A SYSTEM FOR THE ACQUISITION AND DIGITAL ANALYSIS OF LOWER LIMB FLOW WAVEFORMS.

BY : L. SMITH

Supervisors : Dr. W.L. Capper
Assoc. Prof. A.E. Bunn
Prof. E.J. Immelman

THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN BIOMEDICAL ENGINEERING

UNIVERSITY OF CAPE TOWN
MARCH 1994

DECLARATION

I **Leonard Smith** hereby declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science at the University of Cape Town. It has not been submitted before in any form for any degree or examination at any other university

Signed :
Date : Signed by candidate

ACKNOWLEDGEMENTS

The author wish to thank all those who have aided in the work presented herein. A special work of gratitude is expressed to my friends and parents for their moral and emotional support during my stay at university.

ABSTRACT

A PC based waveform acquisition and analysis system has been developed for use in aorto-iliac arterial assessment. A Motorola DSP56001 based system containing dual Analog to Digital converters is used to sample phase quadrature demodulated signals from a commercially available continuous wave Doppler unit. The Power Spectral Density is calculated using an autoregressive model from which the mean velocity waveform is calculated. This waveform is used to calculate the damping factor, vessel compliance and runoff resistance of a simple electrical model of the lower limb arterial circulation using a non-linear regression technique of curve fitting in the time domain. A pilot study using the system shows a significant separation ($p < 0.001$ Mann Whitney U-test) between the damping factors of a normal control group (quartile range = 0.15 - 0.25; median = 0.19) and a patient group with angiographically determined aorto-iliac arterial disease (quartile range = 0.45 - 0.89; median = 0.49).

CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AIC	Akaike's information criterion
ALU	Arithmetic logic unit
AR	Autoregressive
ASPI	Ankle systolic pressure index
BGI	Borland Graphic Interface
CAT	Criterion of autoregressive transfer function
CW	Continuous wave
DMA	Direct memory access
DSP	Digital signal processing
FDD	Floppy disk drive
FFT	Fast Fourier transform
FPE	Final prediction error
FZC	First zero crossing
HCR	Host Control Register
HDD	Hard disk drive
HSR	Host Status Register
IBM	International Business Machines
IIR	Infinite impulse response
IPL	Interrupt priority level
kb	kilobyte
LTA	Laplace transform analysis
LTD	Laplace transform damping
LTE	Laplace transform elasticity
LTR	Laplace transform resistance
Mb	Megabyte
MHz	Megahertz
MIPS	Million instructions per second
ms.	millisecond
MSE	Mean square error
PC	Personal computer
PCA	Principal component analysis
PI	Pulsatility index
PSD	Power Spectral Density
SBI	Spectral broadening index
VDU	Video display unit
VGA	Video Graphics Array

CHAPTER ONE

1.1 INTRODUCTION

Blood flow velocity within arteries varies from slow moving cells near the arterial walls to fast moving cells near the vessel center. When an artery is bathed in ultrasound, reflections occur from the moving cells. The reflected signal thus contains Doppler shifted frequencies which are proportional to the velocities of these cells. It follows that with varying velocities, a spectrum of representative frequencies is obtained. Once the reflected signal is demodulated, spectral analysis can extract the individual frequencies. If a mean frequency is chosen from the spectrum, a resultant mean blood velocity waveform can be constructed for a given time interval.

The waveform shape is dependent not only on the pulsatile pump (cardiac input) but also on the characteristics of the arterial system (e.g. arterial wall compliance, lumen diameter) and the fluid dynamics (e.g. viscosity, density). Studies have shown that the blood velocity waveform shape can be predicted if the characteristics of the arterial system are known. It is postulated that the reverse is also true i.e. if the shape of the blood velocity waveform is known, then the characteristics of arterial system can be extracted mathematically.

Two factors are crucial in this process. The first is that an optimum method of data acquisition and spectral analysis be applied to extract the blood velocity waveform. The second is that a representative model of the arterial system be used to extract the arterial parameters. This study concentrated on the former. A system has been developed which is able to measure, analyse and store the information contained in the reflected signal component. Work in the area of system modelling and parameter extraction is continuing as part of an ongoing project at the University of Cape Town's department of Biomedical Engineering.

