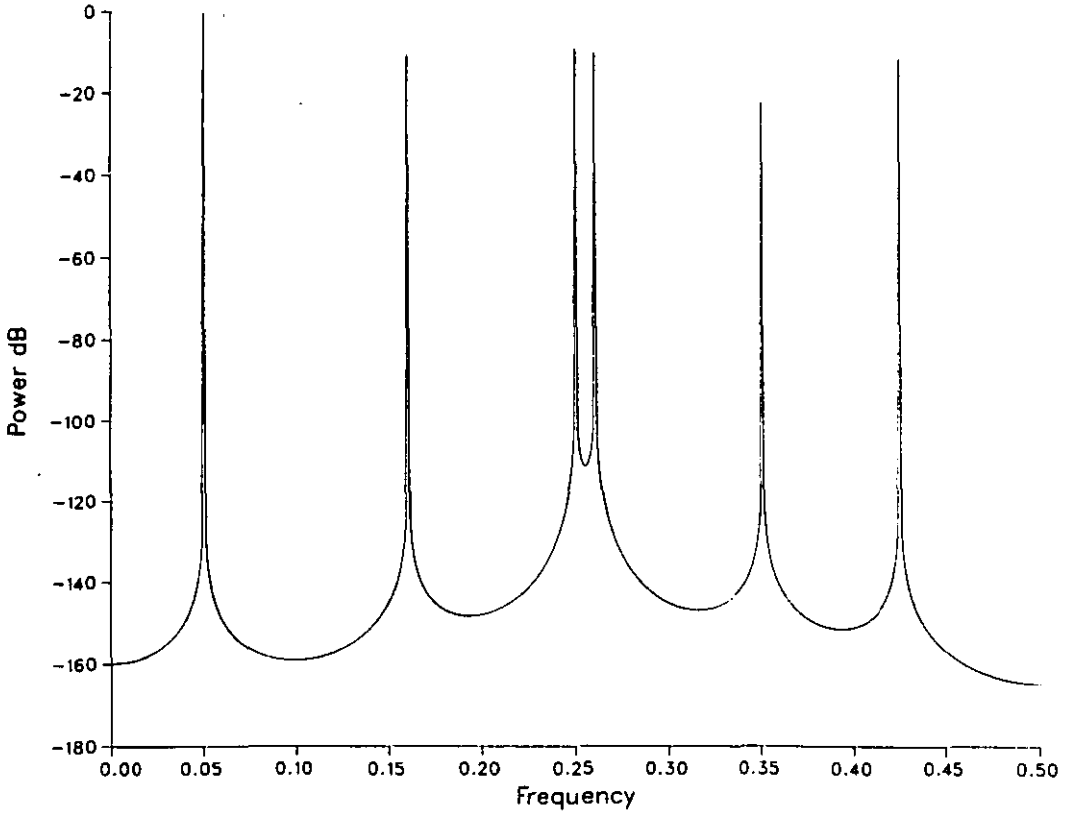
Graph 8.8.3 - Burg's method, 16 coefficients; Six sinusoids



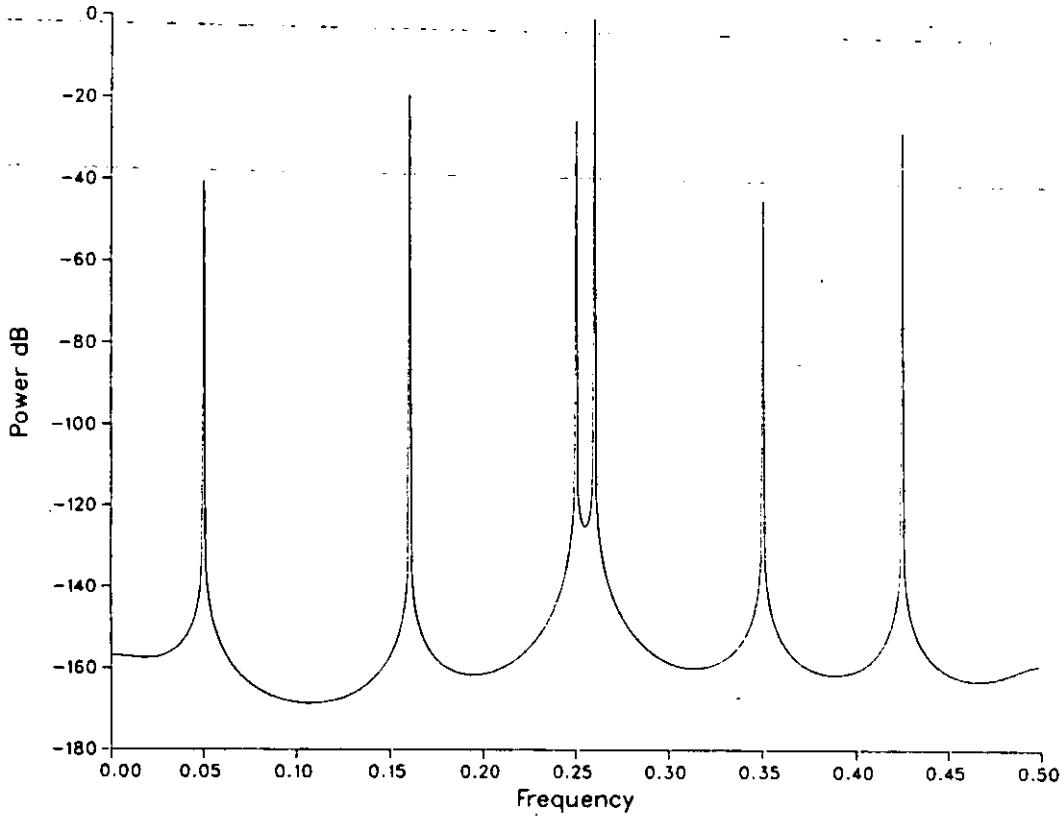
Graph 8.8.4 - Burg's method, 18 coefficients; Six sinusoids



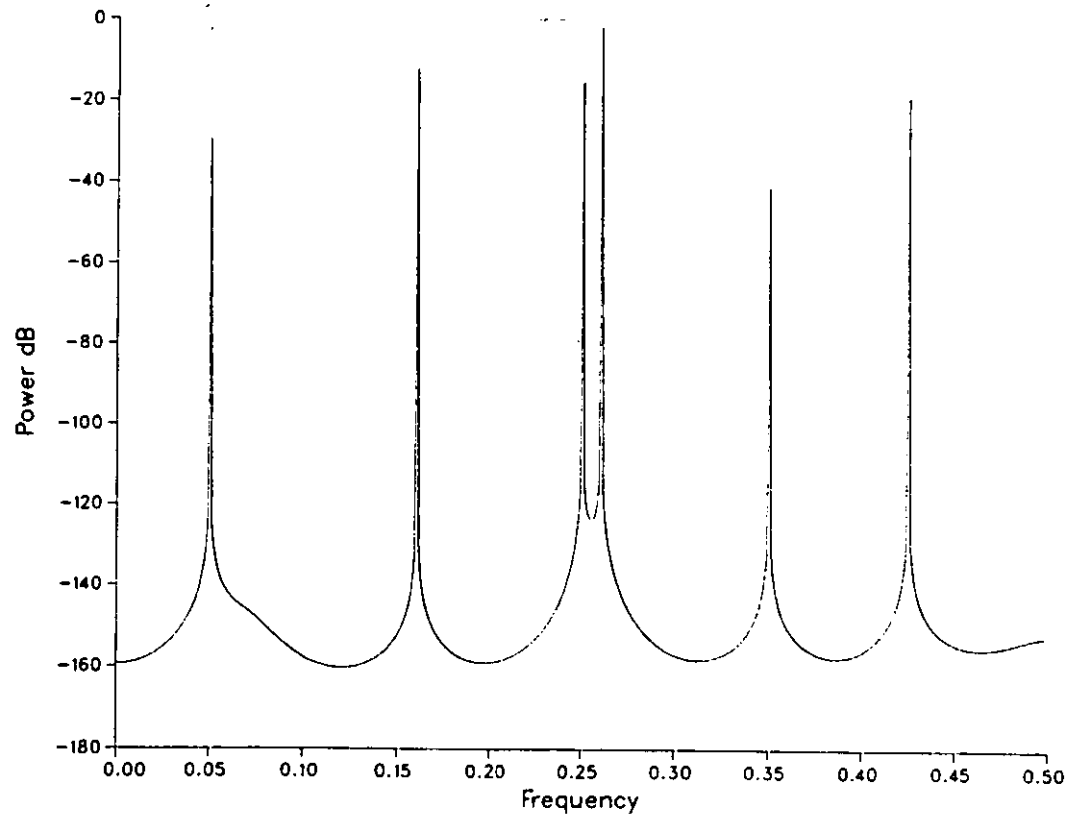
Graph 8.9.1 - FBLS technique, 11 coefficients; Six sinusoids



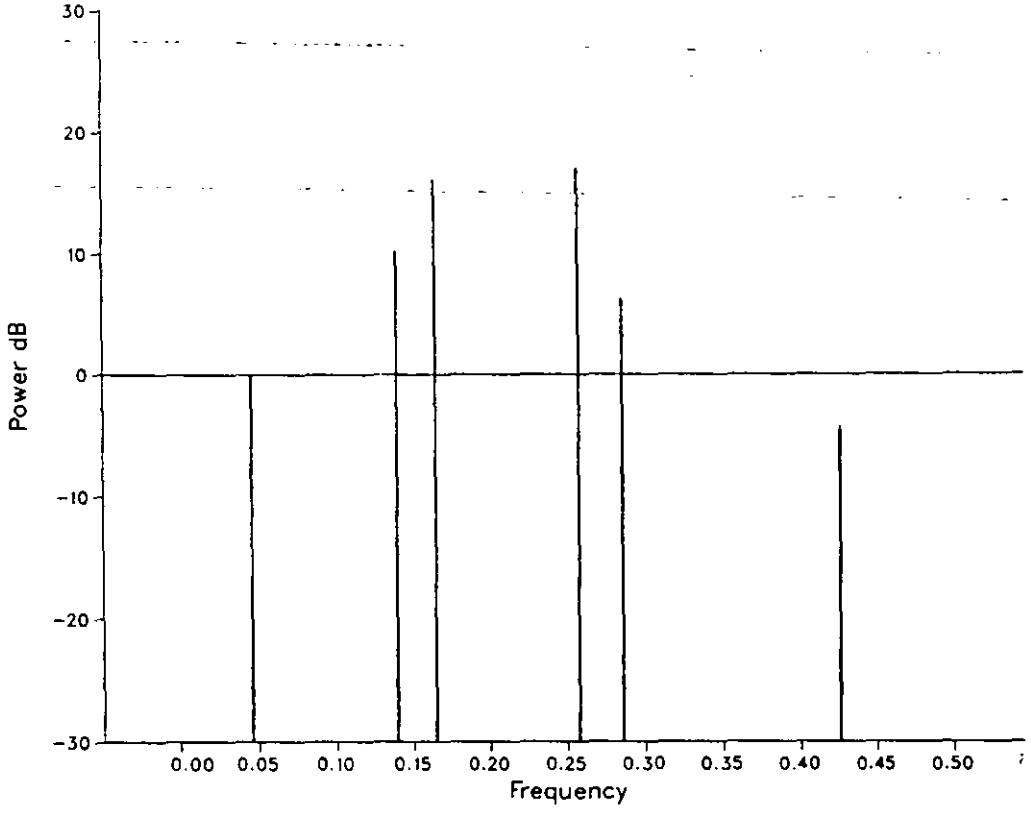
Graph 8.9.2 - FBLS technique, 12 coefficients; Six sinusoids



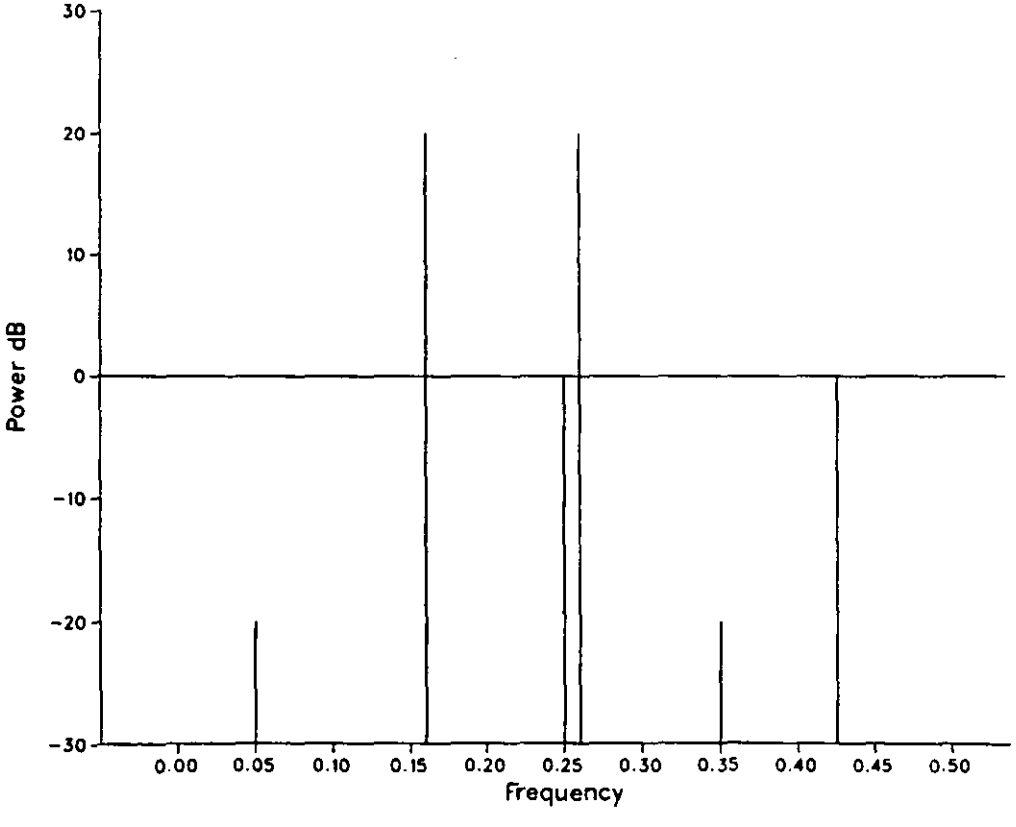
Graph 8.9.3 - FBLS technique, 14 coefficients; Six sinusoids



Graph 8.9.4 - FBLS technique, 18 coefficients; Six sinusoids



Graph 8.10 - Pisarenko, six root pairs; Six sinusoids



Graph 8.11 - Prony, six root pairs; Six sinusoids

CHAPTER 9

Further assessment of FBLS and Prony's techniques

9.1 Introduction.

In this chapter further assessment is made of the performance of the Prony and the forward-backward least squares techniques. This involves testing them with simulated noisy signals and developing rules and algorithms for order selection. Finally an assessment is made of each technique using a real signal recorded from an aero engine mounted transducer.

9.2 Test Signal.

To simulate the type of noisy signals that might be encountered from an engine mounted transducer (e.g. strain gauge, pressure transducer or accelerometer), four data sets constructed from six sinusoids in varying amounts of gaussian noise have been generated. These data sets are 128 samples in length and the sinusoids are all of different phase and varying in amplitude. The six sinusoids are as follows

Frequency [fraction of Fs]	Amplitude [peak]	(dB)	Phase [degrees]
0.05	1.0	(0)	0
0.16	10.0	(20)	60
0.25	1.0	(0)	45
0.26	10.0	(20)	120
0.35	4.0	(12)	180
0.425	1.0	(0)	90

Six sinusoids is considered a fairly typical amount to find

in an average vibration signal and the amplitude range of 20 dB corresponds to vibration signals differing in power by a factor of 100. This is also considered typical for aero engine signals and is a large enough range for most types of analysis.

To simulate noise a random number generator with a gaussian distribution (provided as part of the Loughborough university NAG library) was used to produce the noise data, this was added to the above sinusoids at four different levels. Note that the time response of the four noise sequences are all identical in shape, but different in gain or standard deviation - [σ]. The noise sequences were chosen with standard deviations of 0.25, 0.5, 1.0 and 2.0.

Noise power of a gaussian distribution is defined as σ^2, thus the total noise power in each of the simulated data sequences is as follows

σ	Noise power
0.25	-12 dB
0.5	-6 dB
1.0	0 dB
2.0	6 dB

Fourier transforms of the four data sequences are shown in graphs 9.1.1 to 9.1.4. Two points should be noticed from these spectra, i) the y-axis is referenced to peak amplitude rather than rms and ii) the noise power is spread and reduced by 1/64 (-18 dB) across all of the frequency bins. Thus for example, when the standard deviation is equal to 2, the noise floor across the spectrum is

$$20.\lg(2) + 20.\lg(2) + 10.\lg(1/64) = -9 \text{ dB}$$
$$\text{[noise]} \quad \text{[rms->peak]} \quad \text{[spread]}$$

It should be noted that the spreading effect of the noise power across the entire spectrum does mean that sinusoidal

components can be identified in poor noise conditions. It can be seen that the components with an peak amplitude of 1 (-3 dB power) can still be identified even when the total noise power is 3 dB higher (σ = 1), because the noise floor is actually 15 dB lower than the sinusoidal components. However these components then disappear when the total noise power is 9 dB higher (σ = 2), even though the noise floor is still 9 dB lower, because of the uneven spread of the noise power. This uneven spread could be reduced by ensemble averaging which would flatten the noise floor but this would also slug any transient changes from spectrum to spectrum.

9.3 Forward-backward least squares (FBLS) technique.

As demonstrated in chapter 8, the forward-backward least squares technique provides good frequency estimation and resolution but very poor power estimation. When this technique was tried on the simulated data sequences this was again found to be the case. Refering to graphs 9.2.1 to 9.2.4 it can be seen that when σ = 0.25 and 0.5 the components at 0.25 and 0.26 Fs can be distinguished apart and that the frequency estimates are very good. Note that the choice of 30 AR coefficients is arbitrary (and also the maximum that this particular program could accept). If more than 30 coefficients had been used then the two components at 0.25 and 0.26 Fs would almost certainly be distinguished in the σ = 1.0 and 2.0 cases as well, as will be seen.

Although the sinusoid power levels are very inaccurate (note that the graphs have been normalised to the largest component), the noise power level is significantly lower than in the Fourier transform case. The shape of the noise power, as with the Fourier transform is generally the same irrespective of its amplitude, thus displaying some stability within the algorithm.

At this stage the FBLS technique looks more promising than

the FFT for frequency resolution and estimation even when there is severe noise. It is quite apparent however that the power estimation needs to be considerably better to enable this technique to be of any use. The problem with the power estimation as it stands is that it uses the residual power from the AR filter. In most cases this is small and related more to the power taken out of the time series by the AR filter rather than to the power actually in the time series. To improve on this a different approach is necessary.

9.3.1 Modified forward-backward least squares technique.

It has been demonstrated and well documented in many papers that the DFT is in fact a loosely disguised least square (LS) algorithm in which the frequency matrix is made up of predetermined and harmonically related sinusoids. The unknown matrix contains the amplitude and phase information for each of these predefined sinusoids. When a time series contains a sinusoid of the same frequency as one of those in the frequency matrix, very accurate amplitude estimates are obtained. The Prony spectral line estimation technique blatantly uses an LS technique to perform its amplitude estimation. The difference between this and the DFT approach being that the frequencies chosen for the frequency matrix are first estimated from the time series and are thus not at fixed frequencies or harmonically related. As has been demonstrated, the Prony technique also gives very good amplitude estimates. This LS amplitude estimation technique can be applied to the FBLS technique.

To use the LS amplitude estimation technique a frequency matrix is required, hence the AR coefficients found via the FBLS technique must be used to directly calculate the complex roots, and thus the frequency components, of the AR filter. This of course is very similar to the approach taken in the Prony technique. Applying the same polynomial rooting NAG routine as used in the Prony Fortran program, to the FBLS AR

coefficients produces some very encouraging results, these are shown in table 9.1 (page 202). This table shows the roots, and their frequencies, produced by the polynomial rooting applied to 30 AR coefficients produced by the FBLS technique on the simulated data with $\sigma = 0.5$.

As can be seen the frequencies of the six sinusoids have been estimated reasonably accurately. The worst component being that at 0.25 Fs which has an error of 0.1%, this is quite small when considering that it is still only equivalent to one filter of a 1024 point DFT.

Applying the 15 complex root pairs found above, to the same LS amplitude estimation routine as used in the Prony technique, causes an overflow and fatal error in the matrix inversion section of the routine. It was found that this is due to two of the roots sitting on the real axis (i.e. frequency = 0.0 or 0.5 Fs). When these two roots are removed the power estimation routine works without error; table 9.2 columns 1 and 2 show the results of this LS routine. All of the amplitudes are reasonably accurate except for the rather high 0.25 Fs component.

After trying various modifications to the basic approach, as described above, it was found that there are two ways in which the results to the LS power estimation can be significantly improved.

1) As can be seen from table 9.1 column 2, the magnitude of the roots for the six sinusoids are very close to unity, but the other roots (all noise components) are not so close. Now the final algorithm is supposed to extract only pure sinusoids from a time series, thus to aid in this, all the roots can be modified in magnitude so that they locate exactly on the unit circle. The new amplitude estimates after placing all the roots on the unit circle are shown in table 9.2 column 3.

2) As already described, the roots located on the real axis had to be removed from the root set for the matrix inversion to occur without error, this can be taken a stage further by removing any roots not close to the unit circle. After some trial and error it was found that a difference of ± 0.025 in the root magnitude from unity provided a suitable breakpoint for this technique. The results for this are shown in table 9.2 column 4.

By using both of these techniques even better amplitude estimation can be achieved, this is shown in table 9.2 column 5, the complete algorithm used to obtain this will now be refered to as the modified FBLS technique. As can be seen the amplitude estimation is now very good, especially considering that the signal is embedded in noise.

9.3.2 A.R. filter order estimation.

Although the modified FBLS technique gives very good results, there is still one underlying problem. The number of AR coefficients used to obtain these results has been entered into the program by the author and is certainly not guaranteed to be the best choice. The effect of choosing different numbers of coefficients is shown in graphs 9.3.1 to 9.3.4, 9.4.1 to 9.4.4 and 9.5.1 to 9.5.4, where the modified FBLS technique has been applied to the four sets of simulated data using 20, 30 and 40 AR coefficients respectively.

It can be seen that the number of AR coefficients used in the modified FBLS technique can make significant differences to the resulting spectra. When the number of AR filter coefficients is too low some of the smaller components are missed, as in graph 9.3.4, and when it is too high noise components appear in the spectrum, as in graph 9.5.1. In general, as the noise power increases then the number of coefficients required to extract the sinusoidal components must also increase. There comes a point however when it is

impossible to extract all of the sinusoids without including some noise components in the spectrum. The following two important features can be seen from these graphs

1) From all indications it would appear that no matter how much an AR filter is over determined there is no frequency splitting.

2) Even when noise components are introduced into the spectrum, their amplitudes are not unreasonable and do seem to be stable, as shown in graphs 9.5.1 to 9.5.4 where the noise components stay in very similar positions.

Over determining the AR filter does not seem to present any significant problems to the estimation of frequency or amplitude. Refering to table 9.3 which shows the frequencies and amplitudes of the components in graphs 9.3.2, 9.4.2 and 9.5.2 there appears to be no degradation of the frequency and amplitude estimates of the sinusoidal components as the number of AR coefficients is increased from 20 to 40. The four extra components in the 40 coefficient case are not classed as spurious because they clearly represent colouration in the noise floor. The amplitude of the 0.26 Fs component changes significantly between 20 and 30 coefficients because of the 0.25 Fs component which is not extracted for 20 coefficients but is for 30. In fact it would seem better to over determine the AR filter as the amplitude for the 0.25 Fs component is most accurate when 40 coefficients are used. These findings indicate that good results can be obtained without the risk of spurious components or inaccurate estimates when the AR filter is over determined.

To show how the number of AR coefficients and the level of noise affect the extraction of sinusoidal components a graph showing two limits has been constructed, see figure 9.1. The first limit shows the minimum number of AR coefficients

required to extract all the sinusoids irrespective of whether noise components are also included. The second limit shows the maximum number of AR coefficients that can be used before noise is introduced into the spectrum. Note that the x-axis is referenced to the smallest sinusoidal components, for a reference to the largest add 20 dB. It can be seen that while the noise power is approximately 1 dB or more below the sinusoid power, then there is a region in which all of the sinusoids can be extracted without any noise components occuring . As the noise power increases then sinusoids are either lost, noise components are extracted, or both. As already stated the noise components are not really a problem as they are of acceptable amplitude, and in general they can be identified if the approximate noise power is known.

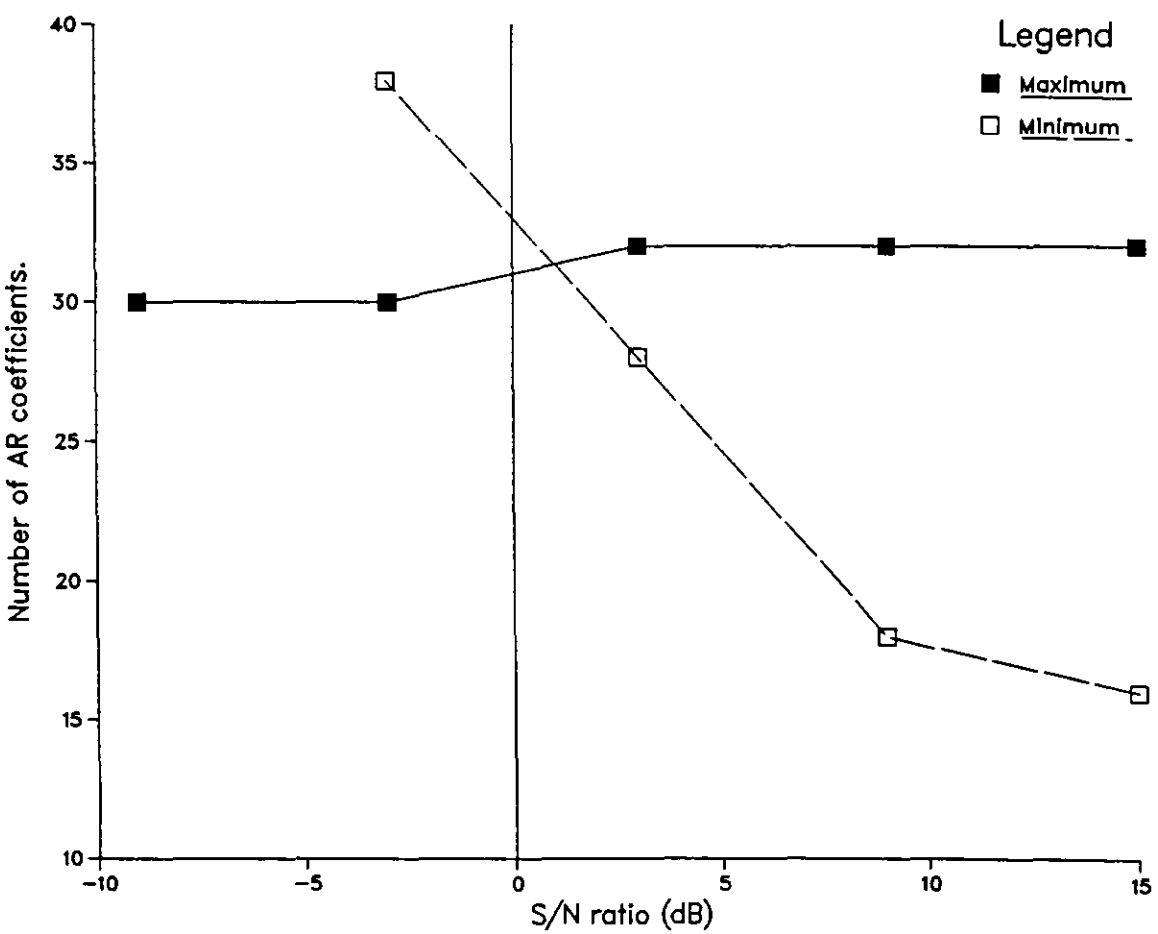


Figure 9.1 - Modified FBLS technique; Maximum and minimum number of AR coefficients against signal to noise power

In most applications envisaged it will be required that all sinusoids present within a signal are extracted. Thus for a given signal to noise power ratio the number of AR coefficients used should be greater than or equal to the corresponding number indicated by the minimum plot. Note that for six sinusoids, the minimum plot is asymtotic to 12 coefficients as noise power is decreased, and that the maximum number of AR coefficients is determined by the array boundries of the program (in this case 40). It is interesting to note that the maximum number of coefficients before noise is extracted is almost constant, this approximately represents the point at which the AR filter becomes overdetermined.

Taking into account the above points, it is possible to derive some basic rules in respect of the number of AR coefficients that should be used in this modified FBLS technique. It should be noted that these are very general and are only based on the analysis of the four sets of simulated data.

1) For signals in relatively low noise conditions (noise power at least 10 dB lower than any sinusoidal components of interest) then the number of AR coefficients chosen for the analysis should be approximately double the ideal number. This should result in all the sinusoid components being extracted without any noise components. Note that the ideal number comes from the fact that each sinusoid can be represented by a complex conjugate root pair and that each root pair is represented by two AR coefficients. Thus if there are 5 sinusoids present within a set of data, 20 AR coefficients should be used.

2) Where the noise power is greater than that allowed above, three or four times the ideal number of AR coefficients should be used. The resulting spectra should display all the sinusoids with power levels down to, or even below

the noise level. Note however that some noise components will probably also be extracted, but because the noise is wide band in nature these components will tend to be quite low in amplitude.

The above two general rules obviously require some knowledge of the spectral content of the data sequence under analysis. This information need only be approximate and could be obtained from an initial Fourier transform of the data. This will provide approximate details of the number of sinusoids present and power of the noise.

9.3.3 Akaike's criterion.

A well known method of determining the correct order of an AR filter for a given input data sequence is the Akaike method [2]. The criteria used to make this decision is based upon the residual power left when the input data sequence has been applied to AR filters of increasing order. The Akaike criterion is as follows

$$FPE(p) = P.\frac{N + (p + 1)}{N - (p + 1)}$$

FPE : Final prediction error.
P = residual mean sum square.

If FPE(p) > FPE(p-1),
then the AR filter order at (p-1) is correct.

This method of determining the correct order of an AR filter has met with mixed success from other researchers, the general comment being that it only really works adequately when the input data sequence is of a purely auto-regressive nature. This of course is not usually the case for data sequences that have come from measured signals in noise. As a result it has been found that this method usually results in the filter order being too small.

In general, as the AR filter order is increased (as naturally occurs during the recursive nature of the FBLS technique) the residual power decreases. However this reduction is not smooth but rather in sudden steps as the roots for each sinusoid are found. The residual power can also occasionally increase by small amounts, these effects are shown in figures 9.2.1 and 9.2.2. These show how the residual power, generated by the FBLS technique used on three sets of simulated data (σ = 0, 0.25 and 0.5), changes for increasing numbers of AR coefficients.

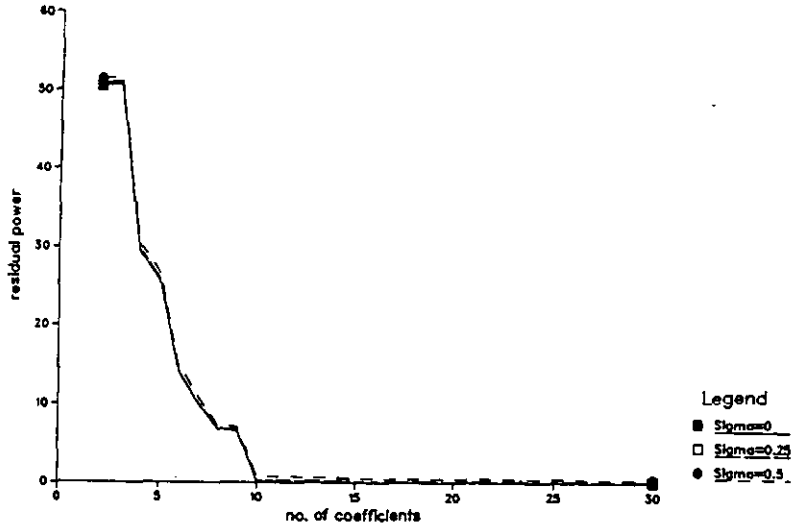


Figure 9.2.1

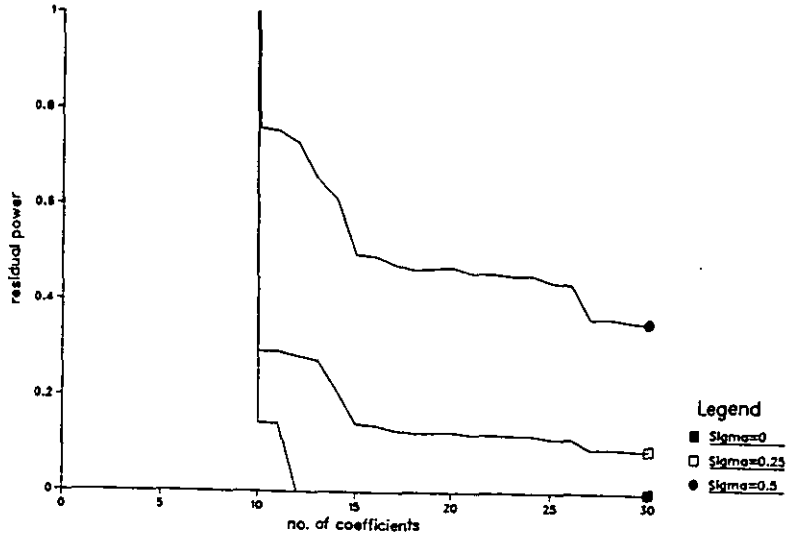


Figure 9.2.2

Residual power against coefficients for FBLS technique

If the Akaike equation is applied to the same three data sequences as above, then FPE ratios as shown in figure 9.3 are produced. The criterion to stop the recursion is that the ratio should be greater than unity, refering to the figure, the recursion should stop for all three data sequences at 10 AR coefficients. This is too early and would result in the 0.25 F(s) component not being found. It is surprising that even when there is no noise the Akaike method would stop the recursion early, as the data sequence in this case is auto-regressive. It can be seen that there are many places where the FPE ratio is greater than unity and that most of them signify nothing of importance. Judging from these results it would appear that Akaike's method of determining the filter order does not work particularly well.

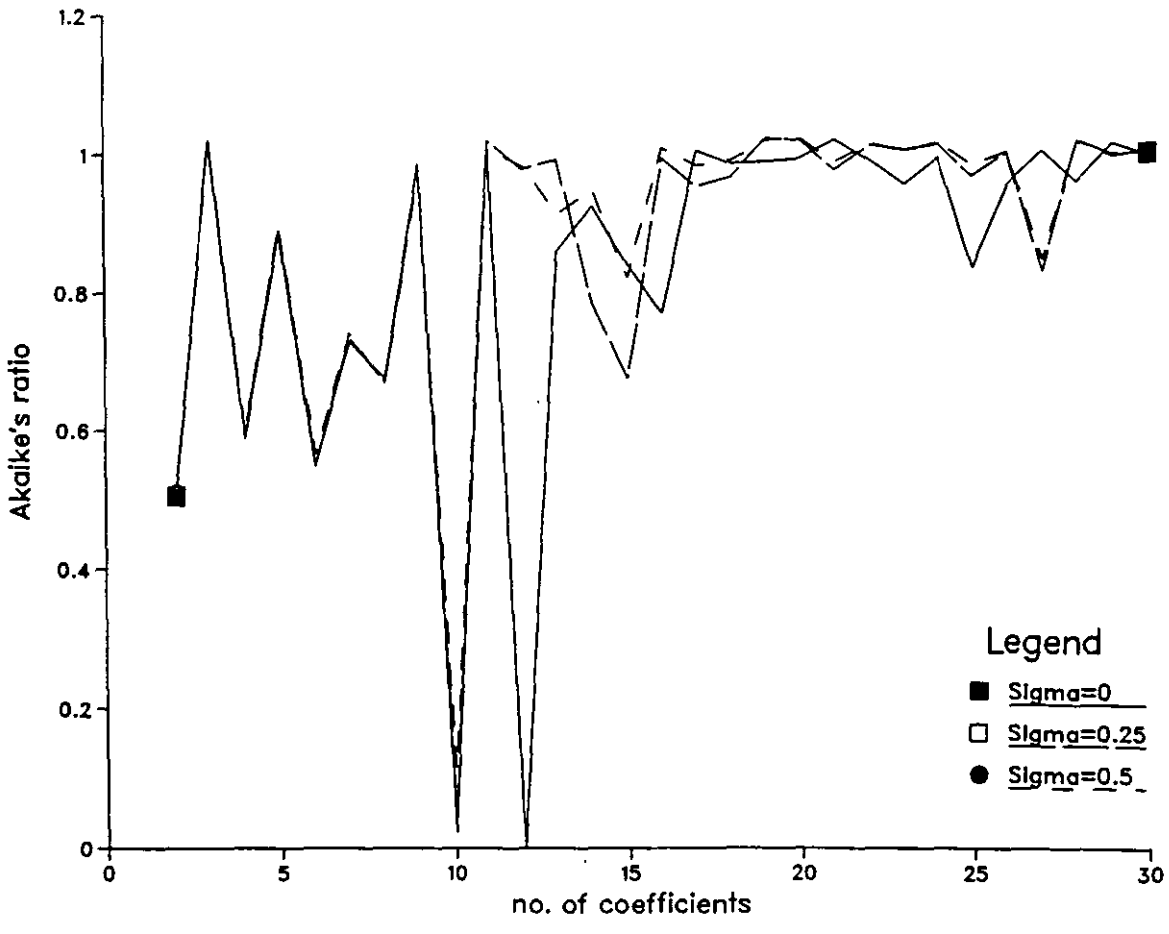


Figure 9.3 - Akaikes criterion applied to FBLS technique

9.4 Prony spectral line estimation technique.

As previously demonstrated in chapter 8 the Prony spectral line estimation (PSLE) technique is capable of estimating the frequency and magnitude of pure sinusoids very accurately, this being due to the constraints imposed by the algorithm upon the AR filter characteristics. The performance of this technique is now assessed by applying it to the same simulated data sequences as used in the FBLS technique assessment. Further development of the algorithm is also attempted.

It was found that when the PLSE technique was applied to any of the simulated data sequences, the same problem occured with power estimation as was seen with the modified FBLS technique. As described earlier, the power estimation routine involves a matrix inversion which causes a fatal run error if the roots of the AR filter are not close to or on the unit circle. To alleviate this problem in the modified FBLS technique, roots not close to the unit circle were discarded and the rest forced onto the unit circle. This was tried in the PSLE program but unfortunately was found to be an unsatisfactory solution. Due to the fundamental nature of this algorithm, in that it tries to estimate pure sinusoids only, the roots produced from the AR filter tend to be either exactly on the unit circle (to within $\pm$10e-6%) or significantly off the unit circle. In the cases where some of the roots in a root set are not on the unit circle, significant errors then occur throughout the whole root set, resulting in poor overall frequency estimation. It was found that in these cases it is not worth continuing with the power estimation, as the accuracy of this is also affected. This is demonstrated in table 9.4 (page 204) where this approach was tried on the σ = 0.5 data sequence using 30 coefficients (arbitrary choice). In this case 14 coefficients were not on the unit circle and so were discarded, power estimation has then been performed on the remaining 16 coefficients (8 sinusoids). As can be seen both frequency and amplitude

estimates are very poor.

9.4.1 Modified PSLE technique.

To attempt to get round the above problem the PSLE program was modified to analyse all AR filters, initially starting with 2 coefficients (plus Ao which is always 1) and working up in pairs of coefficients testing each set of resulting roots for a complete set on the unit circle (i.e for a "good root set"). Note that the number of coefficients by which the AR filter is increased must be two because of the double sided nature of the constrained AR filter. Each good root set is subsequently stored, and the last good root set discarded. There then comes a point when increasing the filter length produces no more good root sets, from trial and error it was found that this point comes by the time 12 bad roots have been found in consecutive bad root sets. The PSLE program was thus modified to include this test of completeness. A flow diagram of this program is shown in figure 9.4.

When the above modified PSLE technique was applied to the simulated data sequences it was found that almost all numbers of AR coefficients produced bad root sets, although there was always at least one good set found at some point. Graphs 9.6.1 to 9.6.4 and table 9.5 show the results obtained when this modified PSLE technique was applied to the four data sequences. As can be seen the algorithm has worked with varying degrees of success. When σ = 0.25 and 0.5 quite reasonable spectra have been produced, in both cases the algorithm has selected 20 coefficients as being the optimum AR filter length and has extracted all six sinusoids together with some noise components. However when the noise level is increased (σ = 1.0 and 2.0) good root sets have only been found for AR filters of 4 coefficients, not surprisingly these produce very poor results. Thus the modified PSLE technique starts to have problems somewhere between σ = 0.5

(noise power 3 dB lower than the smallest sinusoids) and σ = 1.0 (noise power 3 dB greater), i.e. approximately when the noise power and the smallest sinusoid power are equal.

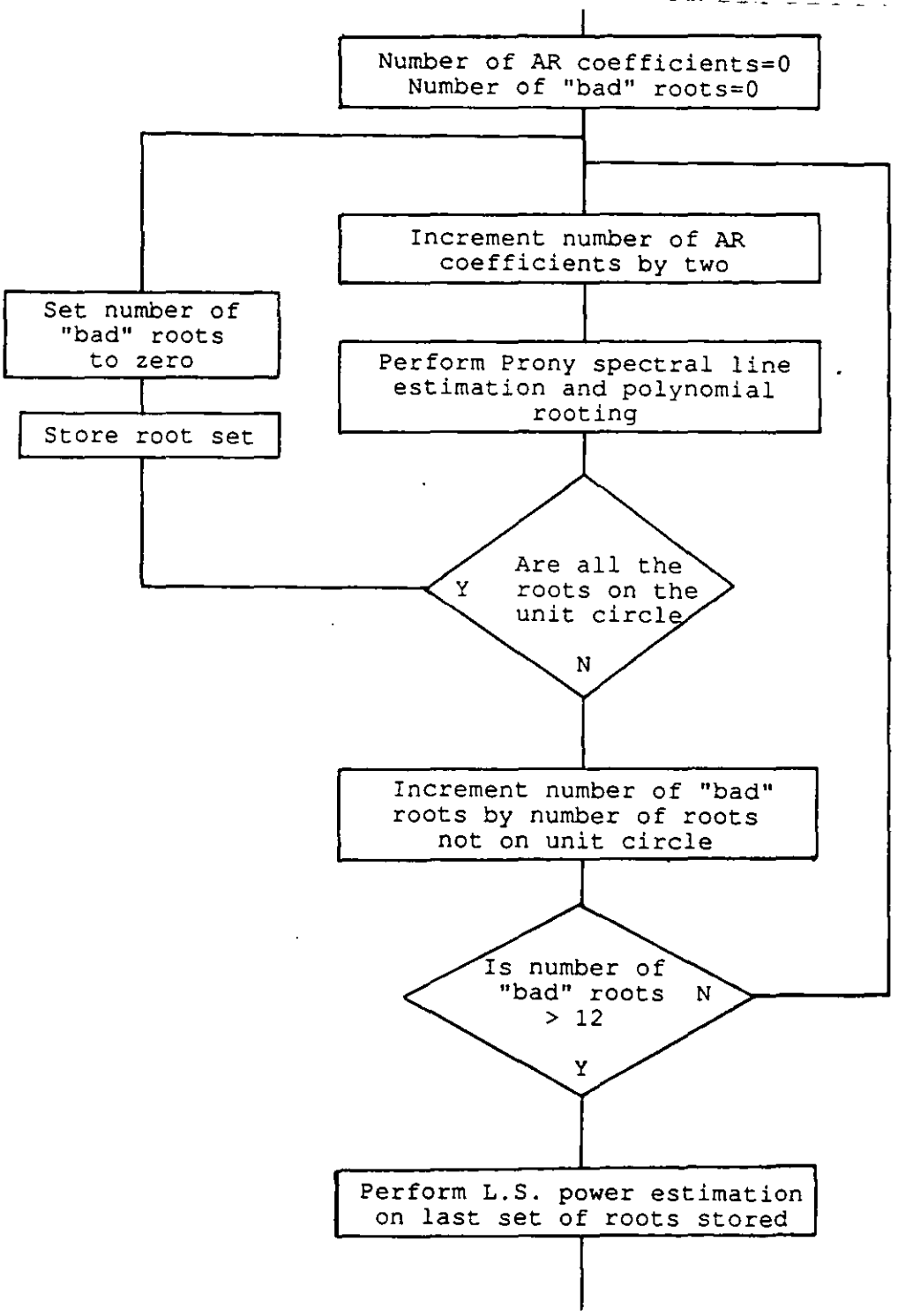


Figure 9.4 - Flow diagram of modified PSLE method

In an attempt to improve on the results obtained for σ = 1.0 and 2.0, the approach shown earlier of discarding bad roots was again tried. In each case 20 and 40 coefficients were used, the results are shown in graphs 9.7.1, 9.7.2 and 9.8.1, 9.8.2 and in tables 9.6 and 9.7 respectively. The results show that in all four cases six good root pairs have been found and also that these are exactly on the unit circle. However, the frequency estimates are very poor and do not necessarily correspond with the frequencies of the six sinusoids, this reaffirms the earlier conclusions found with this approach. These results also bring to light another problem, the combination of high noise levels and overdetermining the AR filter has caused frequency splitting to occur next to the 0.16 Fs and the 0.35 Fs components. In fact the surprisingly good estimate at 0.25 Fs is probably due to frequency splitting from the 0.26 Fs component as much as any other factor.

Even with the lower noise levels there is yet another problem, referring to tables 9.4, 9.5 and 9.6, it can be seen that the polynomial rootings of the various AR filters have produced four duplicated root pairs, these being at 0.0442 Fs - table 9.4, 0.0516 Fs - table 9.5 and 0.2579 and 0.3575 Fs - table 9.6. Apart from there being a significant error in each of the frequency estimates, the amplitude estimates are also poor.

The results for the PLSE technique indicate that there is a point at which the noise level becomes too high for the algorithm and that inaccurate and misleading spectral components occur above this point.

9.5 Comparison between the modified FBLS and PSLE techniques used on simulated data.

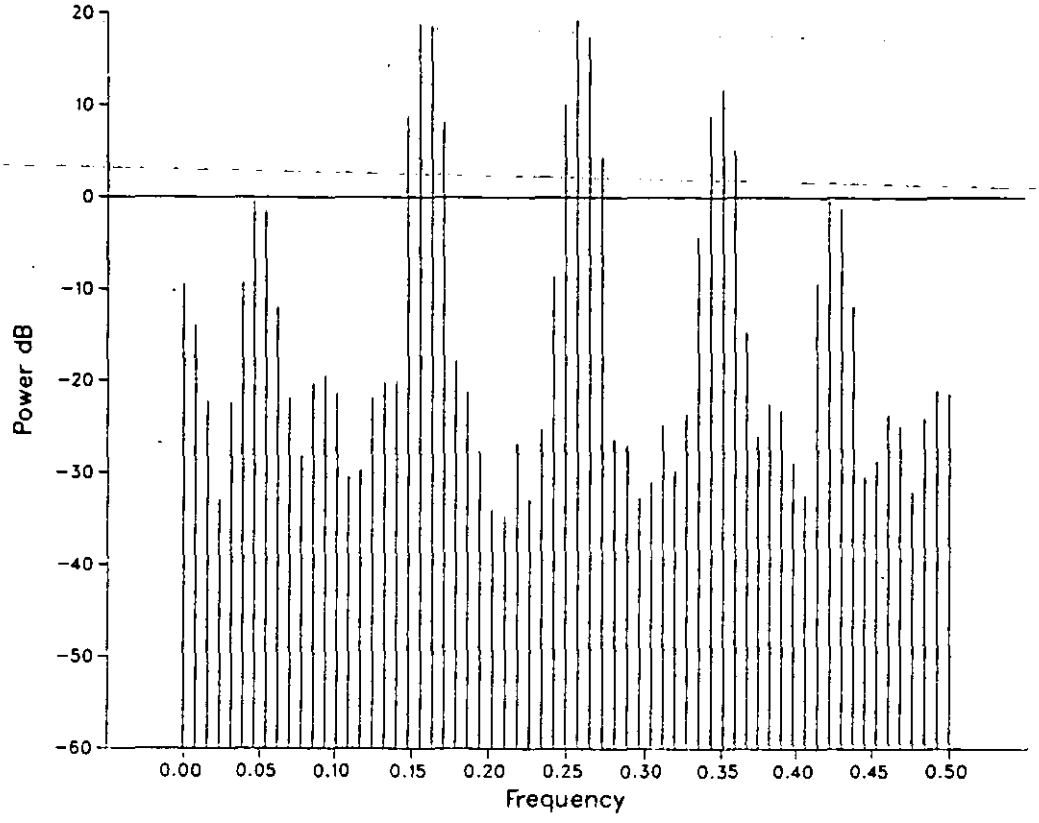
Comparing the results obtained using the modified FBLS and PSLE techniques the following points can be made

1) At noise levels such as σ = 0.5, the results from the FBLS technique are as accurate or better than those from the PSLE technique. (especially with 40 AR coefficients used in the FBLS technique). This is not particularly a poor reflection on the PSLE technique however as it only uses about half the number of AR coefficients to do the same analysis.

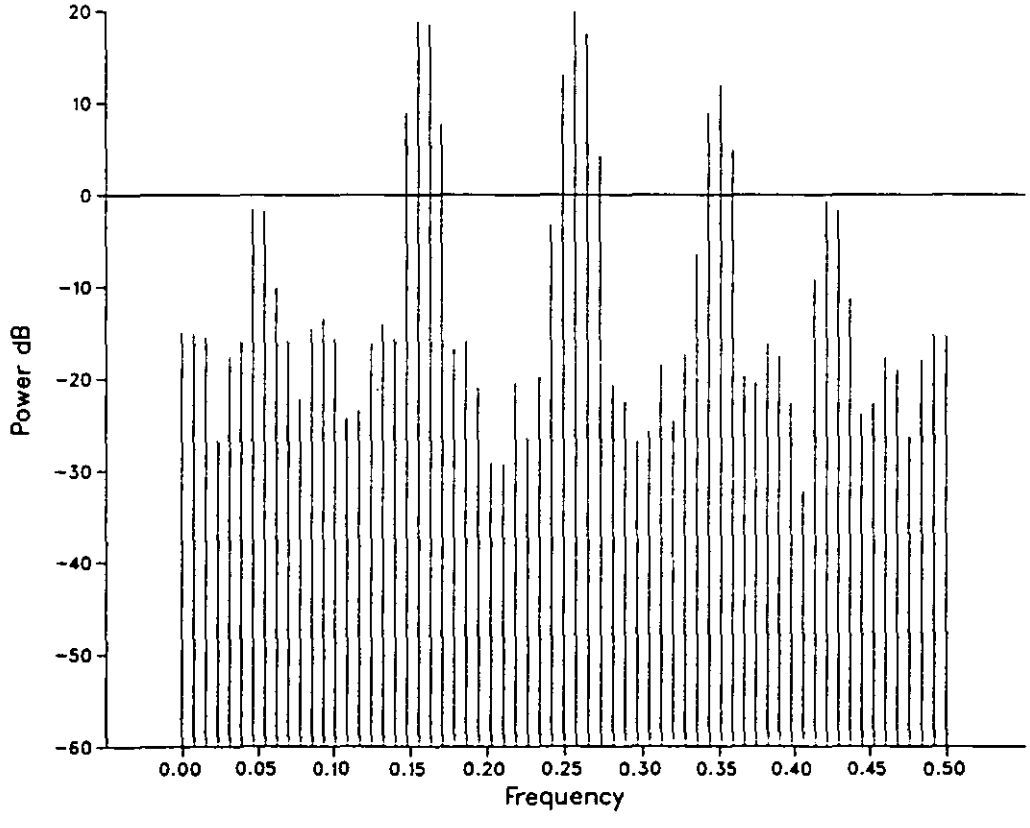
2) When there are significant noise levels (e.g. σ = 1 or 2) the PSLE technique is inaccurate and misleading whereas the FBLS technique can still accurately extract the larger components from the data sequences while the smaller components are simply lost in noise components of equivalent power levels.