Work from this thesis has been accepted for publication by *Automedica*, an international journal for automation in the medical sciences (Capper, Smith and Immelman, 1994).

1.2 THESIS OBJECTIVES

The thesis objectives are :

I To develop software which will allow a Digital Signal Processing (DSP) unit to sample the output from a suitable continuous wave (CW) Doppler demodulator, to calculate the spectral content and to display a plot thereof in real time.

II To develop software to store, load and plot blood velocity waveforms.

III To develop software which will allow an operator to analyse acquired blood velocity waveforms in terms of the pulsatility index (PI) and an equivalent 3rd order electrical model.

IV To use the system in a pilot study to measure the parameters of normal subjects and patients presenting with lower limb vascular disease. The latter group will also be assessed by other clinical means such as arteriography.

CHAPTER TWO

2.1 LITERATURE SURVEY

2.1.1 SAMPLING AND SIGNAL PROCESSING

The continuous wave ultrasound unit, with user selectable frequency of five or ten megahertz (MHz), which was used during this project uses phase quadrature demodulation to output two demodulated signals which differ in phase by 90°. The frequency shifts are within the audio spectrum. Accurate, stable spectral estimation of these Doppler shifted frequencies is of vital importance.

If the demodulated signal is divided into segments, each $N\Delta t$ long (N = number of data points in each slice and sampling rate = $1/\Delta t$) then the frequency content of the signal may be calculated for these N points using the Fourier transform. However, if one considers the random nature of the signal under investigation it becomes evident that the Fourier transform may be inadequate. In the field of signal processing this problem is partly overcome by calculation of the power spectral density (PSD). The power spectral density of a stationary time signal is defined according to the Wiener-Khintchine theorem as the Fourier transform of the autocorrelation function. The solution is best illustrated if one considers the periodogram approach where the spectrum is averaged over a number of short consecutive segments taken from the time signal.

We have seen that the capture and processing of data in real time establishes the need to isolate time frames within a continuous process. The difficulty in calculating the PSD is therefore that the complete time domain function is not available and an estimate of the autocorrelation function has to be made.

A possible solution is to utilise a long time window but the problem is complicated by the non-stationary nature of the Doppler Ultrasound signal. The signal can be considered to be quasi-stationary for a period of no greater than 10ms (Evans and Schlindwein 1989). To apply the PSD theorem the time window is therefore limited to this time duration.

As it follows intuitively that the signal is not zero outside this window the autocorrelation function needs to be adjusted. Spectral methods, such as the periodogram which incorporates a Fast Fourier Transform (FFT) calculation, do not allow for this phenomenon and therefore have a relatively large variance. The parametric spectral techniques, such as autoregressive modelling, use the available autocorrelation values to form a prediction of the points outside the given time window.

An alternative for improving the PSD estimate is to apply a weighting function to the time signal before calculating the autocorrelation coefficients. This has the effect of giving a greater significance to the initial autocorrelation coefficients when the data overlap is a maximum. This process of assigning a relative weighting to the autocorrelation coefficients is commonly employed and can also be found when inspecting the following equation for a biased autocorrelation estimation.

Biased autocorrelation $R_{xx}(m)$ of discrete time sequence $x(n)$ containing N samples is :

$$R_{xx}(m) = \frac{1}{N-m} \sum_{n=0}^{N-|m|-1} x(n)x(n+m) \qquad (2.1)$$

for $m = 0,1,2...N-1$

When the PSD is calculated using the periodogram approach (magnitude squared of the Fourier transform) it is common practice to apply a weighting window to the time signal before calculating the PSD. The influence is slightly more intuitive. Digital time domain waveforms assume a periodic extension of the sampled window. Consider an example of a sampled sine wave. If the sampled window does not exactly overlap a whole number of signal wavelengths, an artificial discontinuity is introduced which manifests as undesirable sidelobes in the frequency domain. By applying a weighting window these discontinuities are suppressed at the expense of signal distortion. Examples of time weighting windows are the Hamming- and Blackman windows.