3) The PSLE program has to find many AR filters to obtain a good root set. This is computationaly very intense especially as there is no recursion involved in the algorithm. The modified FBLS technique on the other hand, can either recursively estimate the AR coefficients up to the desired number, or perform one full FBLS estimation for that number of coefficients.

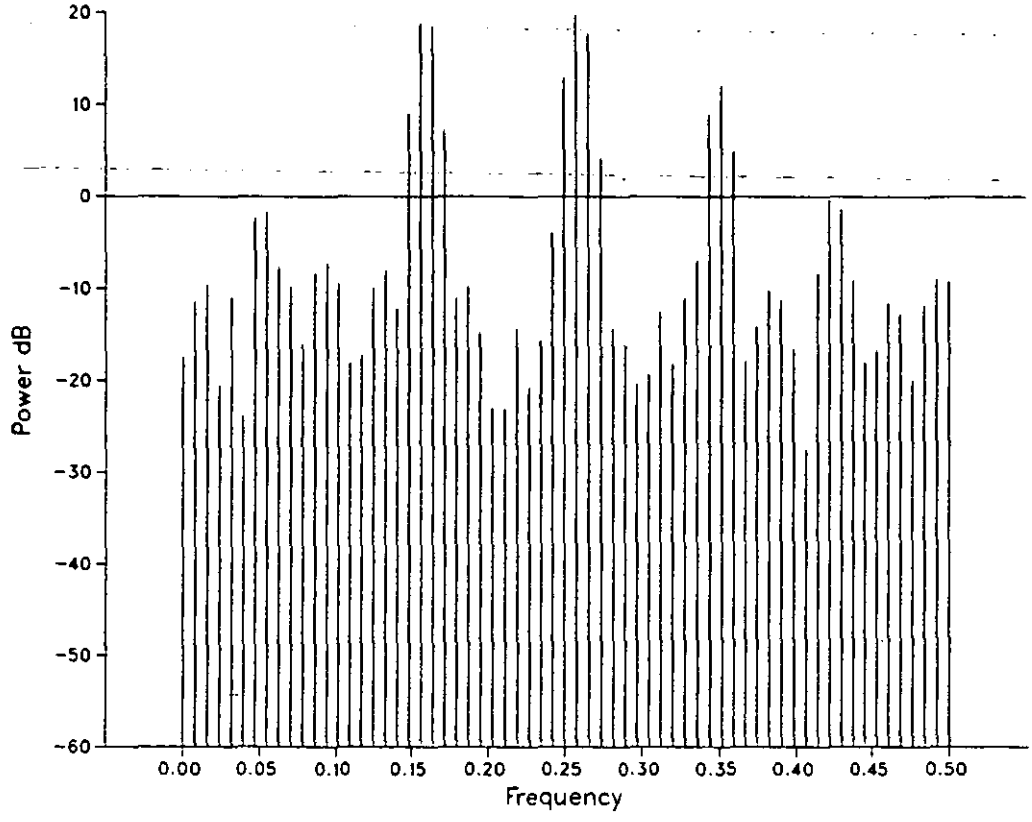
In general the PSLE technique does not appear to be as flexible as the FBLS technique when it comes to signals which are not totally comprised of pure sinusoids. This is not surprising when considering the constraints set up on the AR filter within the PSLE algorithm.



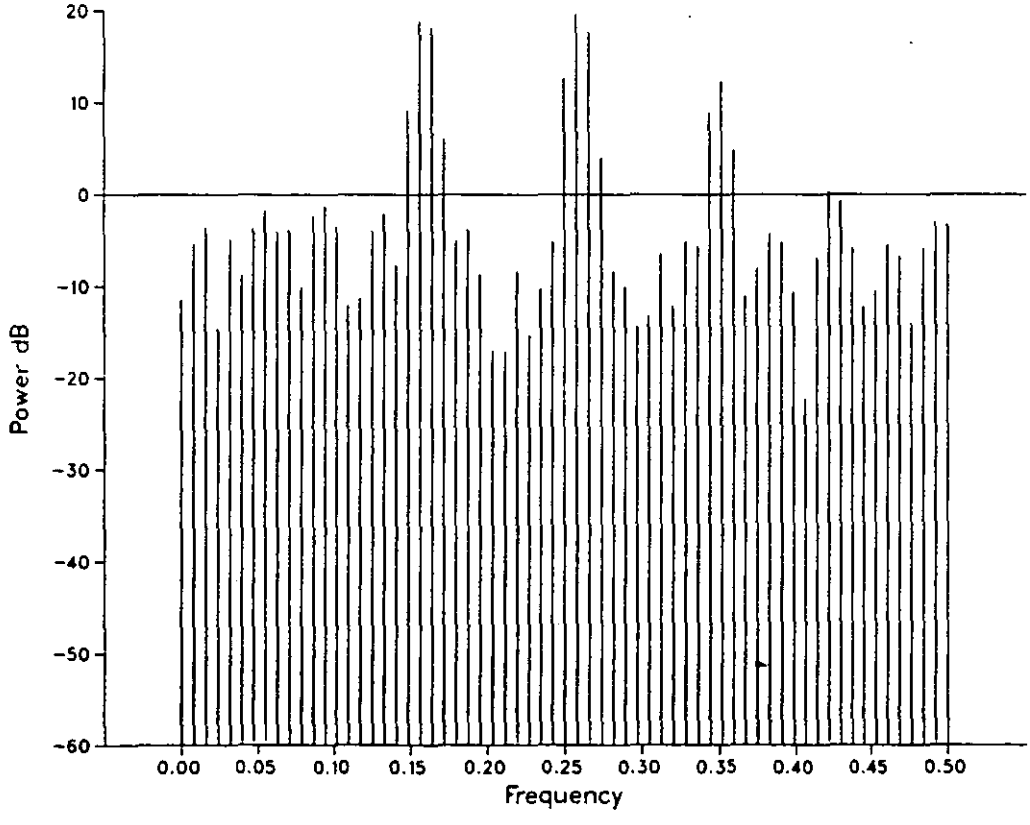
Graph 9.1.1 - DFT of simulated noisy signal; σ = 0.25



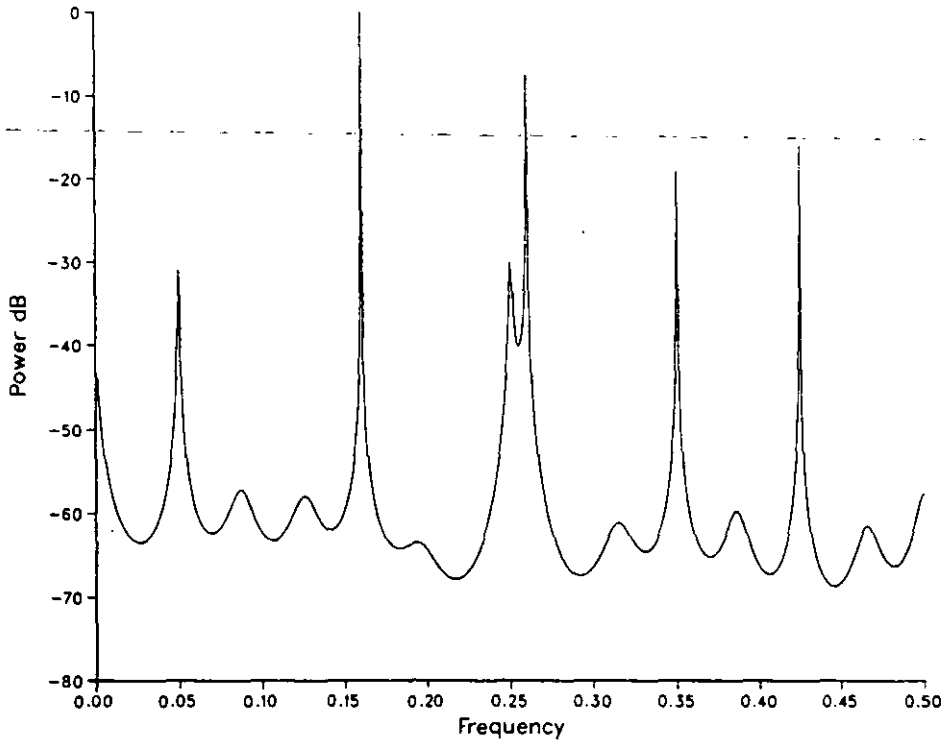
Graph 9.1.2 - DFT of simulated noisy signal; σ = 0.5



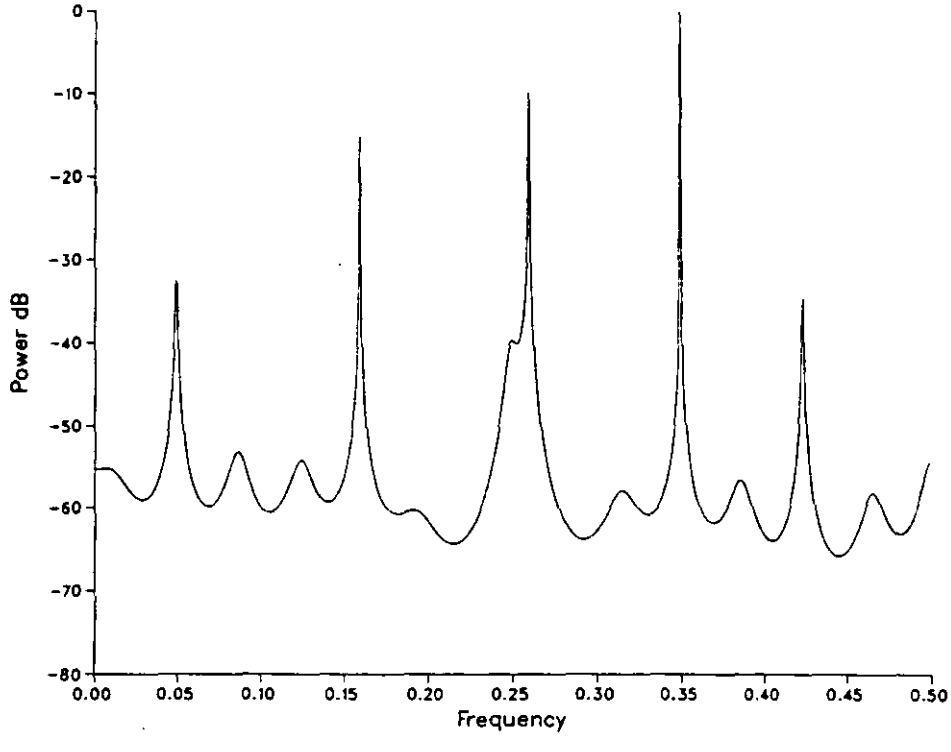
Graph 9.1.3 - DFT of simulated noisy signal; σ = 1.0



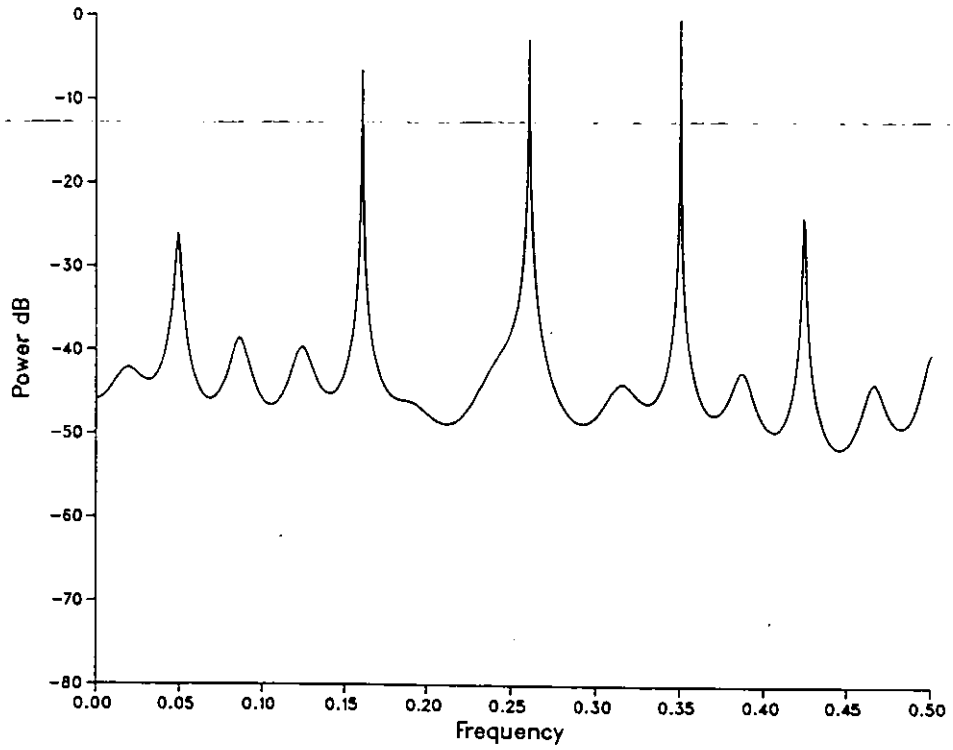
Graph 9.1.4 - DFT of simulated noisy signal; σ = 2.0



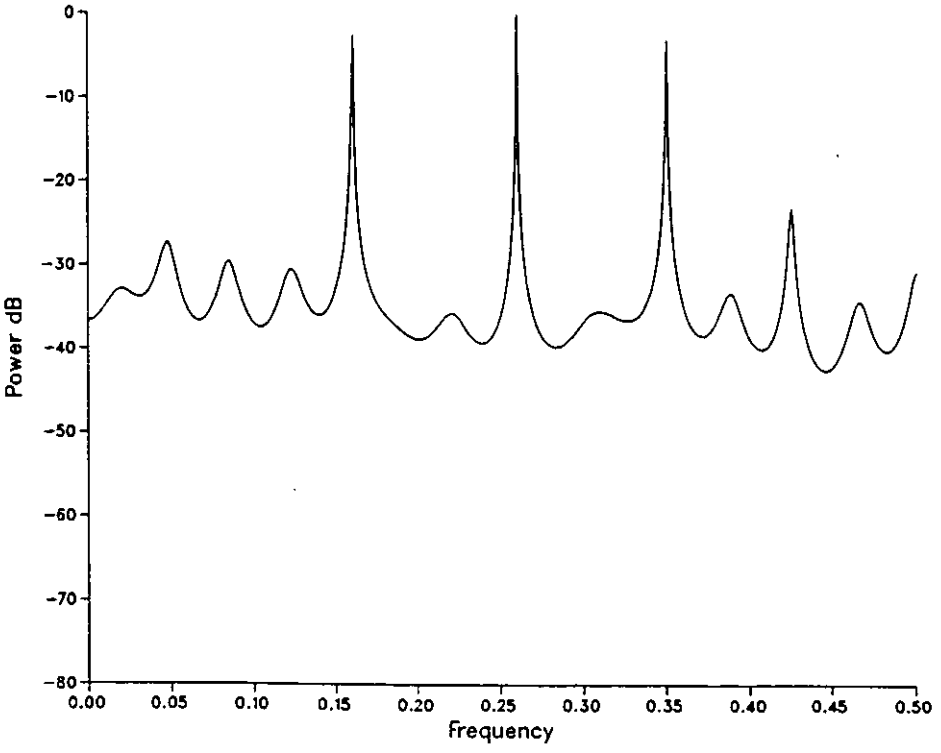
Graph 9.2.1 - FBLS technique, 30 coefficients; σ = 0.25



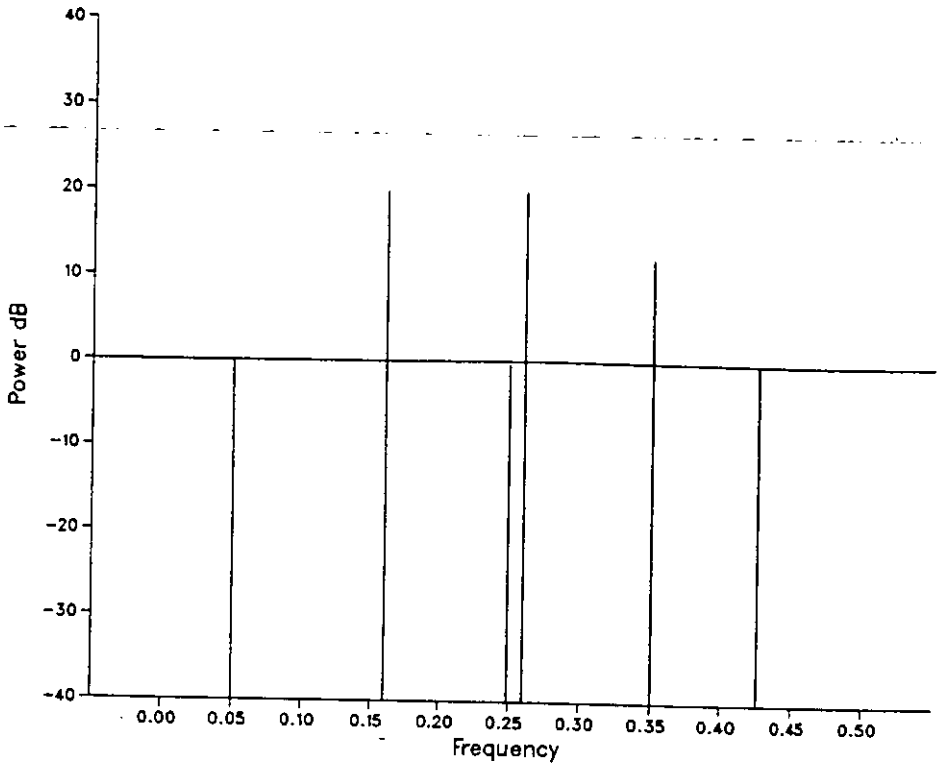
Graph 9.2.2 - FBLS technique, 30 coefficients; σ = 0.5



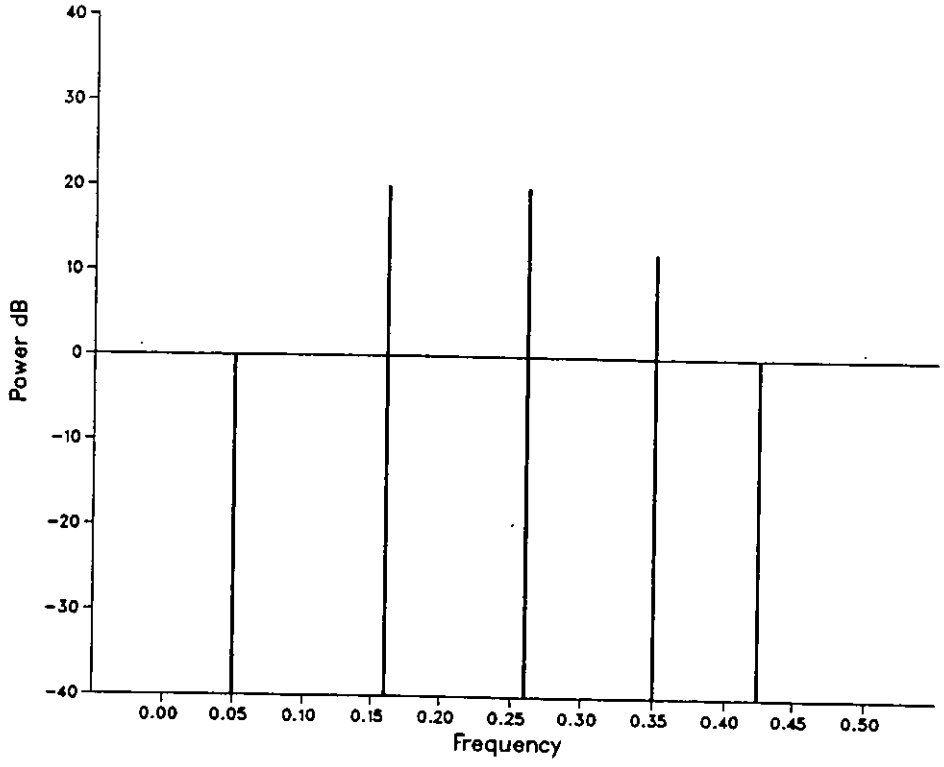
Graph 9.2.3 - FBLS technique, 30 coefficients; σ = 1.0



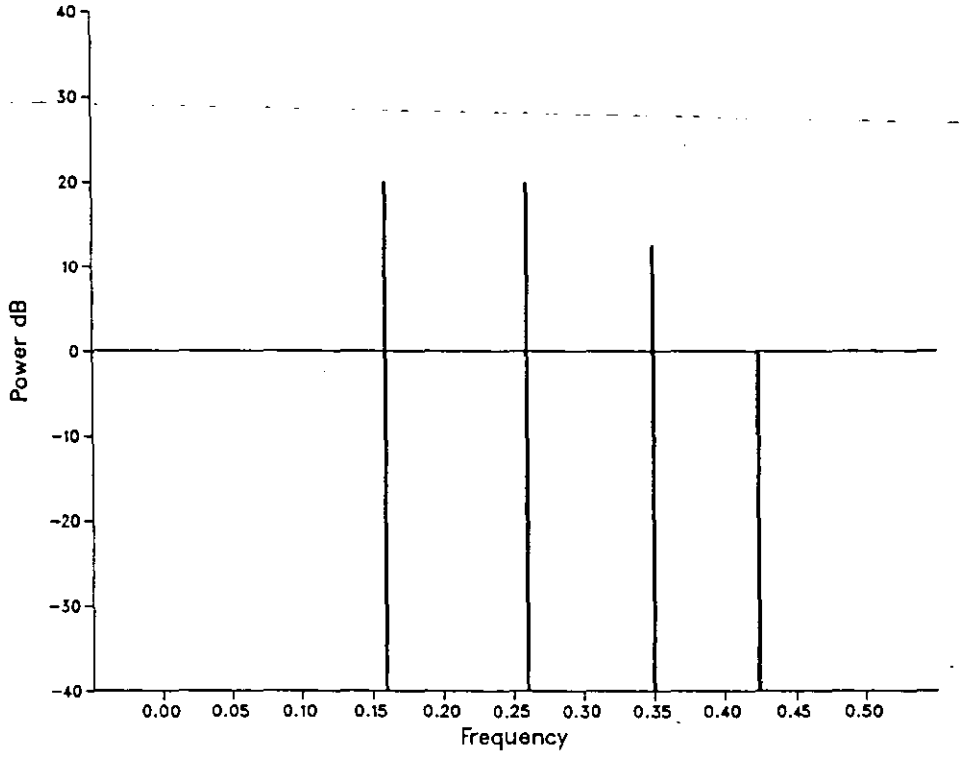
Graph 9.2.4 - FBLS technique, 30 coefficients; σ = 2.0



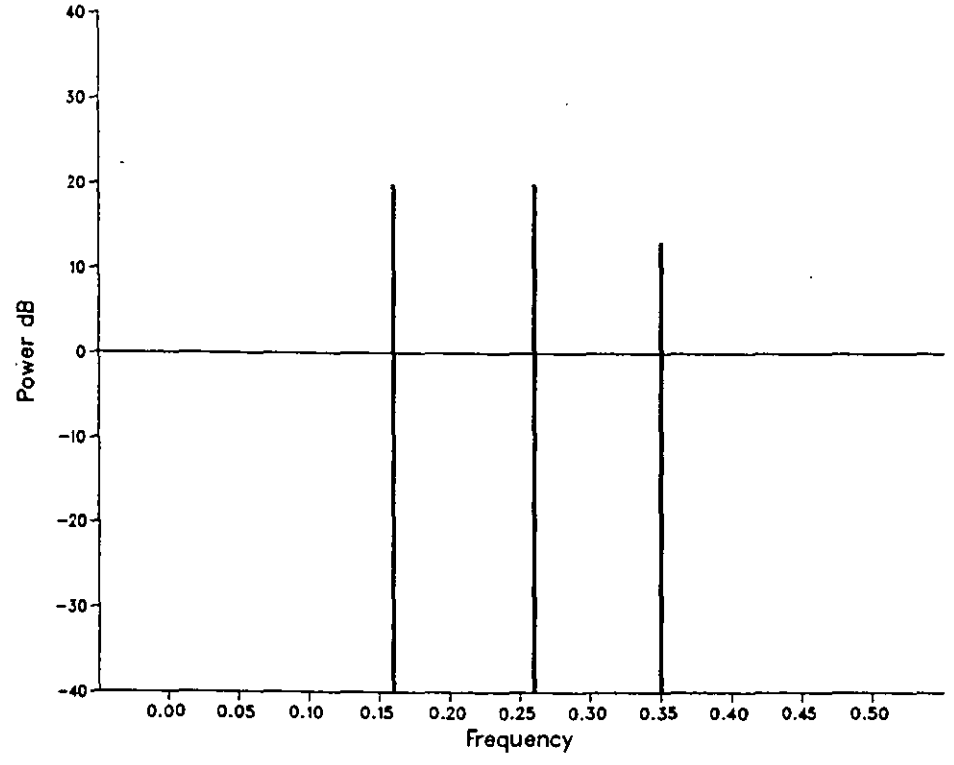
Graph 9.3.1 - Modified FBLS technique, 20 coeff.; σ = 0.25



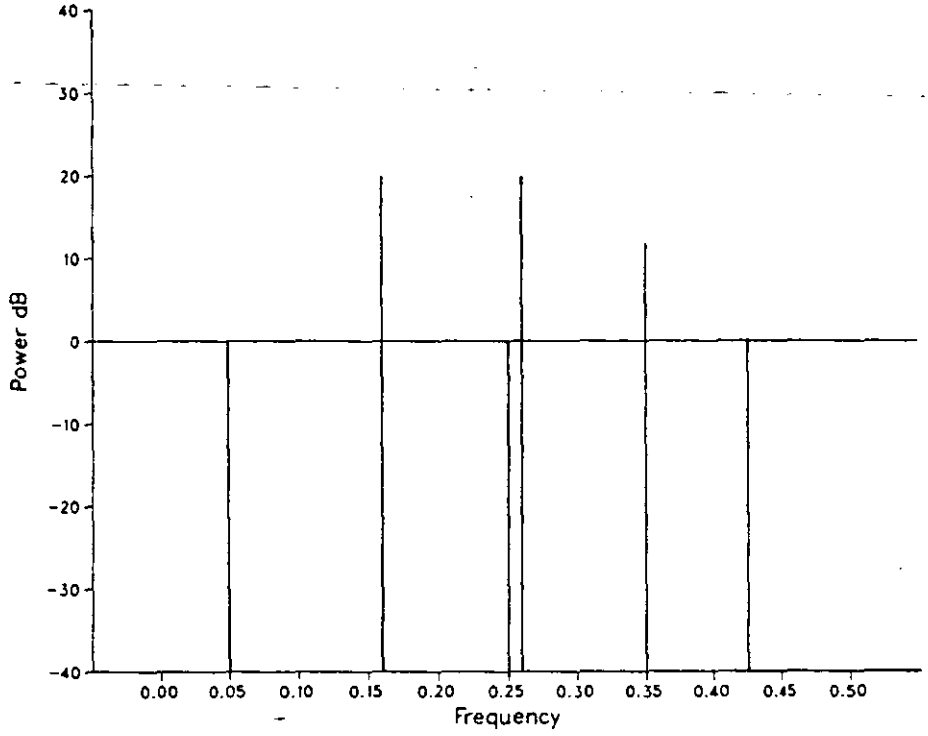
Graph 9.3.2 - Modified FBLS technique, 20 coeff.; σ = 0.5



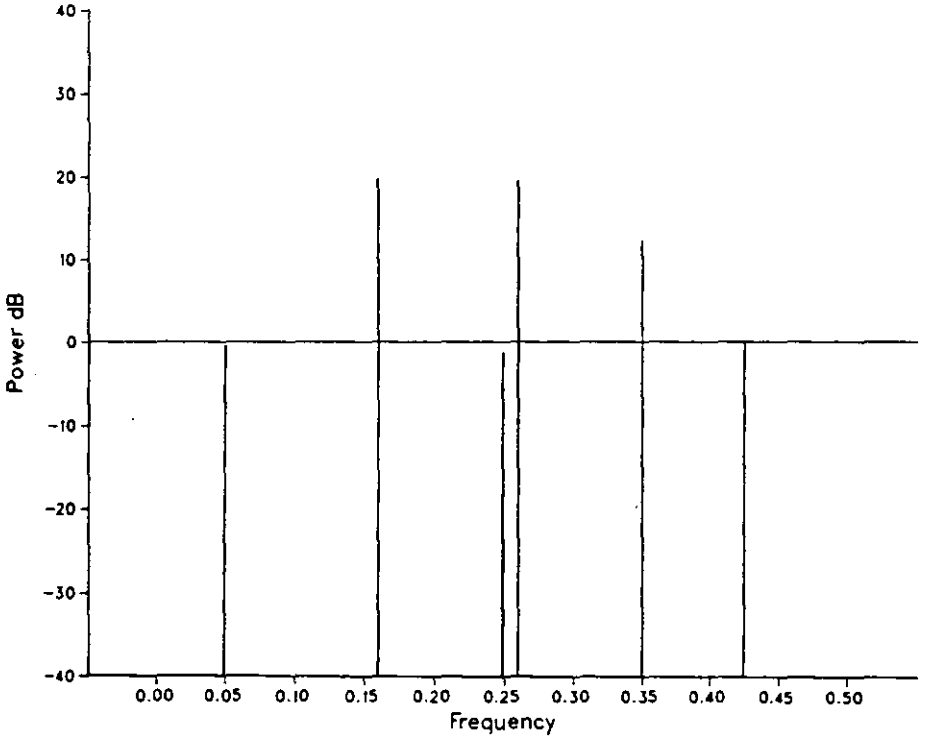
Graph 9.3.3 - Modified FBLS technique, 20 coeff.; σ = 1.0



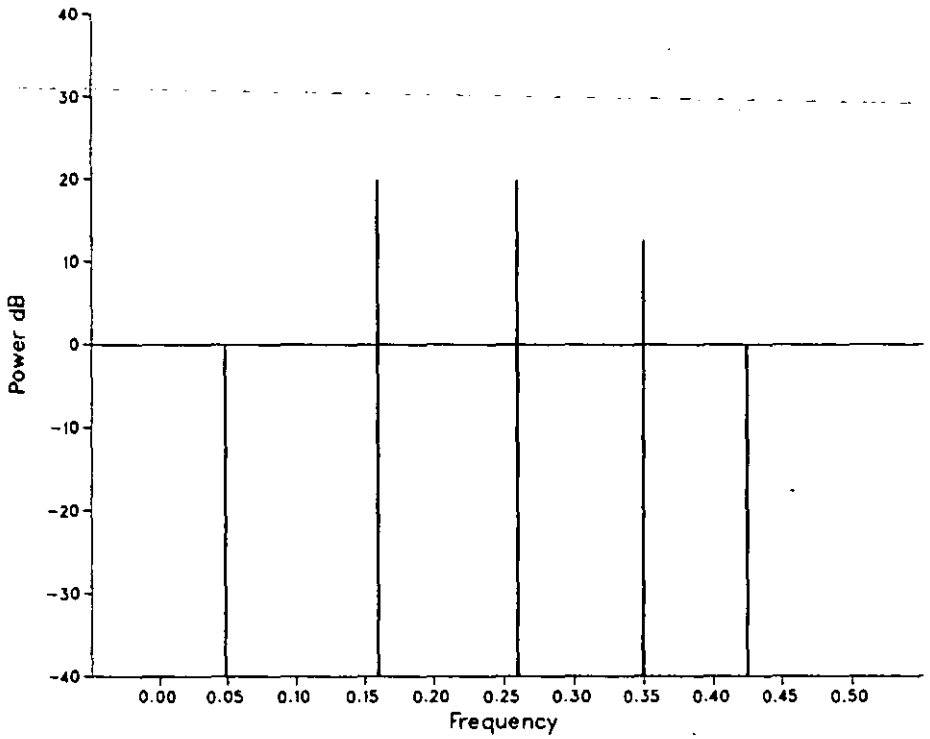
Graph 9.3.4 - Modified FBLS technique, 20 coeff.; σ = 2.0



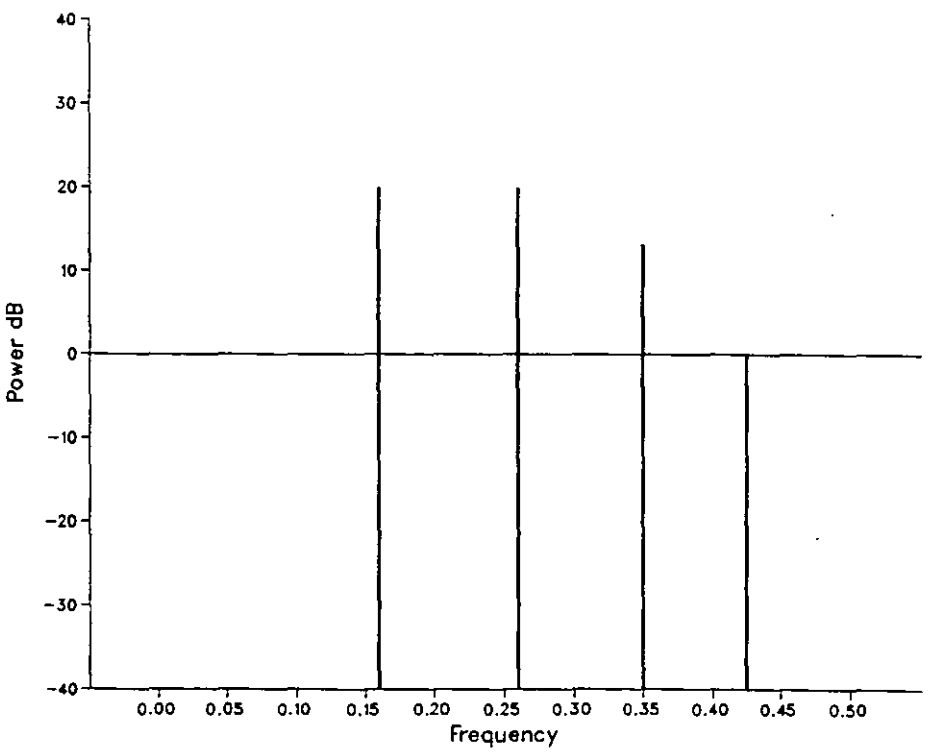
Graph 9.4.1 - Modified FBLS technique, 30 coeff.; σ = 0.25



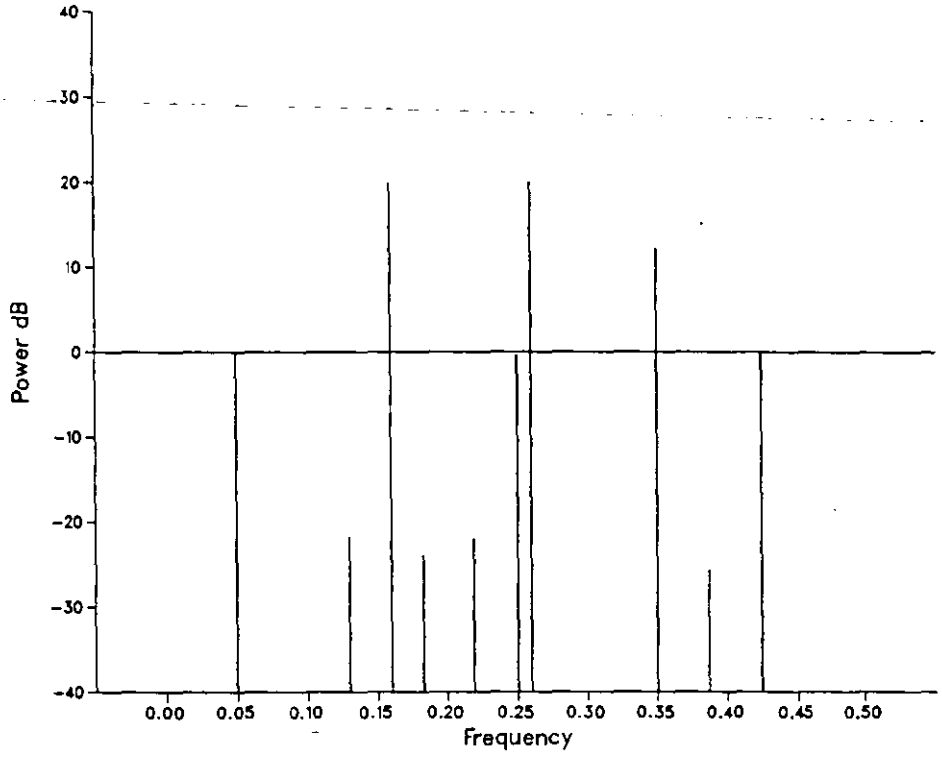
Graph 9.4.2 - Modified FBLS technique, 30 coeff.; σ = 0.5



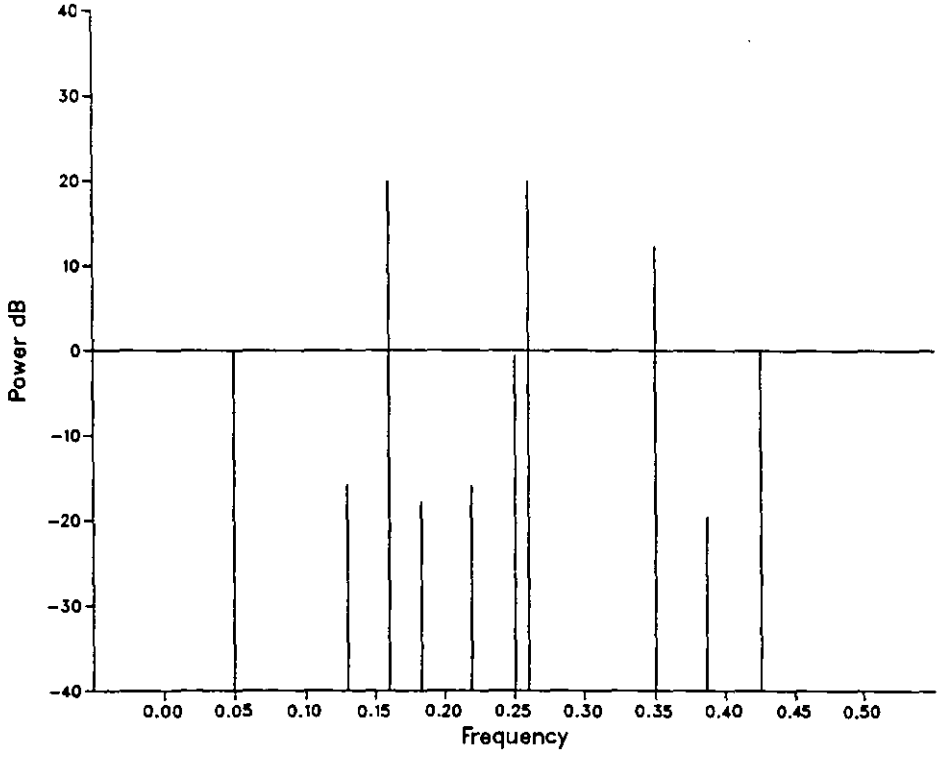
Graph 9.4.3 - Modified FBLS technique, 30 coeff.; σ = 1.0



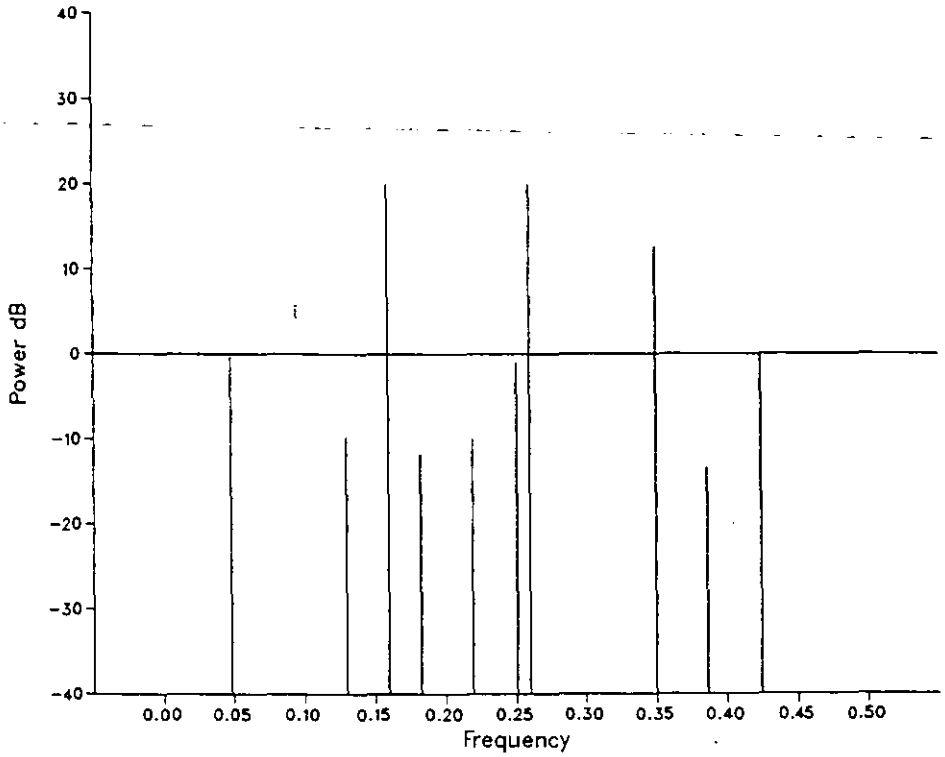
Graph 9.4.4 - Modified FBLS technique, 30 coeff.; σ = 2.0



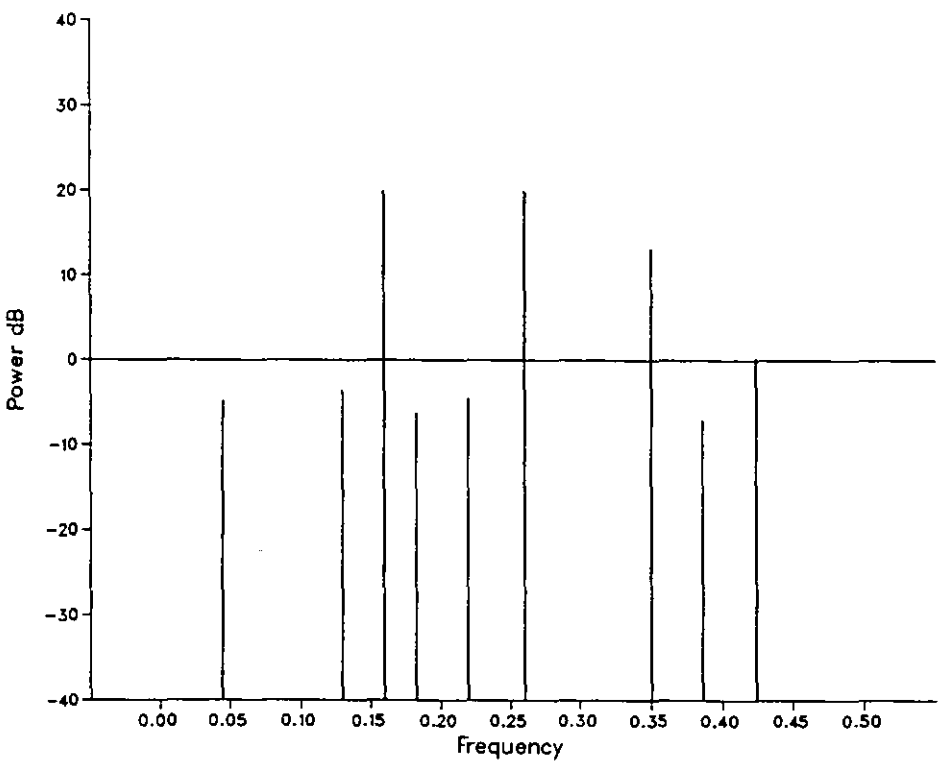
Graph 9.5.1 - Modified FBLS technique, 40 coeff.; σ = 0.25



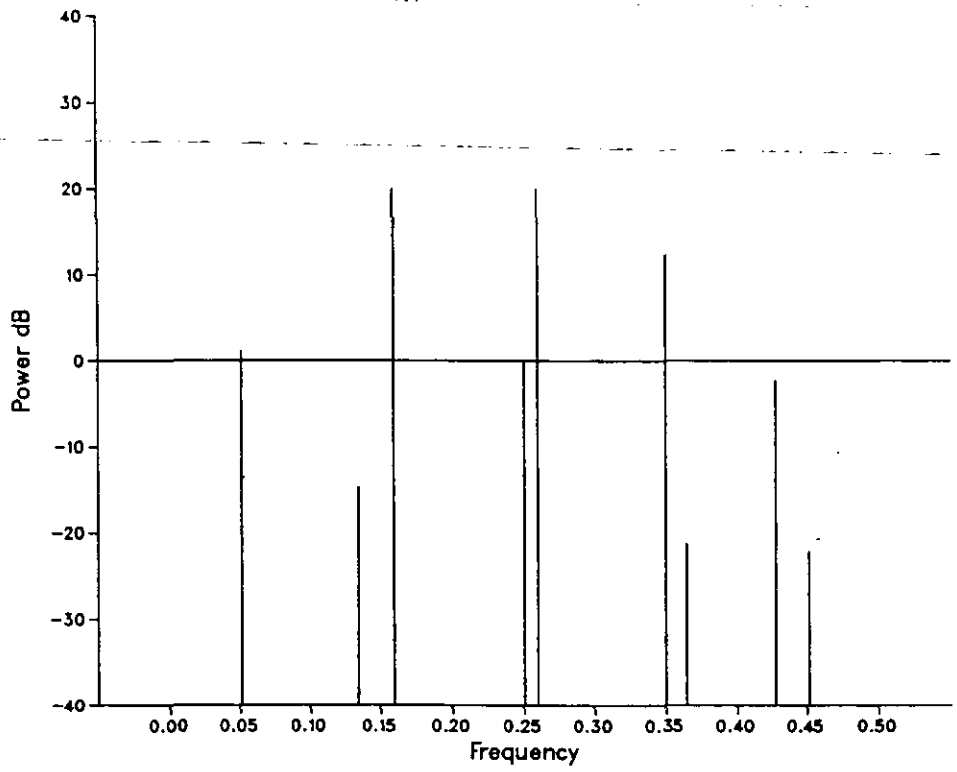
Graph 9.5.2 - Modified FBLS technique, 40 coeff.; σ = 0.5



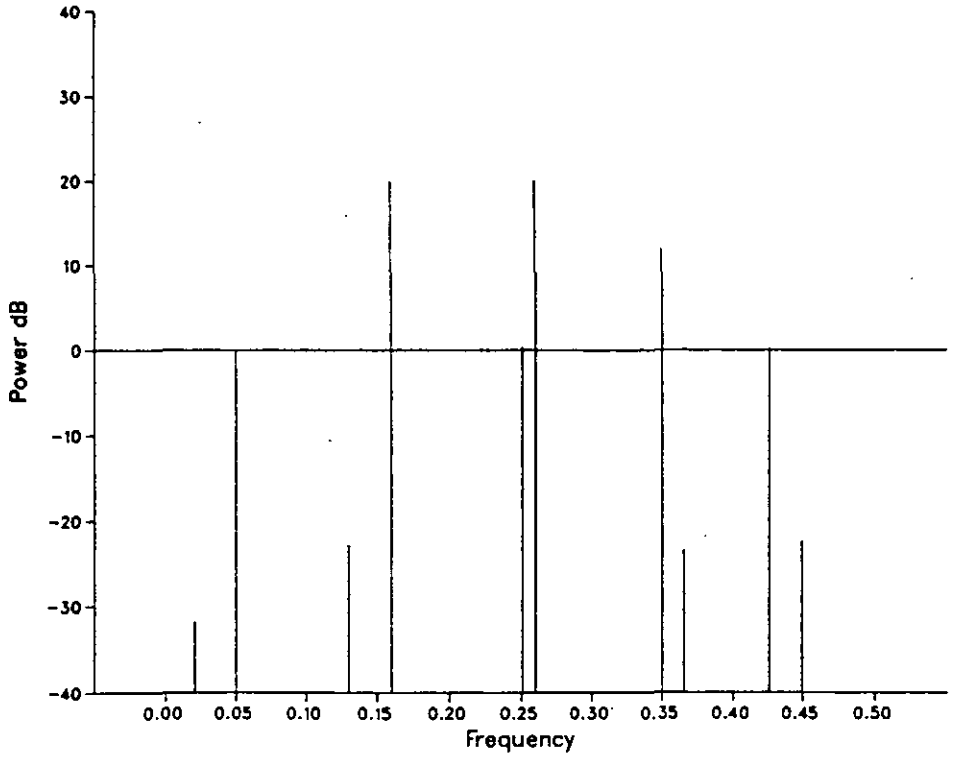
Graph 9.5.3 - Modified FBLS technique, 40 coeff.; $\sigma = 1.0$



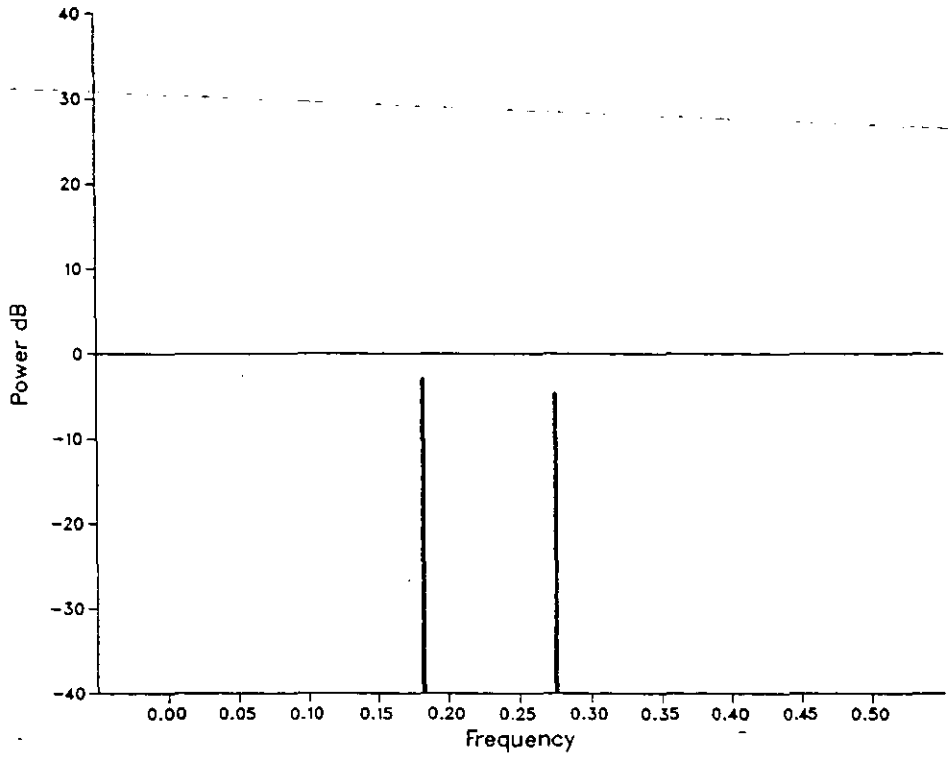
Graph 9.5.4 - Modified FBLS technique, 40 coeff.; $\sigma = 2.0$