Kaluzynski (1989a) has researched the different spectral estimation methods as applied to Doppler ultrasound. His studies included the periodogram, autoregressive (AR) modelling, the maximum likelihood method and the

Wigner-Ville distribution. He concluded that for Doppler spectra the method of maximum likelihood and AR spectral estimation produced the best results.

The selection of the order of the AR process is presented in a follow up study done by Kaluzynski (1989b) (See also Evans and Schlindwein 1990). The study described in this thesis concentrates on AR spectral estimation, because of its relative superiority as a method of spectral estimation and the improvement in spectral resolution obtained (Kaluzynski 1989a).

The principle behind AR spectral analysis is a simple yet powerful one. Consider a system with input $p(z)$, output $x(z)$ and system transfer function $H(z)$, such that

$$H(z) = \frac{x(z)}{p(z)} \qquad (2.2)$$

Assume that we had measured the output signal, $x(z)$, and we propose to calculate the spectral content thereof. AR spectral analysis seeks to find that system , $H(z)$, which when driven by Guassian white noise, $p(z)$, delivers an output equal to $x(z)$. Note that the process is firstly a means of system identification. Using our acquired knowledge of the system, $H(z)$, we can estimate autocorrelation points outside that region from which we have taken direct measurements (With a unit function as input and the system transfer function, $H(z)$, known we can calculate the output to any number of points by application of the discrete difference equation.). Spectral analysis is completed simply by the application of the Wiener-Khintchine theorem i.e. the Fourier transform of the autocorrelation sequence.

Evans and Schlindwein (1989) proposed a real-time autoregressive spectrum analyser based on a TMS 320C25 digital signal processing processor. The AR spectral estimation technique that was employed proved computationally very efficient. Evans and Schlindwein highlight the inherent trade off between sampling frequency, window length and processing power. During system optimisation careful consideration is given to allow all frequencies under consideration to develop in the given time frame to improve the statistical properties of the estimate.

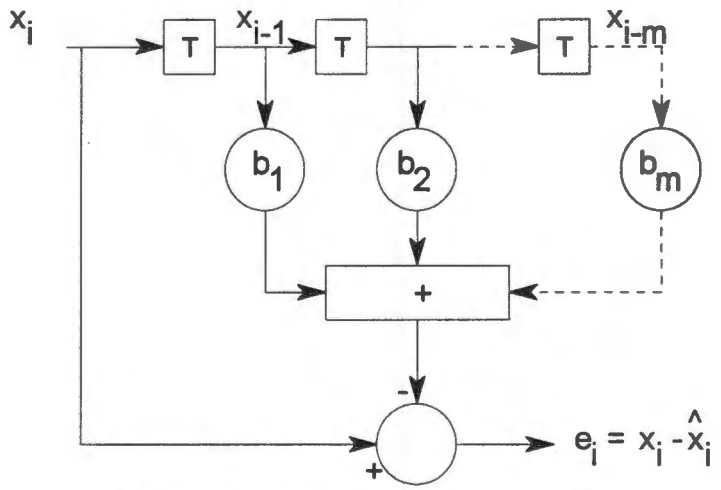
Evans and Schlindwein (1990) applied four different criteria for the selection of the order i.e. first zero crossing (FZC), final prediction error (FPE), Akaike's information criterion (AIC) and the Criterion autoregressive transfer-function

(CAT). The study concluded that in 98 % of the 1280 frames analysed the order selected was ten or less and that for a time domain window of 256 samples the FPE would suffice as selection criteria for the process order.

2.1.1.1 AR SPECTRAL ESTIMATION

As can be seen from the preceding paragraphs AR spectral estimation plays an important role during waveform analysis. Although the mathematical foundation of this technique is well documented it is important to consider briefly the underlying theory behind this approach (Readers who are not familiar with parametric estimation techniques may wish to skip this section).

The Durbin algorithm used during the AR process is a recursive algorithm for performing least square estimates (Giordano and Frank 1985). It allows the received signal to be estimated from a sequence of previously received signal samples. **Figure 2.1** serves to illustrate.