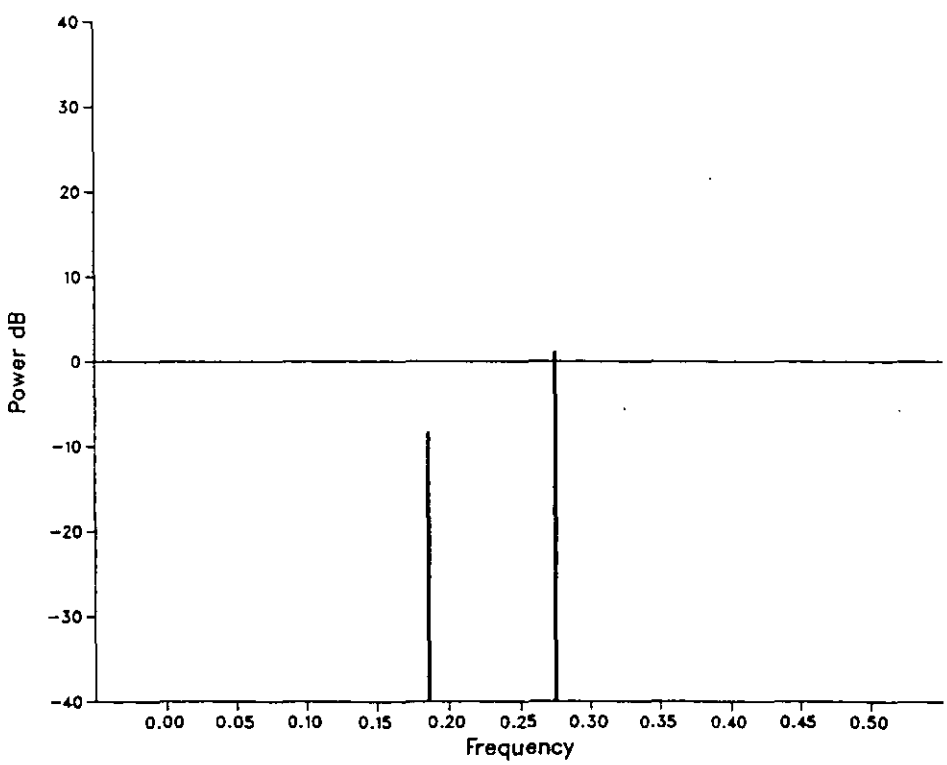
Graph 9.6.1 - Modified PSLE method, 20 coeff. selected; σ = 0.25



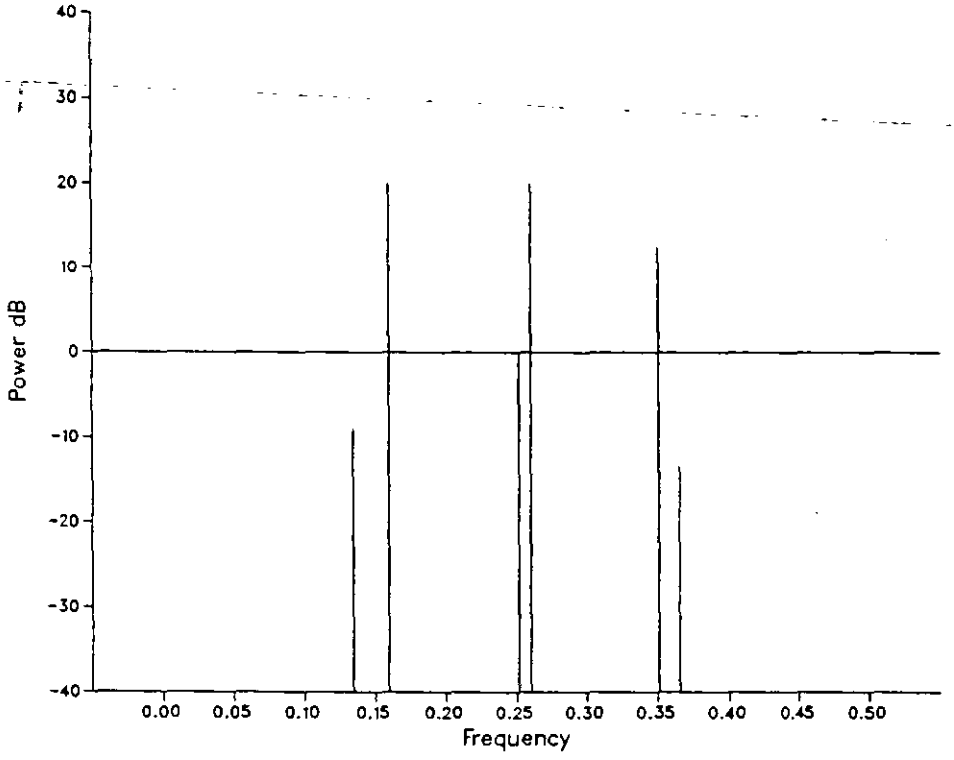
Graph 9.6.2 - Modified PSLE method, 20 coeff. selected; σ = 0.5



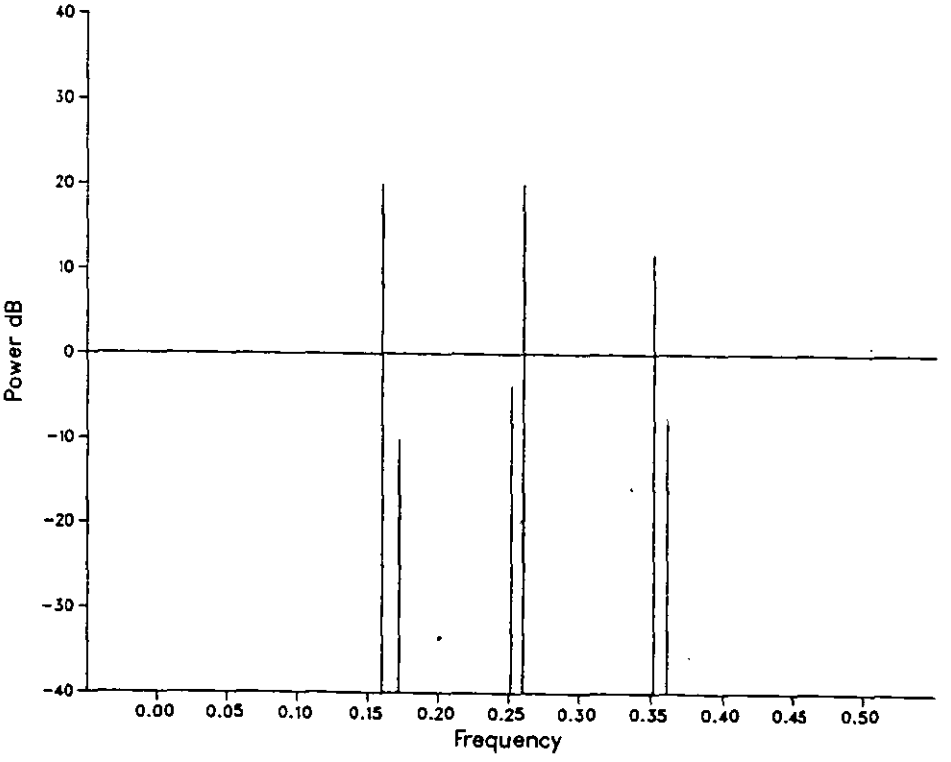
Graph 9.6.3 - Modified PSLE method, 4 coeff. selected; $\sigma = 1.0$



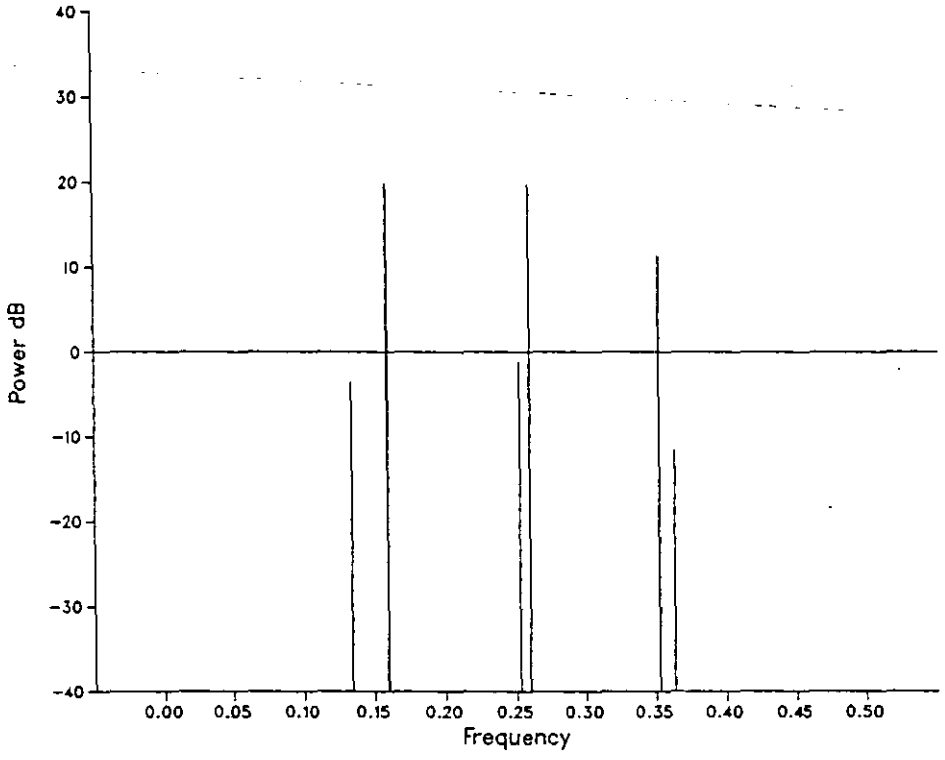
Graph 9.6.4 - Modified PSLE method, 4 coeff. selected; $\sigma = 2.0$



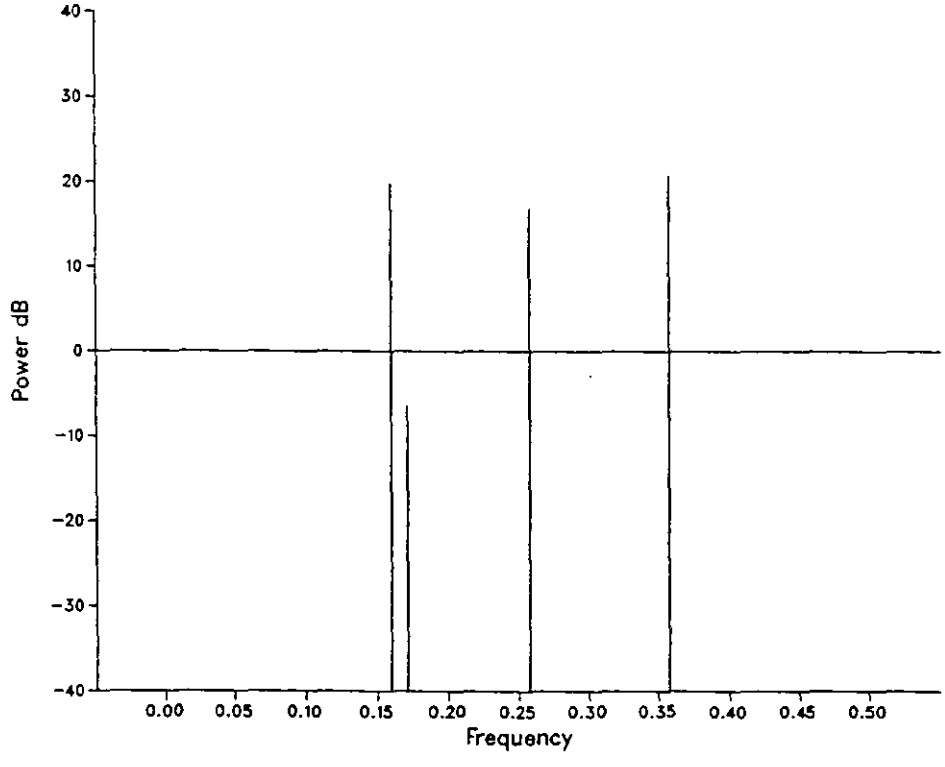
Graph 9.7.1 - Modified PSLE method, 20 coeff. forced; $\sigma = 1.0$



Graph 9.7.2 - Modified PSLE method, 40 coeff. forced; $\sigma = 1.0$



Graph 9.8.1 - Modified PSLE method, 20 coeff. forced; σ = 2.0



Graph 9.8.2 - Modified PSLE method, 40 coeff. forced; σ = 2.0

Roots		Root		Root Frequency	
Real	Imaginary	Amplitude		Fraction of Fs	
0.9505	-0.3055	0.9984	----	0.0495	
0.9505	0.3055	0.9984	----	0.0495	
0.5361	-0.8440	0.9999	----	0.1599	
0.5361	0.8440	0.9999	----	0.1599	
0.0060	-0.9790	0.9790	----	0.2490	
0.0060	0.9790	0.9790	----	0.2490	Sinu-
-0.0632	-0.9975	0.9995	----	0.2601	soids
-0.0632	0.9975	0.9995	----	0.2601	
-0.5877	-0.8090	0.9999	----	0.3500	
-0.5877	0.8090	0.9999	----	0.3500	
-0.8896	-0.4560	0.9996	----	0.4246	
-0.8896	0.4560	0.9996	----	0.4246	
0.9352	-0.1061	0.9412	----	0.0180	
0.9352	0.1061	0.9412	----	0.0180	
0.8272	-0.5037	0.9685	----	0.0871	
0.8272	0.5037	0.9685	----	0.0871	
0.6830	-0.6787	0.9628	----	0.1245	
0.6830	0.6787	0.9628	----	0.1245	
0.3220	-0.8678	0.9256	----	0.1935	Noise
0.3220	0.8678	0.9256	----	0.1935	
-0.0494	-0.5794	0.5815	----	0.2635	
-0.0494	0.5794	0.5815	----	0.2635	
-0.3772	-0.8710	0.9492	----	0.3150	
-0.7273	-0.6312	0.9639	----	0.3862	
-0.7273	0.6312	0.9639	----	0.3862	
-0.9391	-0.2050	0.9612	----	0.4658	
-0.9391	0.2050	0.9612	----	0.4658	
-0.9719	0.0000	0.9718	----	0.5000	
-0.0677	0.0000	0.0677	----	0.5000	

Roots generated by the FBLS technique using 30 AR coefficients and simulated data (σ = 0.5).

Table 9.1

Frequency (Fs)	Amplitude Basic Algorithm	Put all roots on unit circle	Only roots close to unity	Both Modifications
0.0495	1.0886	0.9923	1.1106	0.9997
0.1599	10.0780	9.9642	10.0462	9.9558
0.2490	1.9186	0.8995	1.5442	0.8736
0.2601	10.4230	10.1741	10.3377	10.0031
0.3500	4.1202	4.1278	4.1682	4.1292
0.4246	1.0365	0.9992	1.0282	0.9992
0.0180	0.1109	0.1414		
0.0871	0.1865	0.1654		
0.1245	0.4252	0.0689		
0.1934	0.6676	0.0398		
0.2636	7.2764	0.3437		
0.3150	0.4507	0.0945		
0.3862	0.3841	0.1071		
0.4658	0.3881	0.1005		

Power estimates using roots generated by the FBLS technique. 30 AR coefficients, simulated data (σ = 0.5).

Table 9.2

20 coefficients Frequency Fs /Amplitude		30 coefficients Frequency Fs /Amplitude		40 coefficients Frequency Fs /Amplitude	
0.0499	0.9949	0.0495	0.9997	0.0495	0.9918
0.1600	9.9625	0.1600	9.9558	0.1600	9.9525
		0.2490	0.8737	0.2504	0.9239
0.2601	9.8970	0.2601	10.0031	0.2601	10.0014
0.3500	4.1283	0.3500	4.1292	0.3500	4.1327
0.4245	1.0030	0.4246	0.9992	0.4246	0.9961
				0.1301	0.1613
				0.1829	0.1278
				0.2192	0.1593
				0.3867	0.1051

Frequency and power estimates using the FBLS technique. Simulated data (σ = 0.5).

Table 9.3

Frequency -frac of Fs	Amplitude peak
0.0442	15.8445
0.0442	16.0161
0.1591	6.9255
0.1617	3.3823
0.2511	0.9218
0.2601	9.9370
0.3473	0.6002
0.3504	3.6790

Frequency and amplitude estimates obtained from the PSLE method - all roots not on units circle discarded. 30 AR coefficients (14 discarded), σ = 0.5.

Table 9.4

σ = 0.25		σ = 0.5		σ = 1.0		σ = 2.0	
Freq. (Fs)	Ampl. peak	Freq. (Fs)	Ampl. peak	Freq. (Fs)	Ampl. peak	Freq. (Fs)	Ampl. peak
0.0500	1.0030	0.5156	0.2621	0.1827	0.7132	0.1857	0.3822
0.1600	9.9731	0.5156	1.1346	0.2755	0.5892	0.2749	1.1335
0.2504	0.9784	0.1600	9.9408				
0.2601	9.9957	0.2508	0.9723				
0.3501	4.0727	0.2602	9.9771				
0.4254	0.9827	0.3502	4.1573				
		0.4275	0.7656				
0.0532	0.0273						
0.1345	0.0898	0.1346	0.1839				
0.3647	0.0180	0.3651	0.0867				
0.4536	0.0258	0.4520	0.0786				

Frequency and amplitude estimates generated by the modified PSLE method; Simulated noise signal.

Table 9.5

20 coefficients		40 coefficients		$\sigma = 1.0$
Frequency frac Fs	Amplitude peak	Frequency frac Fs	Amplitude peak	
0.1346	0.3512	0.1598	9.8862	
0.1599	9.9120	0.1718	0.3108	
0.2517	0.9985	0.2512	0.6463	
0.2602	9.9457	0.2598	9.8873	
0.3508	4.2242	0.3516	3.9191	
0.3653	0.2130	0.3612	0.4227	

20 coefficients		40 coefficients		$\sigma = 2.0$
Frequency frac Fs	Amplitude peak	Frequency frac Fs	Amplitude peak	
0.1344	0.6669	0.1598	9.7819	
0.1596	9.8401	0.1710	0.4704	
0.2527	0.8841	0.2579	6.9527	
0.2602	9.8114	0.2579	5.2779	
0.3527	3.6927	0.3575	10.9008	
0.3635	0.2682	0.3575	10.6770	

Frequency and amplitude estimates obtained via the PSLE technique - all roots not on units circle discarded.

Table 9.6

9.6 Application to real aero engine vibration data.

9.6.1 Source of data.

The raw vibration signal from which the vibration data sequence has been digitised, originates from the conditioned output of an accelerometer mounted on an aero engine fan cowling. The conditioning consists of a charge amplifier (to convert charge to voltage) and an integrator to convert from acceleration to velocity (inches/sec). The aero engine to which the transducer had been mounted was an RB211-535E4, this is a three shaft engine capable of delivering 40,000 lb thrust. The vibration levels on such an engine are determined by measuring the amplitudes of the sinusoidal velocity components at the three shaft rotation frequencies. Note that the three shafts are gas coupled, i.e. there is no gearbox, and hence their rotational speeds do not have a fixed relationship.

The data sequence consists of 1024 samples (12-bit 2's complement), sampled at 512 Hz. The signal has also been filtered with an elliptical low-pass filter having a break-point at 200 Hz, a roll-off of 135 dB/octave and a stop-band attenuation of 80 dB.

A second signal has also been sampled and digitised, this is a calibration signal representing a velocity of 2.0 inches/sec at 120 Hz. The measurement and recording of this signal is essential to convert the accelerometer output into engineering units.

9.6.2 Engine terminology.

In the following analysis the three shafts of the RB211 engine will be refered to as the LP, IP and HP shafts (low, intermediate and high pressure). Due to the blade dimensions on each shaft the LP shaft always rotates the slowest and the

HP shaft the fastest. From ground idle to flight take off the shaft speeds cover the following approximate frequency ranges : LP - 15 to 82 Hz, IP - 39 to 128 Hz and HP - 106 to 195 Hz. Off-line vibration analysis is almost always performed via a 1024 point DFT, as shown in chapter 6.

9.6.3 Fourier transform.

Discrete Fourier transforms (1024 points) of the calibration and vibration signals are shown in graphs 9.9 and 9.10 respectively. In graph 9.10 the three sinusoidal vibration components related to each of the shafts can clearly be identified at :- LP - 0.1553 Fs [79.5 Hz], IP - 0.2217 Fs [113.5 Hz] and HP - 0.3301 Fs [169.0 Hz]. It should be remembered that the DFT frequency bins are spaced at 0.00097 Fs [0.5 Hz], thus the estimated frequency of a component is only accurate to $\pm$0.00048 Fs [$\pm$0.25 Hz]. The frequencies and amplitudes of the three components, together with some of the smaller components in the spectrum are tabulated in table 9.7, note that the amplitude has been converted to engineering units using the amplitude of the 120 Hz signal shown in graph 9.9.

A 128 point transform using the central 128 points of the above 1024 transform is shown in graph 9.11. The lack of resolution and increased effect of noise compared to the 1024 point transform is quite evident. The noise floor is approximately 20 dB below the three main vibration components, which is 9 dB higher than for the 1024 point DFT.

9.6.4 Modified FBLS technique.

Spectra produced by the modified FBLS technique using 20, 30 and 40 AR filter coefficients are shown in graphs 9.12.1 to 9.12.3 and tabulated in table 9.8, the amplitudes again being converted to engineering units in the table. It can be seen

that this technique has worked extrememly well, clearly picking out the vibration components in all three cases of AR filter length. It should be pointed out that the calibration signal only required 2 AR filter coefficients (the "ideal" number) to accurately extract the amplitude of the 120 Hz. component, as there is very little noise present in this signal (as seen in graph 9.9). Due to its simplicity this spectrum is not shown, the size of the calibration signal using this technique is 559.6 units at 120.9 Hz.

As would be expected, as the number of coefficients is increased then the number of smaller components within the spectrum also increases. Not all of these components are noise, in fact if these spectra are compared to the DFT spectrum then it becomes apparent that most of these are genuine small vibration components or medium "Q" colourations in the noise. An important point being that the amplitudes of all the smaller components are reasonable in size and generally correspond in amplitude to those seen in the 1024 point DFT.

Refering back to the analysis of this technique on the simulated data sequences, two general rules were derived as to how many AR filter coefficients should be used with any particular set of data. These rules depend upon number of sinusoids and noise level within the data sequence; For the vibration data sequence the number of sinusoids is approximately seven and the noise level is approximately 10dB lower than the smallest of these seven sinusoidal components (using information in the 1024 point DFT). Thus the AR filters should use approximately 28 coefficients [(7*2)*2], as defined in the first of the two general rules. As can be seen from the results in graph 9.12.2 and table 9.8, 30 AR coefficients has indeed extracted all three vibration components with no less accuracy than for 40 coefficients, but with more accuracy than for 20 coefficients. Note also that in the 30 coefficient case it has extracted exactly seven sinusoids - as predicted.

All of the above results have come from the modified FBLS technique applied to the central 128 samples of the 1024 point vibration data sample block. Using the central data allows the results to be compared with the results obtained from the DFT analysis where windowing tends to make the frequency and amplitude results more dependent upon the central data. However as only 128 samples have been used, the analysis can also be applied to other areas within the 1024 point block. The two resulting spectra obtained by applying this technique to the outer 128 data samples are shown in graphs 9.13.1 and 9.13.2. It can be seen that the LP vibration increases significantly between these two sample blocks, the increase being from 0.382 to 0.472 inches/sec (i.e. 24%). This example demonstrates the advantage of using small sample blocks which reduce the smearing or time averaging of the frequency and amplitude information within the sampled data.

At the point during which this vibration data was sampled and digitised the engine speed was static, thus the change in LP vibration amplitude seen above would indicate a beating vibration component.

9.6.5 Modified PLSE technique.

The spectrum produced by the modified PLSE technique applied to the vibration data is shown in graph 9.14 and tabulated in table 9.9. The calibration spectrum again is not shown because of its simplicity, the actual calibration signal amplitude being 559.6 units at 120.9 Hz, exactly the same as for the FBLS case. This analysis has resulted in the algorithm selecting an AR filter of 14 coefficients in length, and has clearly picked out the vibration components. As with the FBLS case this technique has also been applied to the two outer 128 sample blocks, these spectra are shown in graphs 9.15.1 and 9.15.2. The spectrum produced for the first sample block again uses 14 AR coefficients and gives

very reasonable results, however for the second sample block this technique has only been able to use 12 AR coefficients causing significant errors in the amplitude estimates.

The poor result shown in graph 9.15.2 is not surprising if the signal to noise ratio of the vibration data is taken into consideration. Remembering that when this technique was applied to the simulated data it was found that it began to have problems when the noise power and the sinusoid powers are approximately equal. It has already been pointed out that the noise floor of the vibration data (as seen in the 128 point DFT) is approximately 20 dB below the vibration components. This means that the total noise power is only 2 dB below the smallest vibration component, and 8 dB higher than the other components.

The above results show that the modified PSLE technique can be used successfully with real and noisy engine data but that its noise tolerance is rather close to the noise level seen within vibration data (at least for the data used for this assessment).

9.7 Comparison between modified techniques.

The analysis of the DFT, the modified FBLS technique and the modified PSLE technique applied to simulated sinusoids in noise, and to vibration data from an accelerometer, have provided a fairly clear picture of their limitations and strengths.

In all of the tests the DFT has provided predictable results and noise has had no significant ill effects upon the algorithm, it merely being distributed, although not necessarily evenly, across the spectrum. The amplitude estimation is a problem (due to the picket fence effect) but can be alleviated by calculating the total energy within a peak as shown in chapter 3. The resolution within the

spectrum is only a serious problem when the sample blocks are small. Long sample blocks do however average out transient changes.

The amplitudes of the vibration components, as calculated by the DFT, should be quite accurate as each component lies almost exactly on a filter bin (purely by coincidence). The amplitude results obtained by the modified FBLS technique certainly back this up for the IP and HP vibration components which are almost identical. This of course is also a good result for the FBLS technique because the DFT has shown very good power estimation when sinusoids have been coincident with filter bins. The LP vibration amplitude does not match up very well between the two techniques, but the FBLS technique has already shown that between the first and last data samples there is a significant change in the amplitude of this component, and is almost certainly the reason for the different results. The consequence of components lying in between the filter bins can be seen from the results of the small component at 0.31 Fs, the amplitude of this component given by the DFT is at least 14% lower than that given by the FBLS technique.

The modified PSLE technique frequency and amplitude results from the vibration data do not appear to be as good as those from the FBLS technique or even the DFT, and it certainly appears to be on its operating limit with the vibration signal.

When using the DFT there is no parameter selection to make before the analysis (except type of window), however the other techniques both require selection of their AR filter length (i.e. order selection). The modified FBLS technique is much more flexible than the PSLE technique in the number of AR filter coefficients which it can cope with. However some decision as to the filter order must still be made. No satisfactory algorithm (e.g. Akaike criterion) could be found to do this, although some basic rules have been

derived, which under limited testing, seem to provide reasonable guide lines. Also, as demonstrated earlier, the problem of precise order selection with the modified FBLS technique is not too much of a problem as this technique is quite tolerant to overdetermined AR filters, exhibiting no ill effects such as frequency splitting. Thus, if the approximate filter order is not known, then it is quite reasonable to use many more coefficients than is perhaps necessary. All that will happen is that noise components will also be extracted, as always occurs anyway with the DFT.

The noise tolerance and stability of the modified FBLS technique has proved to be very good, matching that of the DFT. The FBLS program which has produced all of the above results is limited to 40 AR filter coefficients (due to memory limitations - the NAG library polynomial rooting and complex matrix inversion uses massive amounts of memory). If this program had allowed larger arrays this technique may well have been able to extract sinusoids even deeper in noise.

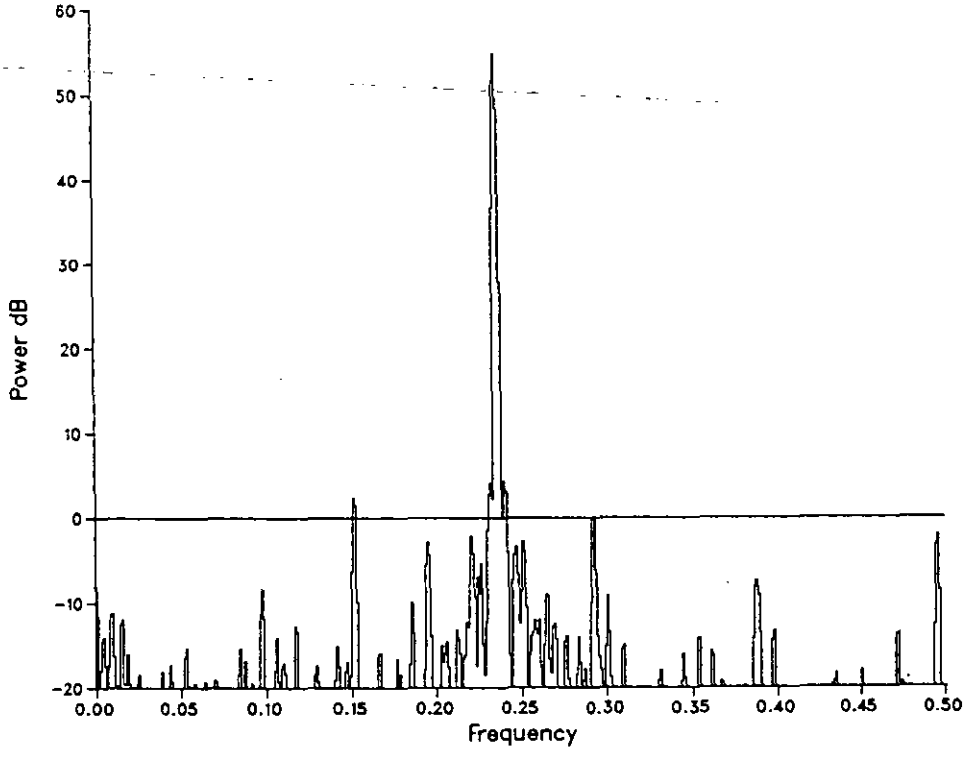
The modified FBLS and PSLE techniques have clearly demonstrated their superior ability to estimate frequency and amplitude over the DFT. However noise does affect both of these techniques, in the FBLS case it simply causes it to extract less components for a given number of AR coefficients, whereas in the PSLE case it eventually prohibits the algorithm from functioning properly. This latter problem is considerable and seriously limits its use with noisy signals.

9.8 Summary.

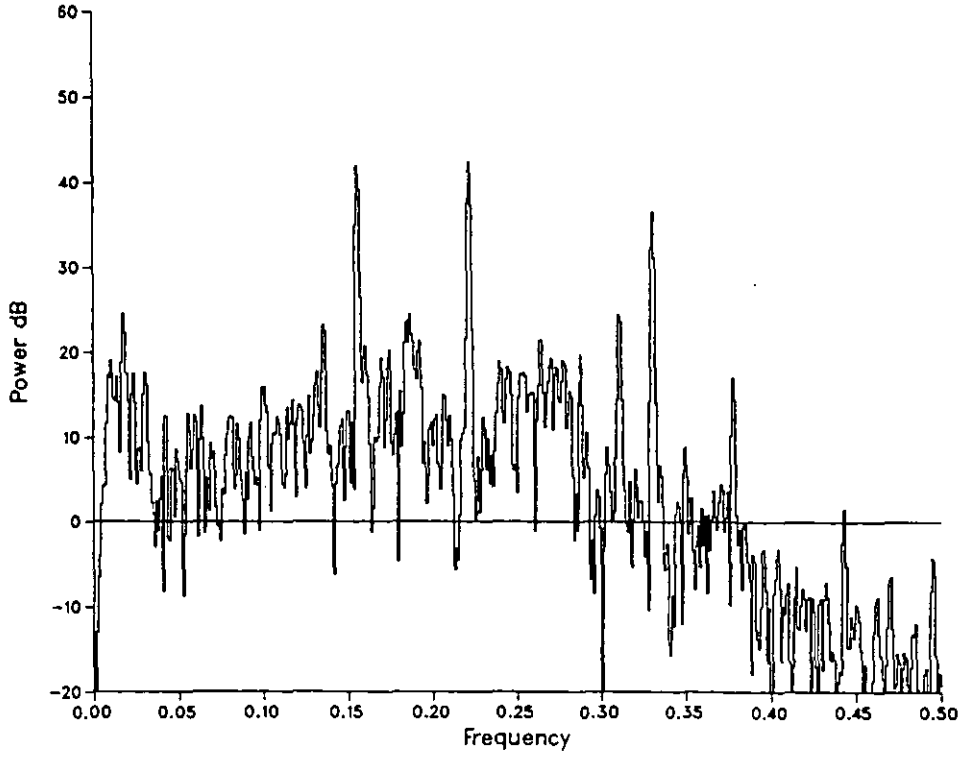
In the last three chapters an assessment has been made of the performance of some modern spectrum analysis techniques. The forward-backward least squares technique and the Prony spectral line estimation technique have shown the most

optimistic results, and subsequent modifications have been tried to further improve their performance. These have been further tested with a simulated noisy signal and with a real aero engine signal.

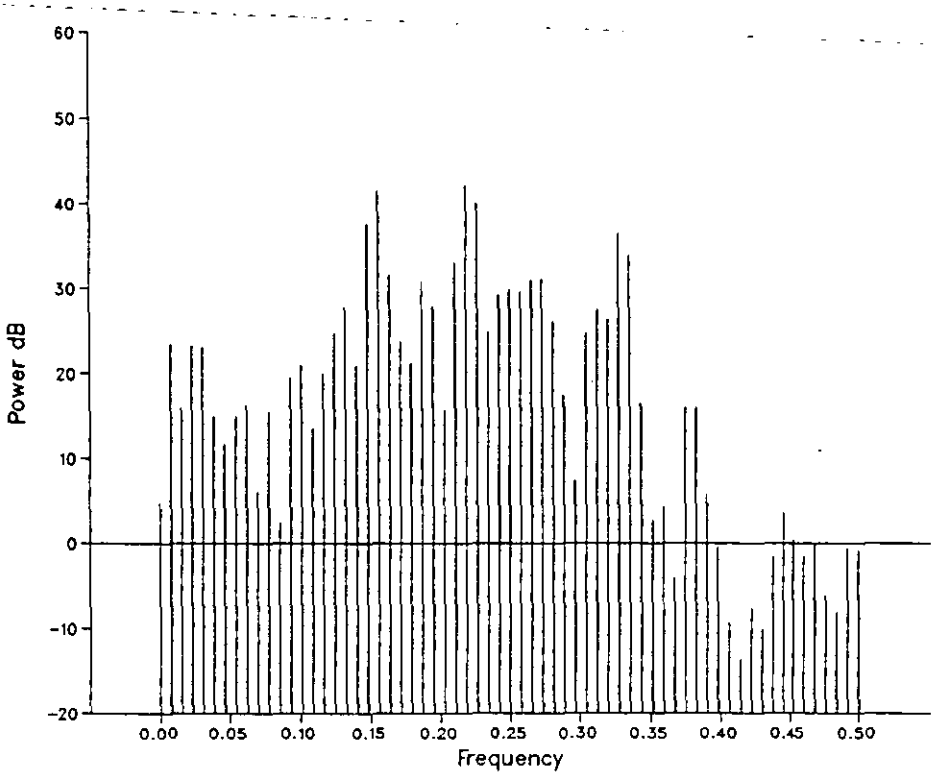
It has been shown that the modified PSLE technique can perform very successfully but has restrictive limitations when dealing with signals in significant amounts of noise. The modified FBLS technique has been very successful and has proved to be the more flexible and tolerant of the two techniques especially when dealing with signals in considerable amounts of noise. Its frequency and power estimations have also been shown to be as good or better than those of the DFT, using data blocks eight times smaller.



Graph 9.9 - DFT, 1024 points; Vibration calibration signal



Graph 9.10 - DFT, 1024 points; Aero engine vibration signal

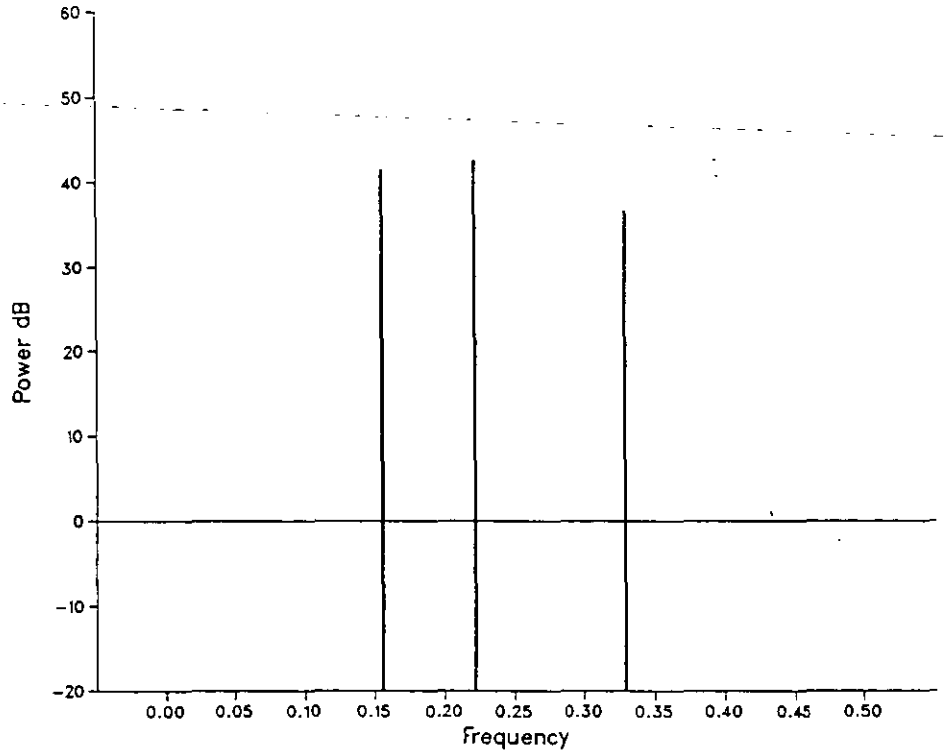


Graph 9.11 - DFT, 128 points; Aero engine vibration signal

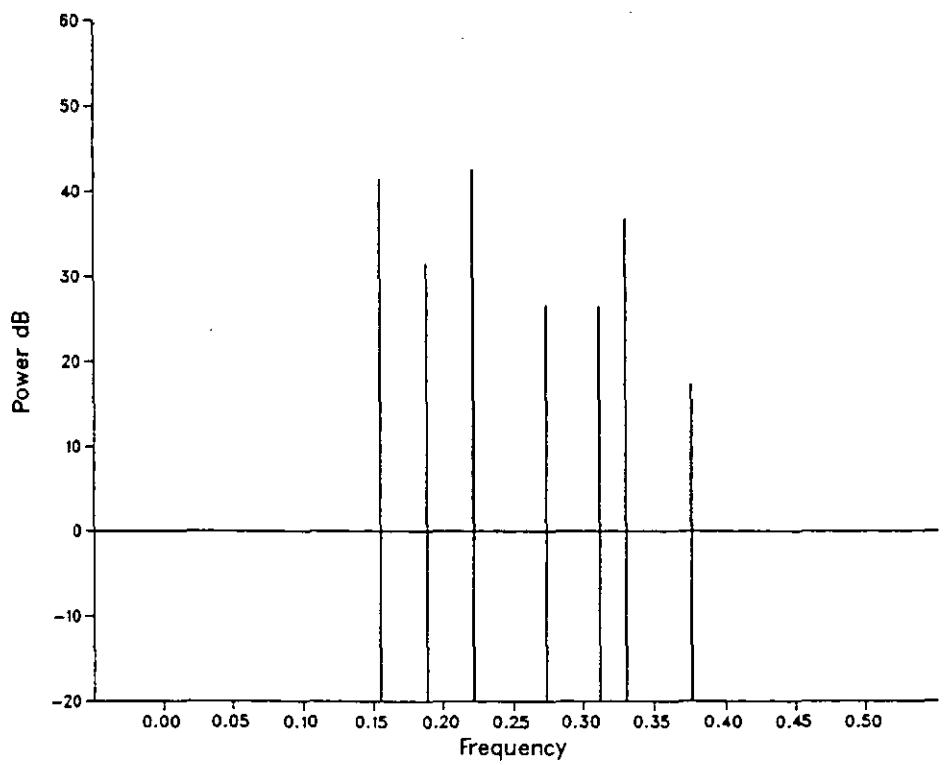
Frequency		Amplitude
frac. of Fs	hertz	in/sec peak
0.0176	19.0	0.061
0.1357	69.5	0.052
0.1553	79.5	0.454
0.1865	95.5	0.061
0.2217	113.5	0.478
0.3105	159.0	0.060
0.3301	169.0	0.245

Frequency and amplitude estimates of main components in the 1024 point DFT of the vibration signal.

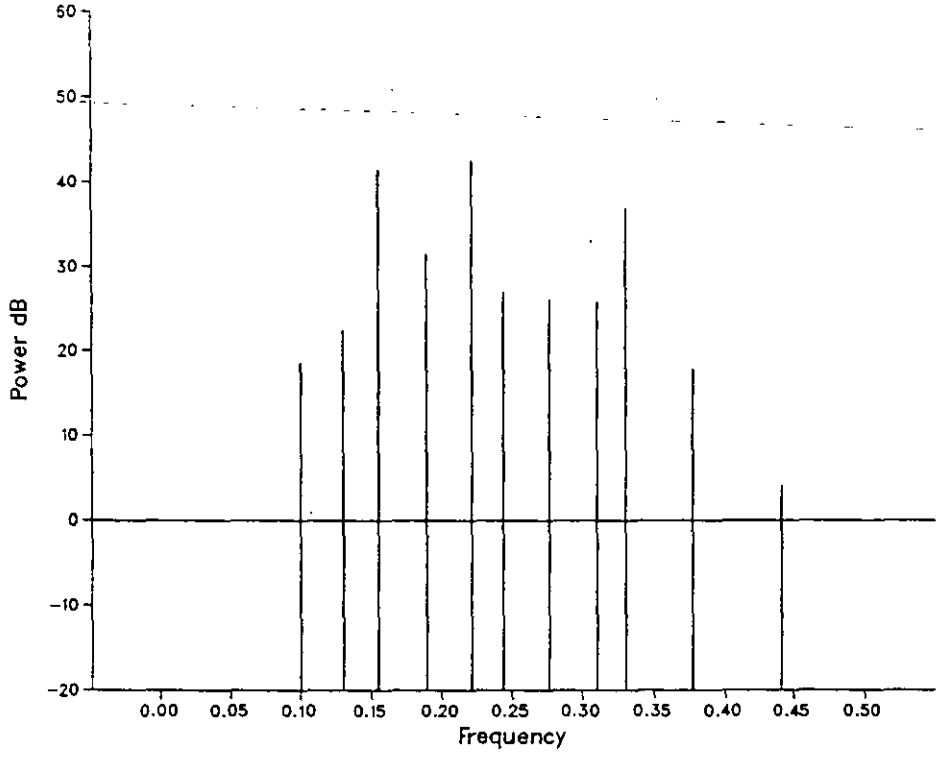
Table 9.7



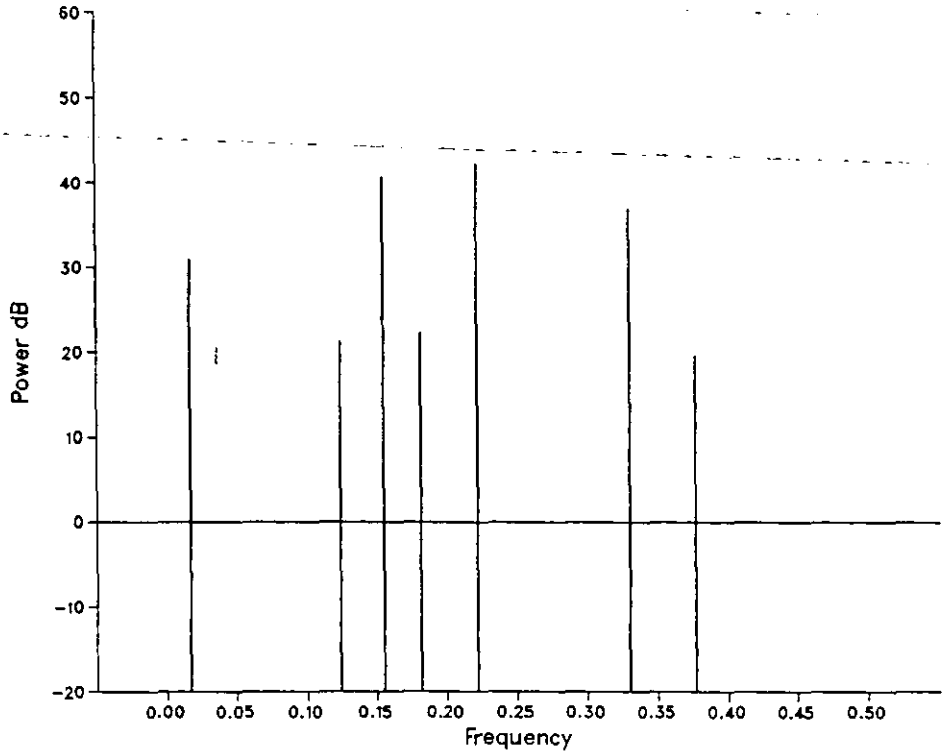
Graph 9.12.1 - Modified FBLS tech, 20 coeff; Vibration signal



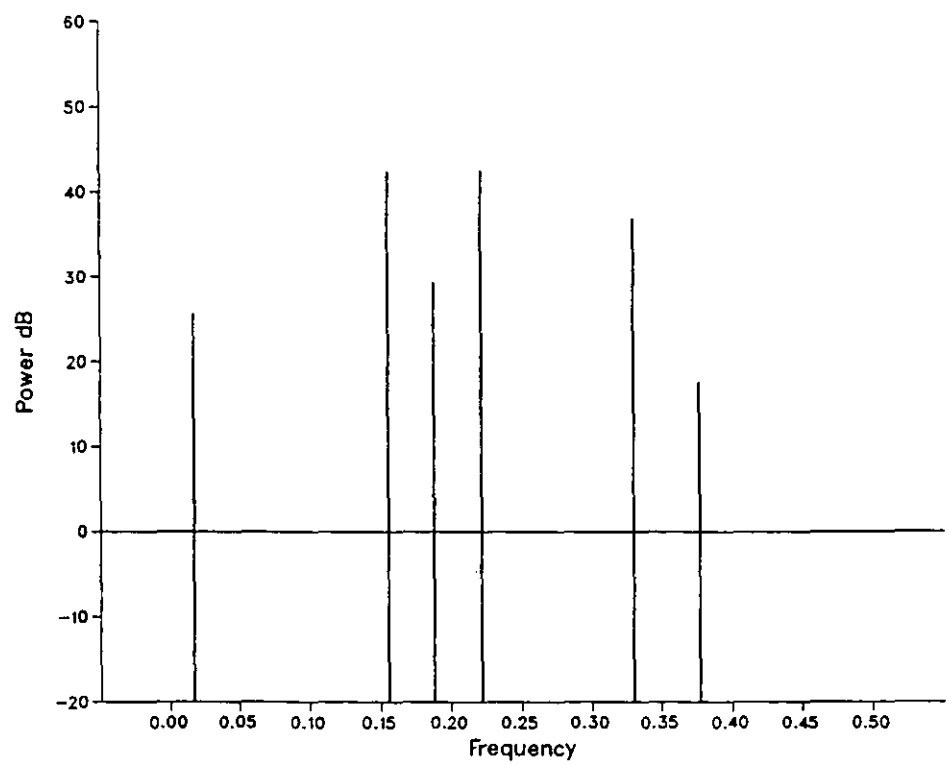
Graph 9.12.2 - Modified FBLS tech, 30 coeff; Vibration signal



Graph 9.12.3 - Modified FBLS tech, 40 coeff; Vibration signal



Graph 9.13.1 - Modified FBLS tech, 30 coeff; Vibration signal 448 samples before graph 9.12.2



Graph 9.13.2 - Modified FBLS tech, 30 coeff; Vibration signal 448 samples after graph 9.12.2

Frequency		Amplitude
frac. of Fs	hertz	in/sec peak
0.1555	79.61	0.422
0.2218	113.56	0.479
0.3292	168.55	0.242

20 AR filter coefficients

Frequency		Amplitude
frac. of Fs	hertz	in/sec peak
0.1551	79.41	0.422
0.1886	96.56	0.134
0.2217	113.51	0.479
0.2723	139.42	0.077
0.3117	159.59	0.076
0.3302	169.06	0.250
0.3768	192.92	0.026

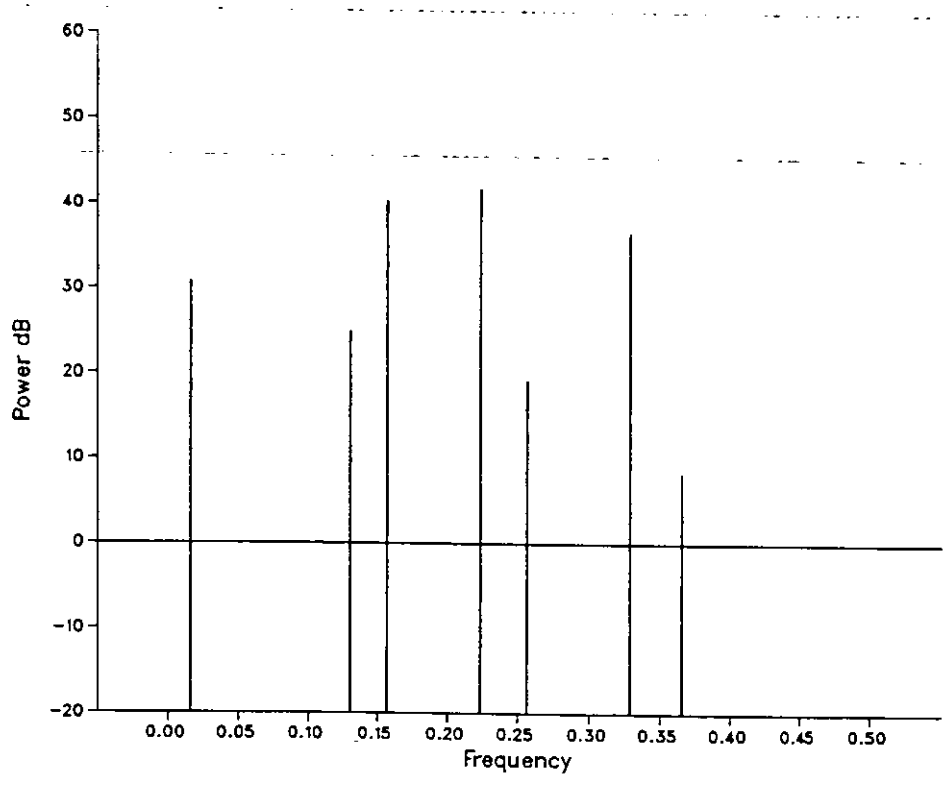
30 AR filter coefficients

Frequency		Amplitude
frac. of Fs	hertz	in/sec peak
0.1003	51.35	0.030
0.1296	66.36	0.047
0.1553	79.51	0.420
0.1894	96.97	0.134
0.2216	113.50	0.479
0.2440	124.93	0.079
0.2763	140.08	0.071
0.3105	158.98	0.070
0.3302	169.06	0.250
0.3776	193.33	0.028
0.4418	226.20	0.005

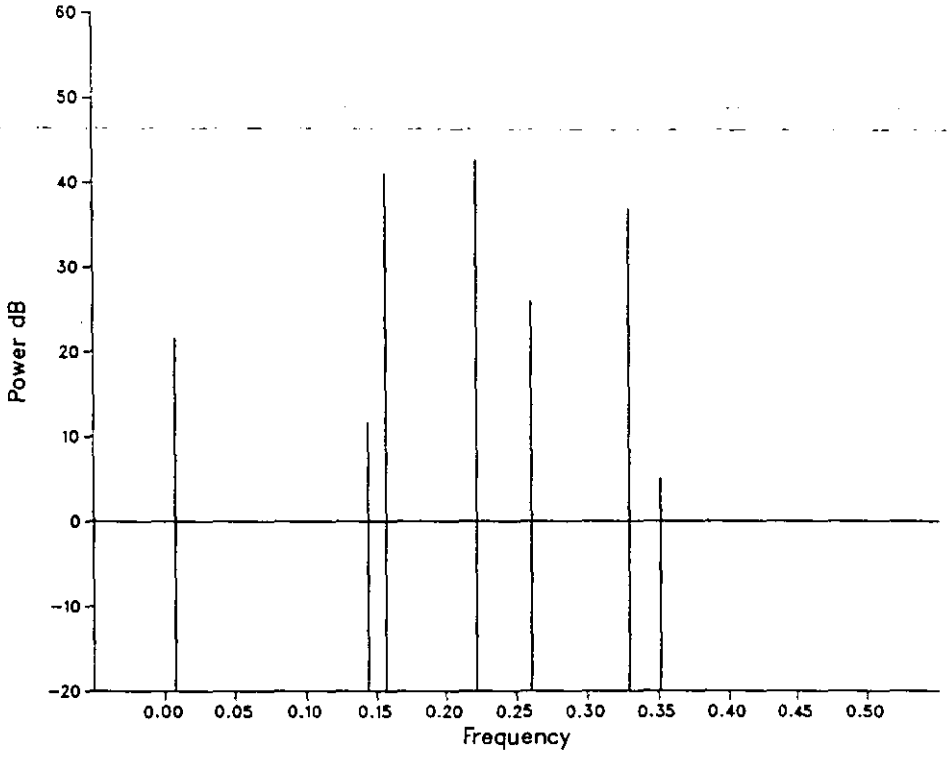
40 AR filter coefficients

Frequency and amplitude estimates obtained via the modified FBLS technique on the vibration signal.