Linear filter as a tapped delay line
figure 2.1

Assume that we have an all pole system. A signal is observed and it consists of the transmitted signal plus noise. An estimate of the received signal may be obtained from delayed versions of the received signal linearly weighted by prediction coefficients. If we minimise the mean square error between the received signal and its estimate then we are left with a set of equations whose solution is the prediction coefficients.

Let x_i and s_i denote the complex low-pass equivalent of the received and transmitted signals taken at the *i* th sample respectively. The received signal is expressed as

$$x_i = s_i + n_i \tag{2.3}$$

for $i = 1,2,\ldots N$

where n_i is assumed to be stationary white noise and N is the number of signal samples. Assuming that the signal is stationary, x_i, may be predicted from x_{i-1}, x_{i-2}, ..., x_{i-M}. It follow that

$$\hat{x}_i = \sum_{m=1}^{M} b_m x_{i-M} \tag{2.4}$$

where b_m are the coefficients of the linear predictor, M is the filter order and $\hat{x}_i$ is the predicted value of x_i.

$$\begin{aligned} P_M &= \varepsilon\left[\left|e_i\right|^2\right] \\ &= \varepsilon\left[\left|x_i - \sum_{m=1}^{M} b_m x_{i-m}\right|^2\right] \end{aligned} \tag{2.5}$$

where e_i is the error signal at the *i* th sample, ε is the ensemble operator and P_M the mean square estimate of the error signal.

$$e_i = x_i - \hat{x}_i \tag{2.6}$$

Since the estimate of the received signal attempts to remove the predictable portion of the received signal, the error signal is approximately a white noise process. Equation (2.6) can be expressed as

$$e_i = \sum_{m=0}^{M} a_m x_{i-m} \tag{2.7}$$

where a_m are the prediction error filter coefficients given by

$$a_0 \equiv 0, a_m = -b_m \quad m \neq 0 \tag{2.8}$$

Substituting of (2.8) into (2.5) leads to

$$P_M = \sum_{m=0}^{M}\sum_{j=0}^{M} a_m a_m^* \varepsilon\left[x_{i-m} x_m^*\right] = \sum_{m=0}^{M}\sum_{j=0}^{M} a_m a_j^* r_{j-m} \tag{2.9}$$

where

$$r_{j-m} = \varepsilon\left[x_{i-m} x_{i-j}^*\right] \tag{2.10}$$

is the ensemble average autocorrelation coefficients. Minimisation of P_M with respect to the predictor coefficients a_l^* leads to the set of linear equations.

$$\frac{\partial}{\partial a_l^*} P_M = 2\sum a_m r_{l-m} = 0 \quad l = 1,\ldots,M \tag{2.11}$$

Equation (2.11) can be re-expressed in a form known as the Yule-Walker equations.

$$\sum_{m=1}^{M} b_m r_{l-m} = r_l \quad l = 1,\ldots,M \tag{2.12}$$

The Durbin algorithm (Code implementation shown in **Appendix III** L4.0392) is an order recursive method for solving (2.12). That is, it solves the coefficients of an M-order predictor recursively from the coefficients of an $(M-1)$-order predictor.

Assuming the input samples have zero mean Eq. (2.12) may be solved by first calculating the autocorrelation estimates.

$$R_l = \frac{1}{N}\sum_{r=0}^{N-l-1} x_r x_{l-r}^* \quad l = 0,1,\ldots,M \tag{2.13}$$

For a stationary process the covariance matrix $\{r_l\}$ is Toeplitz, so that the coefficients b_m, are recursively computed as

$$b_{M,p} = b_{M-1,p} - b_{M,M} b_{M-1,M-p}^* \tag{2.14}$$

for $p = 1,2,\ldots M-1$ where

$$b_{M,M} = \frac{A}{B}$$

$$A = R_l - \sum_{l=1}^{M-1} b_{M-1,l} R_{M-l} \tag{2.15}$$

$$B = R_0 - \sum_{l=1}^{M-1} b^*_{M-1} R_l \tag{2.16}$$

If the Fourier transform of the prediction coefficients is defined as

$$A(f) = \sum_{i=0}^{M} a_i e^{-j2\pi ifT} \tag{2.17}$$

then the PSD may be estimated as

$$\hat{P}(f) = \frac{c}{A(f)A^*(f)} \tag{2.18}$$

where c is a constant and T the sampling frequency.