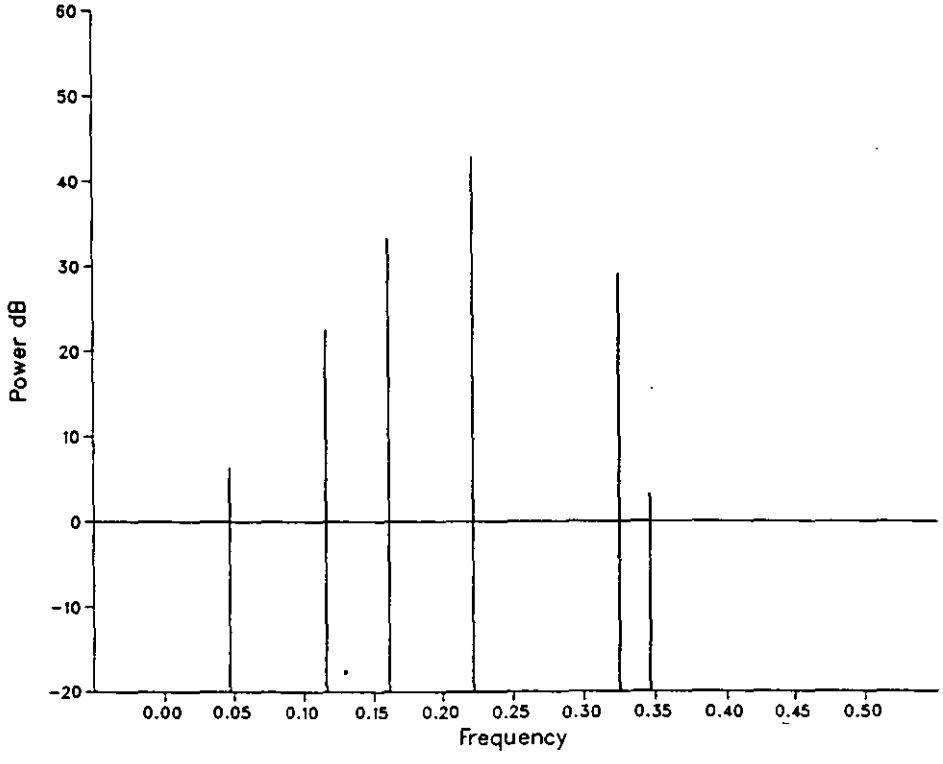
Table 9.8



Graph 9.14 - Modified PSLE method, 14 coefficients selected; Vibration signal



Graph 9.15.1 - Modified PSLE method, 14 coefficients selected 448 samples before 9.14



Graph 9.15.2 - Modified PSLE method, 14 coefficients selected 448 samples before graph 9.14

Frequency		Amplitude
frac. of Fs	hertz	in/sec peak
0.0073	3.73	0.043
0.1441	73.78	0.014
0.1567	80.23	0.400
0.2217	113.51	0.477
0.2607	133.47	0.070
0.3294	168.65	0.244
0.3514	179.92	0.006

Frequency and amplitude estimates obtained via the modified PSLE method on the vibration signal

Table 9.9

CHAPTER 10

Summary and Conclusions

10.1 Intelligent Dynamic Data Acquisition System.

In chapter one a specification was defined for a system that would perform acquisition and analysis of signals produced by aero engine mounted transducers. After a two year design and development period, a system meeting all of these specifications was finalised in a productionised package called IDDAS. During the development period many demonstrations and lectures were given by the author to potential customers within Rolls Royce. As a result several systems very quickly went into service in the test facilities at Rolls Royce Derby. Systems were also bought by Rolls Royce Leavesden and Rolls Royce Bristol for appraisal and software development. Within ten months of the productionised system being available there were 18 sets (at approximately £2,000 a set) in use at Rolls Royce.

One of the reasons this project was tackled was because there were no systems available in the market capable of meeting the specification. On the completion of IDDAS this situation still had not changed. In an attempt to recoup some of the development costs (and hopefully make some profit) several instrumentation manufacturers were approached with the offer of manufacturing and selling IDDAS under license. This offer was taken up by Data Basix of Newbury. This company has also agreed to sell the "Real Time Monitor" (see chapter 6) under license, and is currently developing an expert system which uses IDDAS to provide spectra at high speed.

Having realised that IDDAS has several unique features, a patent application has been submitted for Europe and the USA, this is currently being persued by the Rolls Royce patents department.

The considerable processing power available in IDDAS should ensure its use in engine analysis for several years to come. So far, no reasonable application has been found which IDDAS can not cope with, in fact in many applications it is an overkill. An example of this is in the "Real Time Monitor" (see chapter 6) in which only four out of the forty spectra available per second are actually used. This is due to the relatively slow graphics operations which the PDP-11 has to perform. The only remaining work involving IDDAS is in extending the software capabilities. One requirement which will inevitably be requested, is a zoom facility, there are no problems in adding this type of facility as there is plenty of data memory left, but due to time limitations has not been written yet.

When IDDAS finally does become too slow, or cannot support further software changes for certain applications, a mark II will have to be designed. There are now many signal processing micro-processors available, including several upgrades on the TMS32010. The most powerful of these is the TMS320C25 which has many hardware and software enhancements over the TMS32010 and runs twice as fast. However this processor may not be the best choice for an upgrade of IDDAS as there are other processors such as the transputer which are at least as powerful. It is not particularly sensible for the author to recommend a successor to the TMS32010 at this stage because faster and more powerful processors will inevitably appear on the market by the time an upgrade is required. However it is worth suggesting that the new processor should perform fully implemented floating point arithmetic on board, and that it should have an optimised complier for a high level language such as "C". Both of these qualities will allow much more complex signal

processing algorithms than the FFT to be performed, and will allow faster and easier development of programs in comparison to programming in assembly code.

Summarising, it is fair to say that the production of IDDAS has been an unqualified success and has met all the hopes of three years previous. There is no doubt that IDDAS has improved the quality of engine testing and already in one case may have saved a very expensive engine from self destruction!.

10.2 Modern spectrum analysis techniques.

An evaluation of some modern spectral estimation techniques has been carried out in order to assess the possibility of using one or more of them for aero engine analysis, Also, to determine what the performance advantages and disadvantages of these techniques are as compared with the FFT.

Using the results obtained by Kay and Marple [27] as a starting point, four techniques were programmed onto the Loughborough main frame computer (using Fortran) and applied to two test signals. The results of these tests showed that Burg's autoregressive method and the Pisarenko harmonic decomposition technique were not very suitable for practical applications. This was because Burg's method exhibited frequency splitting and because they both exhibited frequency biasing, the latter being due to constraints within their respective algorithms. This was not the case however with the forward-backward least squares (FBLS) and the Prony spectral line estimation (PSLE) techniques which both displayed very good frequency estimation. Although the power estimation of the FBLS technique was not very good it was decided to further test these two techniques with more realistic and noisy signals.

Before further testing the FBLS technique an alternative method was developed for estimating power, which is based on a least squares method and involves the roots of the AR filter. This proved to be very successful and enabled accurate estimation of power. An additional modification was added which discards components with non-sinusoidal properties, this also served to improve the power estimation.

The results of applying the FBLS and PSLE techniques to signals consisting of pure sinusoids in gaussian noise provided a clear indiction of their capabilities. In general it was found that Prony's technique could not cope with conditions where the noise power was greater than that of any of the sinusoidal components. In these conditions, spurious components, worsening frequency and power estimates, and frequency splitting occured. It was also found that these effects appear very quickly, i.e. at a particular point the algorithm suddenly changes from producing good estimates to misleading and inaccurate estimates.

The FBLS technique provided somewhat different results. The effects of noise were much less marked and only gradually deteriorated the frequency and power estimates. Although it was also found that as noise increases, similar accuracies can be maintained simply by using more AR filter coefficients. At some point this eventually causes spurious components to occur in the spectra. It should be noted however that the amplitude levels of these components are always approximately equal to that of the noise floor. Thus, large enough amounts of noise will eventually stop small components from being identified, but that increasing noise does not cause any real concern for the algorithm and good frequency and power estimates are still retained for the larger sinusoids.

Applying the FBLS technique to a signal obtained from an engine mounted accelerometer produced very encouraging results. A comparison is made between an FBLS spectrum

produced from 128 points with that of a 1024 point FFT. In the comparison the FBLS technique's frequency and power estimates appear to be as good, if not better, than those of the FFT. It is also shown that the FBLS technique detects a significant amplitude change in one of the vibration components which is missed by the FFT due to the FFT's longer sample block. Only limited success was obtained with the PSLE because of the relatively high noise level.

The problem of AR filter order selection has really not been solved in this work, but it has been shown that the FBLS technique can be used with highly over determined AR filters without detrimental effects, such as frequency splitting, occuring. This means that the order selection is not critical and that as long as a minimum number of filter coefficients are used then all the required information will be obtained. Some rules of thumb are given for estimating this minimum number of coefficients, and largely depend on number of sinusoids and noise level. These are shown to work well with the vibration signal.

Summarising, the forward-backward least squares technique, coupled with a power estimation method developed by the author can perform extremely well on real signals, and provides significant improvements over the performance of the FFT. This technique however is not straight forward to use and there is still a considerable amount of work to do in providing a robust package that can be used generally in engine analysis.

[1] ABLES J G "Maximum entropy spectral analysis". Astron. Astrophys. suppl. series vol 15, pp 383-393, 1974.

[2] AKAIKE H "Fitting auto-regressive models for prediction". Ann. Inst. Stat. Math. v.21, pp 243-247, 1969.

[3] ANDERSON N "On the calculation of filter coefficients for maximum entropy spectrum analysis". Geophysics vol 39, pp 69 72, Feb 1974

[4] BARRODALE I and ERICKSON R E "Algorithms for least square linear prediction and maximum entropy spectral analysis". Geophysics vol 45, pp 420-466, March 1980.

[5] BERGLAND G D "A guided tour of the fast Fourier transform". IEEE Spectrum vol 6, pp 41-52, July 1969.

[6] -- "The FFT algorithm for real valued series". Commun. Assoc. Comput. Mach. vol 11, pp 703-710, Oct 1968.

[7] BLACKMAN R B and TUKEY J W "The measurment of power spectra from the point of view of communications engineering". New York: Dover 1959.

[8] BRAUN S and RAM Y "Time and frequency identification methods in over-determined systems". Mech. Systems and Signal Processing vol 1, pp 245-257, 1987.

[9] BOUSFIELD B M "Assessment of FFT windows for use in real time". Rolls Royce EIR00928, Oct 1985.

[10] -- "Intelligent dynamic data acquisition system". Rolls Royce EIR00987, Nov 1986.

[11] -- "IDDAS/PDP-11 handlers and routines". Rolls Royce EIR 00987, Jan 1987.

[12] BOX G and JENKINS G "Time series analysis". Forecasting and control, San Fransisco: Holden-Day, 1970.

[13] BURG J P "The relationship between maximum entropy spectra and maximum likelihood spectra". Geophysics vol 37, pp 375-376, April 1972.

[14] BURRS C S and PARKS T W "DFT/FFT and convolution algorithms". Wiley interscience publications 1985.

[15] CAPON "High resolution frequency-wave number spectrum analysis". Proc IEEE pp 1408-1418 1969.

[16] -- "Probability distribution for estimators of the frequency wave number spectrum". Proc IEEE 1785-1786, 1970

[17] COOLEY J W and TUKEY J W "An algorithm for machine calculation of complex Fourier series". Math. Computing vol 19, pp 297-301, April 1965.

[18] COOLEY J W, WELCH P D and LEWIS P "Historical notes of the FFT". IEEE Trans., Audio and Electro-acoustics vol 15, pp 76-79, June 1967.

[19] -- "The FFT and its applications". IBM research papers, RC 1743 Feb 1967.

[20] DANIELSON G C and LANCZOS C "Some improvments in practical Fourier analysis and their application to X-ray scattering from liquids". J.Franklin Inst. vol 233, pp 365-380 and 435-452, April 1942.

[21] DURBIN J "The fitting of time series models". Rev Inst. Int. de Stat., vol 28, pp 233-244, 1960.

[22] HARRIS F J "The use of windows for harmonic analysis with the discrete Fourier transform" Proc. IEEE vol 66, 1978.

[23] -- "Trigonometric transforms". San Diego university.

[24] HAYES M H and CLEMENTS M A "An efficient algorithm for computing Pisarenko's harmonic decomposition using Levinson's recursion". IEEE Trans. Acoustics, Speech and Sig. Processing vol 34, June 1986.

[25] HILDERBRAND F B "Introduction to numerical analysis". New York: McGraw-Hill, 1956.

[26] GHAIDAN A K "A study of the application of modern techniques to speech waveform analysis", Ph.D thesis, Loughborough university, May 1986.

[27] KAY S M and MARPLE S L "Spectrum analysis - A modern perspective" Proc. of IEEE vol 69, pp 1380-1419 Nov 1981.

[28] LEVINSON N "The Wiener criterion in filter design and prediction". J.Math. Phys. vol 25, pp 261-278, 1947.

[29] LACOSS R T "Data adaptive spectral analysis methods". Geophysics vol 36, pp 661-675, Aug 1971.

[30] MARPLE L "A new auto-regressive spectrum analysis algorithm". IEEE Trans. on Acoustics, Speech and Signal Processing vol 28, Aug 1980.

[31] MORRIS L R "Digital signal processing software". DSPS Inc. fourth edition 1984.

[32] NUTALL A H "Spectral analysis of a univariate process with bad data points, via maximum entropy and linear prediction techniques". Naval underwater systems centre, March 1976.

[33] -- "Spectrum estimation by means of overlapped processing of windowed data". Naval underwater systems centre, 1969.

[34] OPPENHEIM A V and SCHAFER R W "Digital signal processing" Englewood Cliffs, N.J.: Prentice Hall 1975.

[35] OTNES R W and ENACHSON L. "Digital time series analysis". New York: Wiley 1972.

[36] ORFANDIS S J "A reduced music algorithm". IEEE pp 165-167, July 1986.

[37] PISARENKO V F "On the estimation of spectra by means of non-linear function of the covariance matrix". Geophysical J. Royal Astronomical Soc. vol 28, pp 511-531, 1972.

[38] -- "The retrieval of harmonics from a covariance function". Geophysical J. Royal Astronomical Soc. vol 33, pp 347-366, 1973.

[39] RUNGE C and KONIG H "Die grundlehren der mathematischen wissenschoffen". Vorlesurgen uber Numerisches Rechnen vol 11, Berlin: Julius Springer 1924.

[40] SCHUSTER A "On the investigation of hidden peridices with application to a supposed 26 day period of meteorological phenomena". Terrestial Magnatism vol 3, pp13-41 March 1898.

[41] -- "The periodogram of magnetic declination as obtained from the records of Greenwich observatory during the years 1871-1885". Trans. Cambridge philisophical Soc vol 18, pp 107-135, 1899.

[42] SORELLA S and GHOSH S K "Improved method for the discrete fast Fourier transform". Rev. Sci. Instrum. vol 55, pp 1348-1351, Aug 1984.

[43] ULRYCH T J and BISHOP T N "Maximum entropy spectrum analysis and auto-regressive decomposition". Rev. Geophysics and Space physics vol 13, pp 183-200, Feb 1975.

[44] ULRYCH T J and CLAYTON R W "Time series modelling and maximum entropy" Phys. Forth Planetary Interiors vol 12, pp188-200, Aug 1976.

[45] WEINER N "Generalised harmonic analysis". Acta Mathmatica vol 55, pp 117-258, 1930.

[46] "Texas TMS32010 users guide". Texas inst. incorp. 1983.

[47] "What is the fast Fourier transform". Proc. IEEE vol 55, Oct 1976. Selection of authors.

[48] "Programs for digital signal processing". IEEE digital signal processing committee, IEEE press 1979.

```
                ;*********************************************************
                ;
                ;              SQUARE ROOT SUBROUTINE
                ;          Enter with value in accumulator
                ;          Exit with square root in accumulator
                ;
                ;              Copyright Rolls Royce 1986
                ;                *** BRUCE BOUSFIELD ***
                ;                       27-3-86
                ;*********************************************************
                ;
047F  5810      SQRT:   SACH    TEMPH   ; store entry value
0480  5011              SACL    TEMPL
0481  FB0004AE          BLEZ    ZEROO
0483  6200              SUBH    ONE
0484  FA000489          BLZ     LOWW    ; branch if top word=0
                ;
0486  2E00              LAC     ONE,14
0487  F900048A          B       HII
                ;
0489  2700      LOWW:   LAC     ONE,7
048A  5014      HII:    SACL    XOLD    ; xold=1/2 of max root
048B  5002              SACL    TEMP    ; temp=1/2 of max root
048C  2F02      ROOT:   LAC     TEMP,15
048D  5802              SACH    TEMP    ; divide temp by two
048E  6200              SUBH    ONE
048F  FA0004A4          BLZ     END     ; branch if temp WAS equal to 1
                ;
0491  6510              ZALH    TEMPH
0492  6111              ADDS    TEMPL   ; acc=entry value
0493  6A14              LT      XOLD
0494  6D14              MPY     XOLD
0495  7F90              SPAC            ; acc=entry value - square of guess
0496  FF0004B0          BZ      FINI    ; branch if guess**2=entry value
0498  FC00049F          BGZ     HIGHER  ; branch if entry value higher
                ;
049A  2014              LAC     XOLD    ; if entry value smaller
049B  1002              SUB     TEMP    ; then reduce xold
049C  5014              SACL    XOLD
049D  F900048C          B       ROOT
                ;
049F  2014      HIGHER: LAC     XOLD    ; if entry value greater
04A0  0002              ADD     TEMP    ; then increase xold
04A1  5014              SACL    XOLD
04A2  F900048C          B       ROOT
                ;
04A4  6510      END:    ZALH    TEMPH   ; final test after '1' was
04A5  6111              ADDS    TEMPL   ; added/subtracted to check
                                        ; whether guess is still too big
04A6  6A14              LT      XOLD
04A7  6D14              MPY     XOLD
04A8  7F90              SPAC
04A9  FD0004B0          BGEZ    FINI
                ;
04AB  2014              LAC     XOLD    ; if too big reduce by 1
04AC  1000              SUB     ONE
04AD  7F8D              RET
                ;
04AE  7F89      ZEROO:  ZAC             ; return with zero
04AF  7F8D              RET
                ;
04B0  2014      FINI:   LAC     XOLD
04B1  7F8D              RET             ;return with xold in accumulator
```

```
001A  0000      LOGS: .WORD     0,1,3,4,6,7,8,10,11,13,14,15,17,18,19,20
002A  0016            .WORD     22,23,24,26,27,28,29,31,32,33,34,35,37,38
0038  0027            .WORD     39,40,41,42,44,45,46,47,48,49,50,51,52,53
0046  0037            .WORD     55,56,57,58,59,60,61,62,63,64,65,66,67,68
0054  0045            .WORD     69,70,71,72,73,74,75,76,77,78,79,80,81,81
0062  0052            .WORD     82,83,84,85,86,87,88,89,90,91,91,92,93,94
0070  005F            .WORD     95,96,97,97,98,99,100,101,102,103,103,104
007C  0069            .WORD     105,106,107,107,108,109,110,111,111,112
0086  0071            .WORD     113,114,115,115,116,117,118,118,119,120
0090  0079            .WORD     121,121,122,123,124,124,125,126,127,127
009A  0080            .WORD     128
                ;
                ;***************************************************************
                ;
                ;       CONVERTION TO LOGARITHMIC VALUE
                ;
                ;       LOGS :enter with 32 bit number in acc
                ;            :exit with 12 bit log in acc
                ;
                ;            Copyright Rolls Royce 1985
                ;             *** BRUCE BOUSFIELD ***
                ;                    10-4-85
                ;***************************************************************
                ;
                ;
042F  6881      LOG:    LARP    £1
0430  5810              SACH    TEMPH
0431  5011              SACL    TEMPL     ; save 32 bit input
0432  FF000477          BZ      ZOUT      ; branch if zero
                ;
0434  2010              LAC     TEMPH
0435  FF000459          BZ      LOW       ; branch if nothing in top word
                ;
0437  1000              SUB     ONE               ; branch if top word
0438  FF00046C          BZ      SCASE             ; is equal to 1
                ;
043A  710F              LARK    AR1,£15           ; set counter to 15 (count 16)
043B  2110      LOG1:   LAC     TEMPH,1           ; shift the top word until
043C  5010              SACL    TEMPH             ; the most significant bit
043D  FA000441          BLZ     DONE1             ; overflows, then branch
043F  F400043B          BANZ    LOG1              ; dec AR1
                ;
0441  3103      DONE1:  SAR     AR1,TEMP1  ; as if by magic the aux reg
                ;                          contains the bit number where
                ;                          the msb was originally.
0442  6898              MAR     *-         ; decrement AR1
0443  2F11      LOG2:   LAC     TEMPL,15   ; shift right the bottom word to
0444  5811              SACH    TEMPL      ; line up with the shifted top word
0445  2011              LAC     TEMPL
0446  7912              AND     LGMASK1    ; remove sign extension
0447  5011              SACL    TEMPL
0448  F4000443          BANZ    LOG2       ; dec AR1
                ;
044A  2010              LAC     TEMPH   ; add together the left shifted top
044B  0011              ADD     TEMPL   ; and bottom words
044C  5010              SACL    TEMPH
044D  2710              LAC     TEMPH,7 ; get the top 7 bits of this word
044E  0F00              ADD     ONE,15  ; stop truncation error
044F  5810              SACH    TEMPH
0450  2010              LAC     TEMPH

0451  7913              AND     LGMASK2 ; remove sign extension; log fraction
0452  000F              ADD     LOGADD
0453  6710              TBLR    TEMPH   ; get fraction from table
0454  2010              LAC     TEMPH
0455  0703              ADD     TEMP1,7   ; add bit number; log integer
0456  0B00              ADD     ONE,11    ; add 16 'cos its the top word
0457  F900047A          B       FIN
                ;
0459  710F      LOW:    LARK    AR1,£15           ; set count to 15 (count 16)
045A  2111      LOG3:   LAC     TEMPL,1           ; shift bottom word until the
045B  5011              SACL    TEMPL             ; msb overflows, then branch
045C  FA000460          BLZ     DONE2
045E  F400045A          BANZ    LOG3              ; dec AR1
                ;
0460  3103      DONE2:  SAR     AR1,TEMP1         ; AR1 contains bit number
0461  2711              LAC     TEMPL,7           ; get top 7 bits
0462  0F00              ADD     ONE,15            ; stop truncation error
0463  5811              SACH    TEMPL
0464  2011              LAC     TEMPL
0465  7913              AND     LGMASK2  ; remove sign extension; log fraction
0466  000F              ADD     LOGADD
0467  6711              TBLR    TEMPL             ; get fraction from table
0468  2011              LAC     TEMPL
0469  0703              ADD     TEMP1,7           ; add bit number; log integer
046A  F900047A          B       FIN
                ;
046C  2711      SCASE:  LAC     TEMPL,7         ; get top 7 bits of bottom word
046D  0F00              ADD     ONE,15          ; stop truncation error
046E  5811              SACH    TEMPL
046F  2011              LAC     TEMPL
0470  7913              AND     LGMASK2  ; remove sign extension; log fraction
0471  000F              ADD     LOGADD
0472  6711              TBLR    TEMPL             ; get fraction from table
0473  2011              LAC     TEMPL
0474  0B00              ADD     ONE,11            ; add 16; msb was in bit 16
0475  F900047A          B       FIN
                ;
0477  7F89      ZOUT:   ZAC
0478  6880              LARP    £0
0479  7F8D              RET             ; return with zero in accumulator
                ;
047A  1900      FIN:    SUB     ONE,9   ; subtracts 4 from log integer, this
047B  FA000477          BLZ     ZOUT    ; removes some of the noise!!!
047D  6880              LARP    £0
047E  7F8D              RET             ; return with log in acc
                ;
```

```
C
C     M/2 stages, spectrum splitting,
C     then final stage of 2*M point FFT
C
      m=m/2
      n=2**m
      do 20 l = 1,m
         le = 2**l
         le1 = le/2
         u = (1.0,0.0)
         w = cmplx(cos(pi/float(le1)),sin(pi/float(le1)))
         do 20 j = 1,le1
            do 10 i = j,n,le
            ip = i + le1
C
C           Radix 2 butterfly
C
            t = x(ip)*u
            x(ip) = x(i) - t
10          x(i) = x(i) + t
            u = u*w
20    continue
C
C
      x(n+1)=cmplx(aimag(x(1)),0.0)
      x(1)=cmplx(real(x(1)),0.0)
C
      do 3 i = 2,n/2+1
         temp=real(x(i))+real(x(n-i+2))
         temp2=aimag(x(i))-aimag(x(n-i+2))
         temp3=real(x(i))-real(x(n-i+2))
         temp4=aimag(x(i))+aimag(x(n-i+2))
         x(i)=cmplx(temp/2,temp2/2)
         x(n-i+2)=cmplx(temp/2,-temp2/2)
         x(i+n)=cmplx(temp4/2,-temp3/2)
         x(2*n-i+2)=cmplx(temp4/2,temp3/2)
3     continue
C
         m=m+1
         l=m
         n=2**m
         le = 2**l
         le1 = le/2
         u = (1.0,0.0)
         w = cmplx(cos(pi/float(le1)),sin(pi/float(le1)))
         do 80 j = 1,le1
            do 90 i = j,n,le
            ip = i + le1
C
C           Radix 2 butterfly
C
            t = x(ip)*u
            x(ip) = x(i) - t
90          x(i) = x(i) + t
            u = u*w
80    continue
C
```

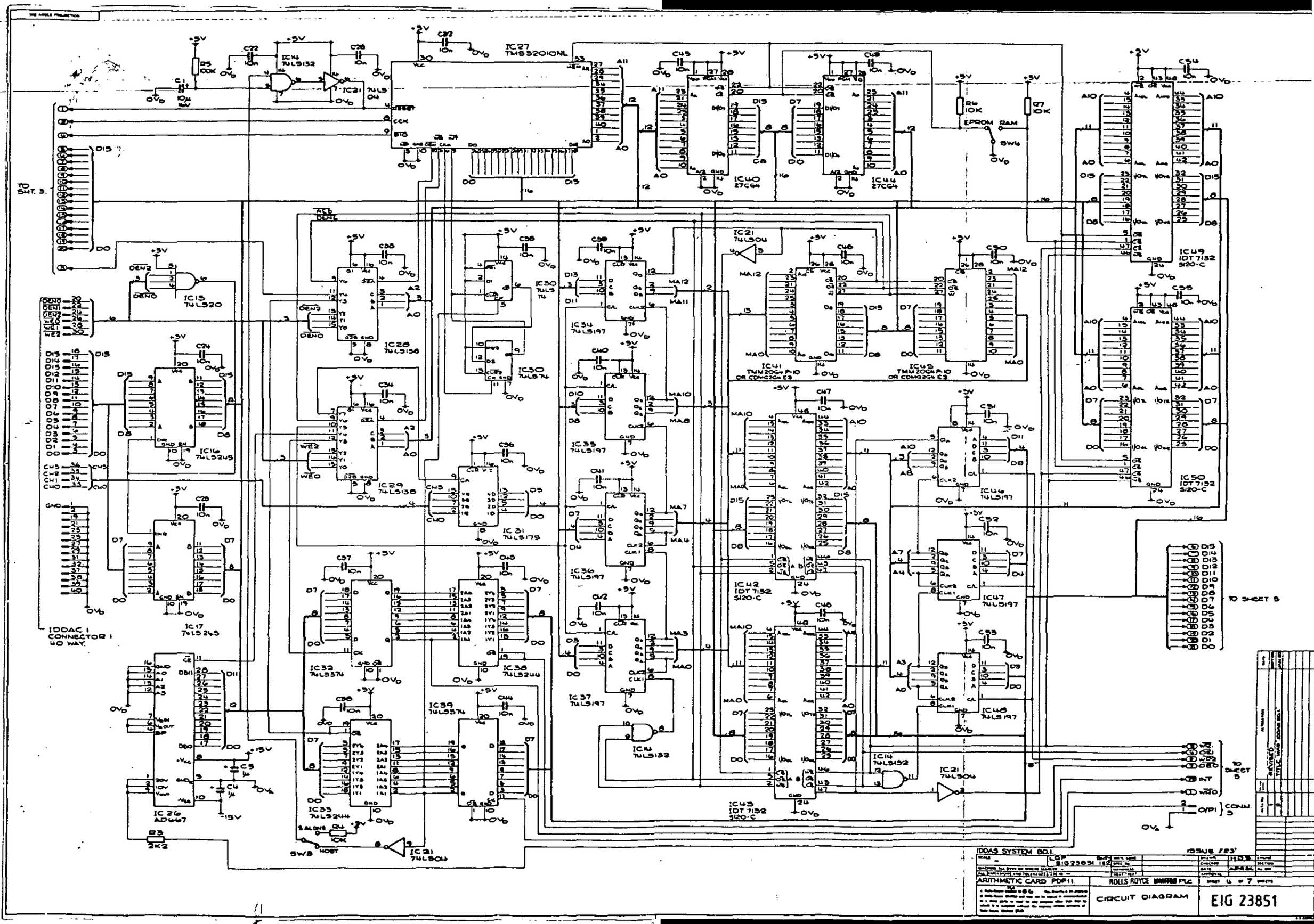
PAGE 1: 'RING_BUFFER' NOVEMBER 10 1986

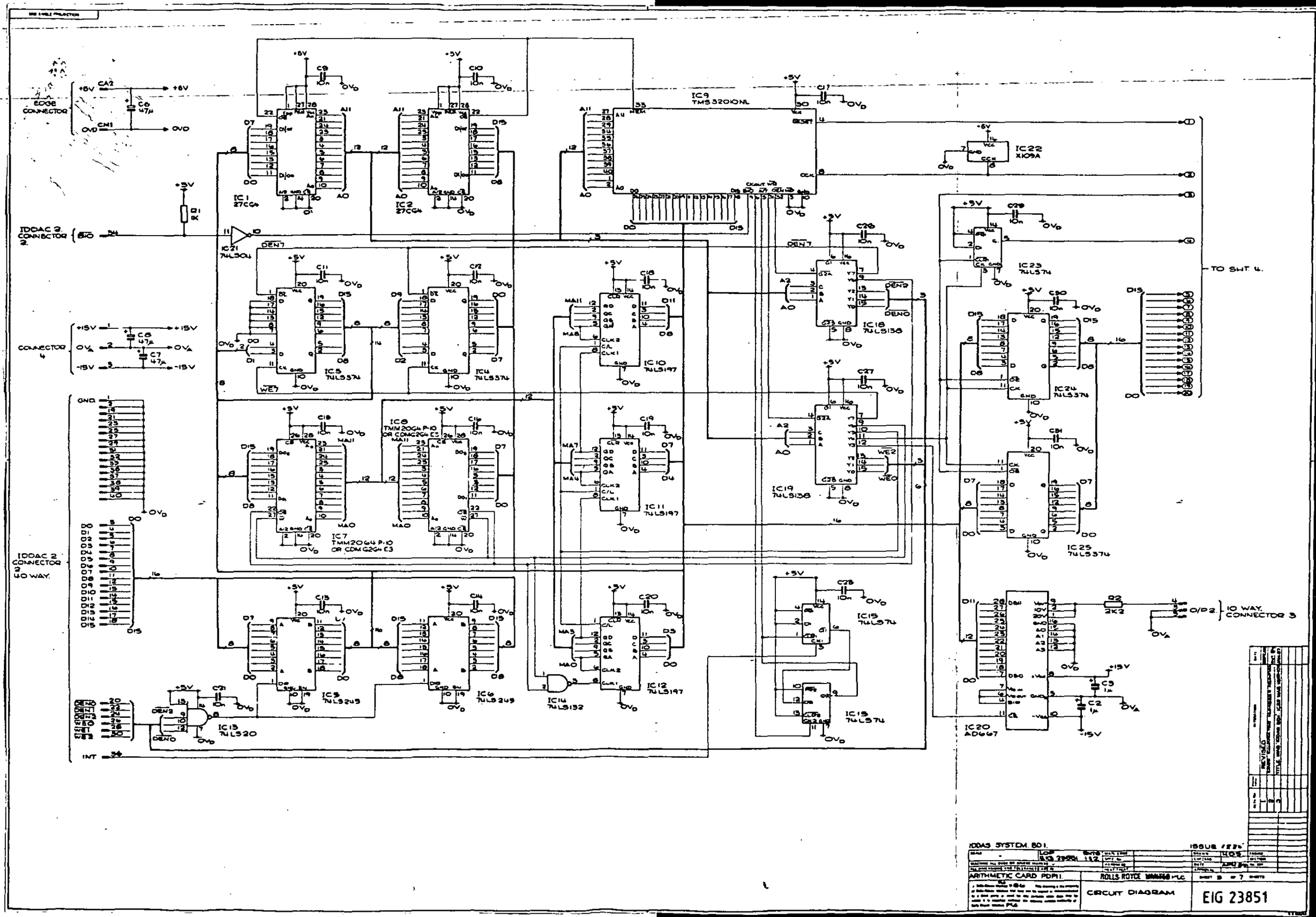
```
                        1 ****************************************************
                        2 *
                        3 *            Ring Buffer for IDDAS
                        4 *
                        5 *           Bruce Bousfield  9-7-86
                        6 *
                        7 *        Copyright Rolls Royce July 1986
                        8 *
                        9 ****************************************************
                       10 *
                       11          TTL   'RING_BUFFER'
                       12 *
0000                   13 DAC      EQU   0         output DAC
0001                   14 COMLINK  EQU   1         latch to SP processor
0001                   15 ADC      EQU   1         input ADC
0002                   16 RAMADD   EQU   2         external ram address latch
0003                   17 RAMDAT   EQU   3         external ram data latch
                       18 *
0000                   19 ONE      DATA  1         =1
0001                   20 ZERO     DATA  1         =0
0002                   21 POINT    DATA  1         current address of output buffer
0003                   22 TEMP     DATA  1         temporary store
0004                   23 TOP      DATA  1         address of top of input buffer
0005                   24 COUNT    DATA  1         general counter store
0006                   25 OUTPUT   DATA  1         intermediate output data store
                       26 *
                       27 *
                       28 ************************************
                       29 *
0000 F900 0009         30          B     START     reset vector
                       31 *
                       32 *
                       33 ************************************
                       34 *
                       35 *    Interrupt service routine
                       36 *
                       37 ************************************
                       38 *
0002 4103              39 INT      IN    TEMP,ADC  get input; **interrupt entry poin
0003 2004              40          LAC   TOP
0004 0000              41          ADD   ONE
0005 5004              42          SACL  TOP       increment address of top of buffe
0006 4A04              43          OUT   TOP,RAMADD
0007 4803              44          OUT   TEMP,RAMDAT put input value at top of buffe
0008 7F8D              45          RET
                       46 *
```

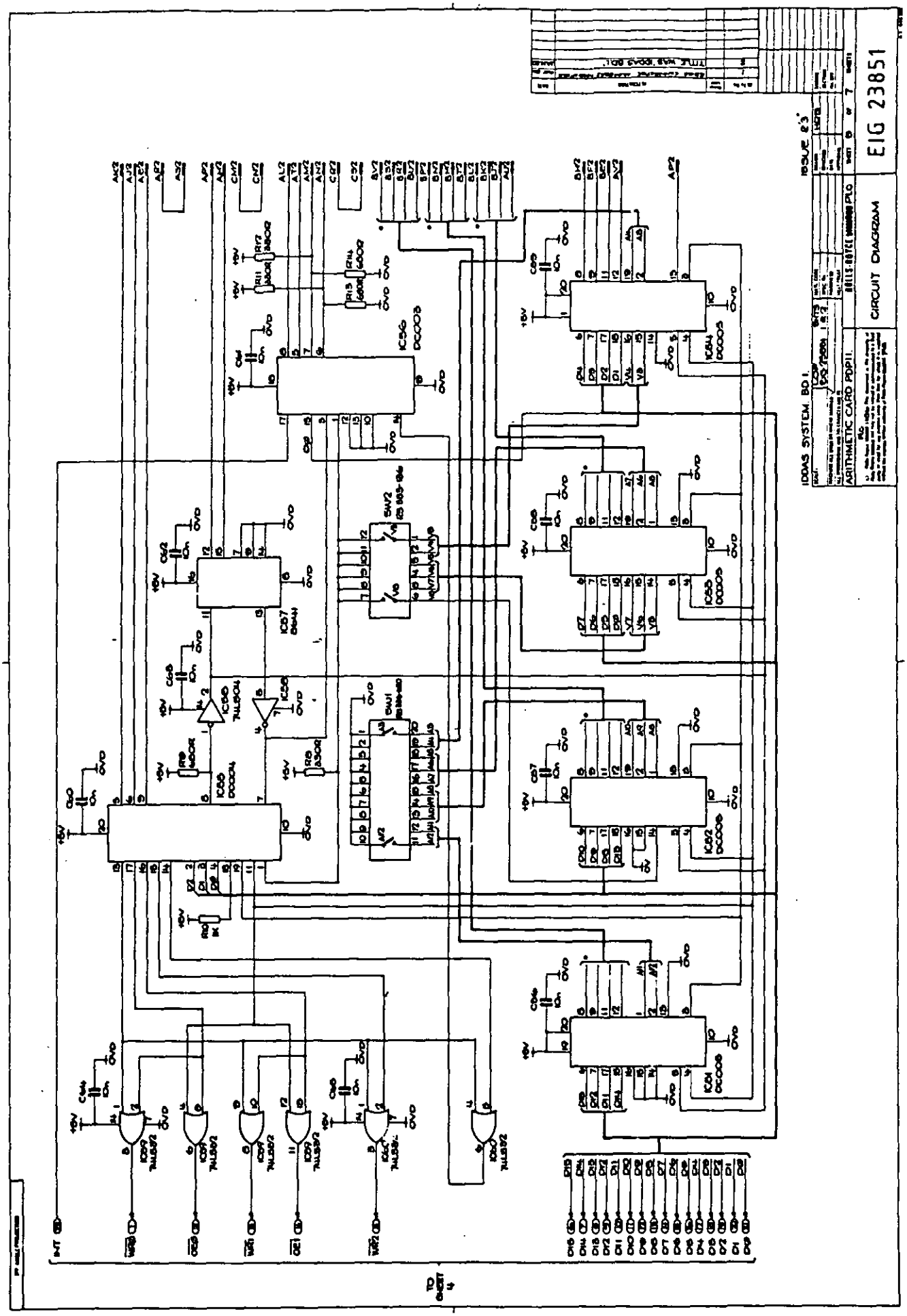
PAGE 2: 'RING_BUFFER' NOVEMBER 10 1986

```
                       48 ****************************************
                       49 *
                       50 *      Main Routine
                       51 *
                       52 *
                       53 *
0009 F800 0027         54 START    CALL  INIT      initialise
                       55 *
0008 7F82              56 ME       EINT            enable interrupt
000C F600 0008         57          BIOZ  ME        wait for transfer instuction from
                       58 *
000E 7F81              59          DINT            disable interrupt
000F 2004              60          LAC   TOP
0010 1A00              61          SUB   ONE,10    subtract 1024 from top of buffer
0011 5002              62          SACL  POINT     store in POINT
0012 7F82              63          EINT            enable int
                       64 *
                       65 * ensures interrupt routine not locked out for more than 1
                       66 *
                       67 *
0013 7F81              68 LOOP     DINT            disable int
0014 4A02              69          OUT   POINT,RAMADD
0015 4306              70          IN    OUTPUT,RAMDAT get data from buffer
0016 7F82              71          EINT            enable int
0017 4906              72          OUT   OUTPUT,COMLINK send it to SP_processor
                       73 *
                       74 * stop interrupt routine lockout
                       75 *
0018 7F81              76          DINT            disable int
0019 2002              77          LAC   POINT
001A 0000              78          ADD   ONE
001B 5002              79          SACL  POINT     increment output buffer address
001C 7F82              80          EINT            enable int
                       81 *                        EINT is required often as it is n
                       82 *                        in the interrupt routine (to save
                       83 *                        The SP_proc can only accept a new
                       84 *                        every 28 clock cycles, thus this
0010 7F82              85          EINT            a minimum loop time of 28 cycles
001E 7F82              86          EINT            interrupts occur, obviously the l
001F 7F82              87          EINT            slows down when interrupts are fr
0020 7F82              88          EINT            but this is only noticeable at in
0021 7F82              89          EINT            frequencies above 66 Khz.
0022 7F82              90          EINT
                       91 *
                       92 *
0023 F600 0008         93          BIOZ  ME        check to see if more wanted by SP
0025 F900 0013         94          B     LOOP      if so carry on
                       95 *
                       96 *
                       97 *
```

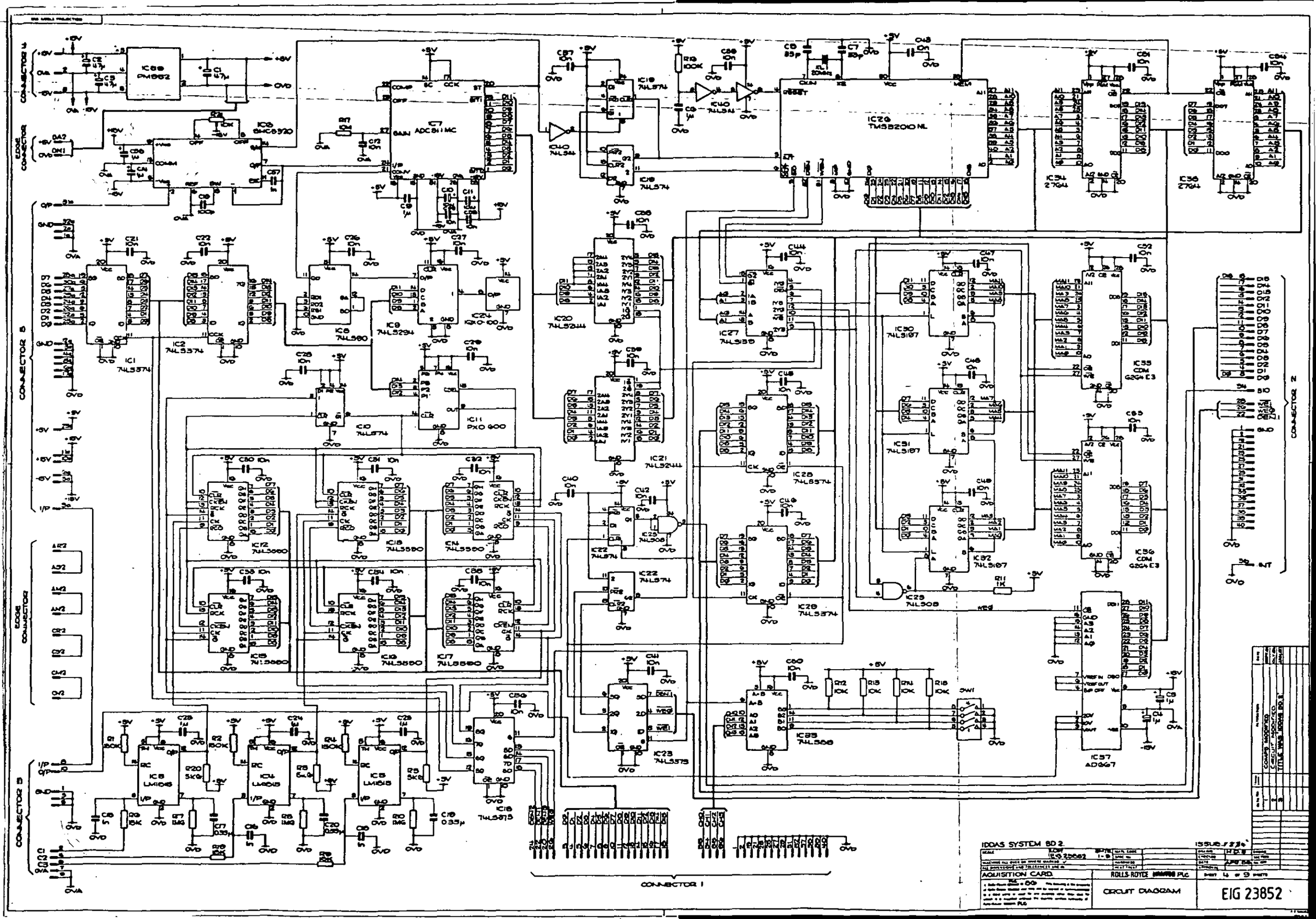
APPENDIX E IDDAS Circuit diagram (Arithmetic 1)




EDGE CONNECTOR
IDDAC 2 CONNECTOR 2.
CONNECTOR 4
IDDAC 2 CONNECTOR 3 40 WAY.
IC1 27C64
IC2 27C64
IC3 74LS245
IC4 74LS374
IC5 74LS374
IC6 74LS245
IC7 TMM2064 P-10 OR CDM6264 E3
IC8 TMM2064 P-10 OR CDM6264 E3
IC9 TMS3201ONL
IC10 74LS197
IC11 74LS197
IC12 74LS197
IC13 74LS20
IC14 74LS132
IC15 74LS374
IC15 74LS374
IC18 74LS138
IC19 74LS138
IC20 AD667
IC21 74LS04
IC22 X1084A
IC23 74LS374
IC24 74LS374
IC25 74LS374
TO SHT 4.
10 WAY. CONNECTOR 3
IDDAS SYSTEM 801
ARITHMETIC CARD PDP11
ROLLS ROYCE
CIRCUIT DIAGRAM
EIG 23851


EIG 23851
CIRCUIT DIAGRAM
IDDAS SYSTEM BD 1.
ARITHMETIC CARD PCB 11.

APPENDIX E IDDAS Circuit diagram (Acquisition)



```
C*****************************************************************
C
C       Prony spectral line estimation
C
C*****************************************************************
C
      double precision xin(128), xt(20,128), r(20,20), y(128), a(20)
      double precision t(128,20), h(128,20), xx(128,20)
      complex b(40), temp2(40)
      double precision aa(20,20), wka1(20), wka2(20), a(20)
      double precision tol, poly(41), zre(40), zim(40)
      double precision sum
      complex csum
      real res(20,3)
      integer p, twop1
      complex zed(40)
      complex phi(128,40), phit(40,128)
      complex epsiln(40,40)
      double precision wkspce(40,40)
      double precision depsiln(2,40,40), dtemp2(2,40)
      double precision db(2,40)

      data p1/0.1d0/

      print *,'enter p?'
      input *,p

      open(unit=5,file='ref.dat')
      open(unit=6,file='plotpn.dat')
      read(5,*)nn

C*************************************************

      do 10 i=1,nn
         read(5,*)xin(i)
10    continue

      close(5)

C*****************************************

C   create toeplitz plus hankel matrices

      twop1=2*p+1
      irow=nn-2*p
      icol=p

      do 40 ir=1,irow
         do 40 ic=1,icol
            t(ir,ic)=xin(p-ic+ir)
40    continue

      do 80 ir=1,irow
         do 80 ic=1,icol
            h(ir,ic)=xin(ir+ic+p)
80    continue

      do 100 ir=1,irow
         do 100 ic=1,icol
            xx(ir,ic)=t(ir,ic)+h(ir,ic)
            xt(ic,ir)=xx(ir,ic)
100   continue

      do 110 ir=1,irow
         y(ir)=2d0*xin(ir+p)
110   continue

C*********************************************************

C    form r (Xt.X) and S (Xt.Y)

      do 200 ir=1,icol
         do 200 ic=1,icol

            sum=0.
            do 210 j=1,irow
               sum=sum+xt(ir,j)*xx(j,ic)
210         continue
            r(ir,ic)=sum

200   continue

      do 220 ir=1,icol
         sum=0.
         do 230 j=1,irow
            sum=sum+xt(ir,j)*y(j)
230      continue
         a(ir)=sum
220   continue

C*********************************************************

C    evaluate 'a' for R.a = S    using linear eqt route

      ir=20
      n=p
      iaa=20
      ifail=1

      call f04atf(r,ir,s,n,a,aa,iaa,wka1,wka2,ifail)

      if (ifail.eq.0) goto 240
      print *,'failed'
      stop

240   continue

C*********************************************************

C     create polynomial and then find zeroes

      do 300 i=1,p
         poly(p+1+i)=-a(i)
         poly(p+1-i)=-a(i)
300   continue

      poly(p+1)=2d0

      tol=x02aaf(p1)
      ifail=1

      call c02aef(poly,twop1,zre,zim,tol,ifail)

      if(ifail.eq.0) goto 350
      print *,'fail poly'
      stop

350   p1=4.*atan(1.0)

      p=2*p
      do 400 k=1,p
         zed(k)=cmplx(zre(k),(zim(k)))
400   continue
```

```
134
135         do 440 k=2,p,2
136            kk=k/2
137            res(kk,1)=atan(abs(aimag(zed(k)))/real(zed(k)))/2.0*pi
138            if(real(zed(k)).lt.0.0) res(kk,1)=res(kk,1)+0.5
139  440    continue
140
141  C*********************************************************
142
143  C      calculate amplitude
144
145         do 500 i=1,nn
146            do 510 j=1,p
147               phi(i,j)=zed(j)**(i-1)
148               phit(j,i)=conjg(phi(i,j))
149  510       continue
150  500    continue
151
152         do 800 i=1,p
153            do 800 j=1,p
154               csum=cmplx(0.,0.)
155               do 810 k=1,nn
156                  csum=csum+(phit(i,k)*phi(k,j))
157  810          continue
158               epsiln(i,j)=csum
159  800    continue
160
161         do 550 i=1,p
162            csum=cmplx(0.,0.)
163            do 560 k=1,nn
164               csum=csum+(phit(i,k)*cmplx(xin(k),0d0))
165  560       continue
166            temp2(i)=csum
167  550    continue
168
169  C*********************************************************
170
171         do 600 i=1,p
172            do 600 j=1,p
173               depsiln(1,i,j)=real(epsiln(i,j))
174               depsiln(2,i,j)=aimag(epsiln(i,j))
175  600    continue
176
177         do 630 i=1,p
178            dtemp2(1,i)=real(temp2(i))
179            dtemp2(2,i)=aimag(temp2(i))
180  630    continue
181
182         ia=40
183         ib=40
184         ic=40
185         n=p
186         m=1
187         ifail=1
188         call f04adf(depsiln,ia,dtemp2,ib,n,m,db,ic,wkspce,ifail)
189         if(ifail.eq.0) goto 610
190         print *,'inverse fail'
191         stop
192
193  610    do 620 i=1,p
194            b(i)=cmplx(db(1,i),db(2,i))
195  620    continue
196
197         do 650 i=2,p,2
198            kk=i/2
199            res(kk,3)=(cabs(b(i))+cabs(b(i-1)))
200  650    continue
201
202         do 660 k=2,p,2
203            kk=k/2
204            res(kk,2)=(atan(abs(aimag(b(k)))/real(b(k))))/pi*180
205            if(real(b(k)).lt.0.0) res(kk,2)=res(kk,2)+180
206  660    continue
207
208         do 680 i=1,p/2
209         print *,'freq=',res(i,1),'  amp=',res(i,3),'  phase=',res(i,2)
210  680    continue
211
212         write(6,*)'input data'
213
214         do 900 i=1,p/2
215            write(6,*) res(i,1),',',20*alog10(res(i,3))
216  900    continue
217
218         write(6,*)'eod.'
219
220         stop
221         end
```