A more explicit representation of the spectrum in terms of an all pole (AR) model can be obtained by substituting (2.17) into (2.18) using $a_0 \equiv 1$.

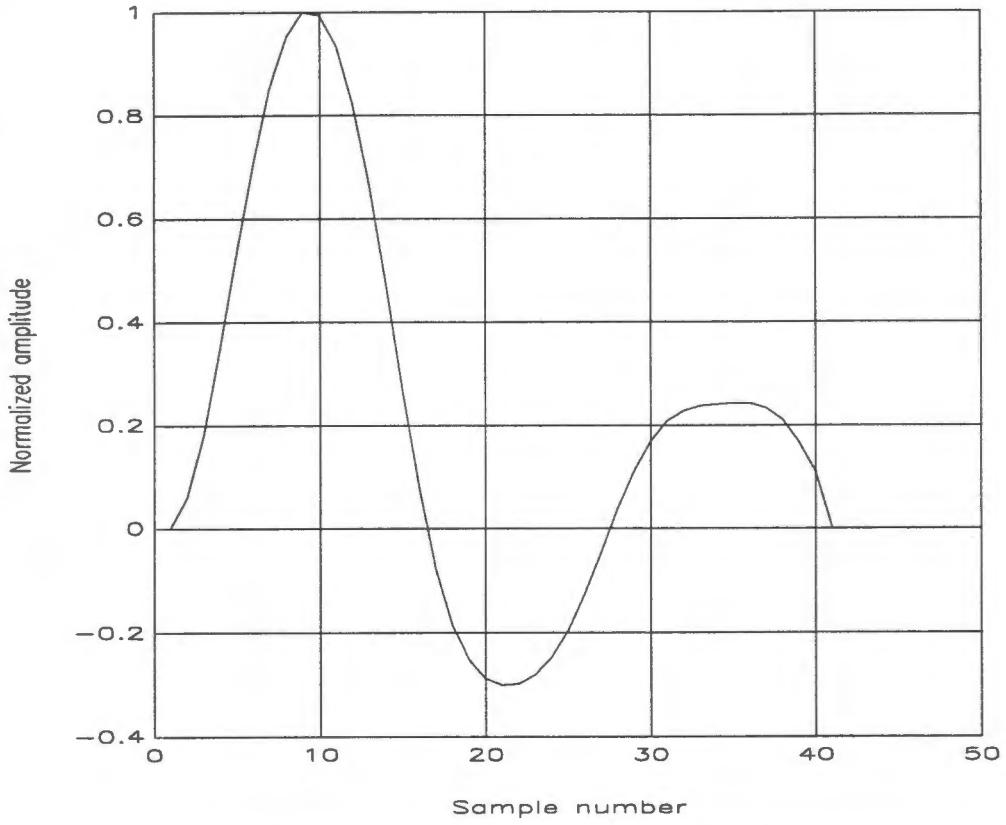
$$\hat{P}(f) = \frac{c}{\left| 1 + \sum_{i=1}^{M} a_i e^{-j2\pi ifT} \right|^2} \tag{2.19}$$

The power spectral estimate is then obtained using Eq. (2.14) with $b_m = -a_i$.

2.1.2 BLOOD VELOCITY WAVEFORM ANALYSIS

Following the process of PSD estimation the blood velocity waveform is constructed through the calculation of a representative blood velocity value for each consecutive time frame. The two methods most commonly employed are the mean- and peak velocity processors (Skidmore et al 1989). The mean velocity is calculated through the identification of that point, in the frequency domain, which has an equal amount of energy to the left and right. Note that the absolute value thus calculated has only a relative importance because of the angle dependence as explained in Appendix I. A criticism of the peak velocity process is the inherently noisy environment in which measurements are performed. This can be clearly understood when considering the turbulent nature of blood flow.

The velocity waveform thus calculated has, given a normal arterial pathology, a characteristic tri-phasic shape. The waveform shape changes in the event of arterial disease and/or under conditions which act as a disturbance to normal flow.



Tri-phasic shape of a normal velocity waveform
figure 2.2

Early studies have shown the loss of pulsatility of a velocity waveform in the presence of arterial disease (Fitzgerald et al 1971). The pulsatility index (PI) as a quantitative measure of disease was first described in terms of the signal's Fourier components, but the technique was later simplified (Gosling and King 1974) to the ratio of the peak to peak and mean velocities.

The variation of the velocity profile in the post-stenotic field gives rise to a broadening of the spectral estimate. The use of the spectral broadening index (SBI) as a quantitative index of turbulent flow has however not been popular. This is partly due to the dependence of the frequency variation on the probe angle and the inherent shortfall of spectral methods applied. It is shown (Wijn et al 1987) that the SBI depends strongly on the insonation of the vessel and

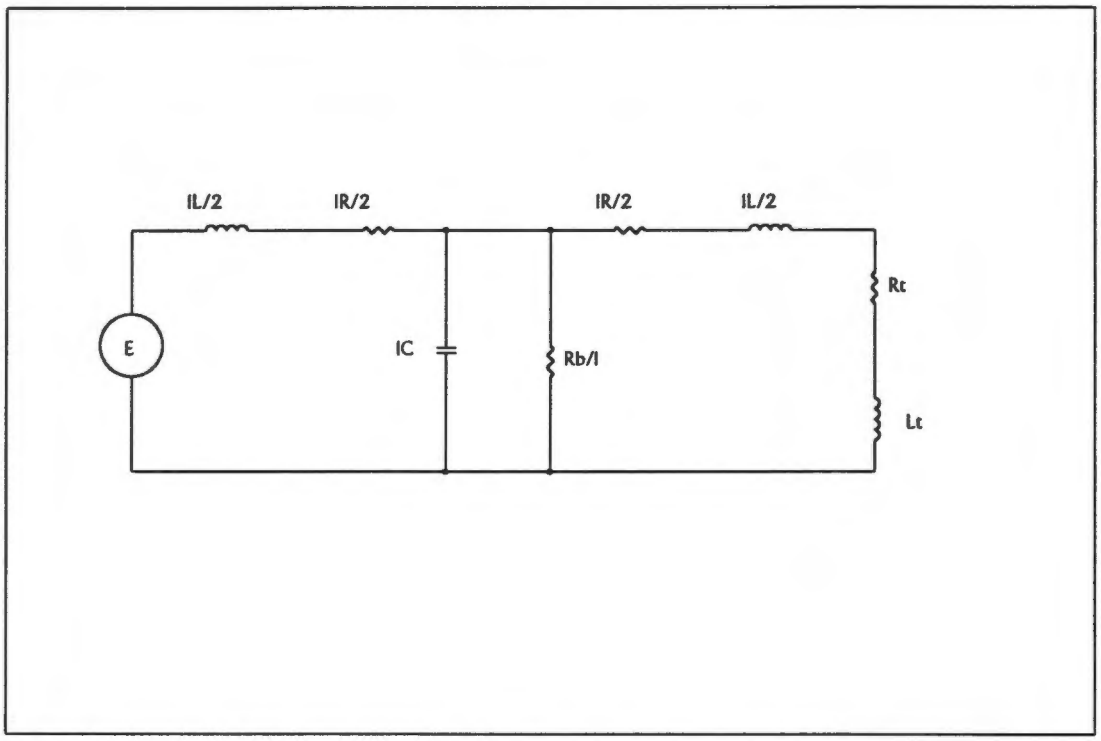
geometry of the ultrasound beam. In intra-operative measurements a large variation is possible due to near field (Fresnel region) operation of the Doppler probe.

The transfer function approach (Brown et al 1978) relates the frequency transforms of measurements taken at two separate sites, namely the proximal end of the femoral and popliteal arteries. The former is considered as the input to the arterial segment and the popliteal waveform the output. A second order system was formulated to describe the blood flow in the arterial segment and the roots of the characteristic equation solved. The pole positions are indicative of the segment's physical state. A severe criticism of this approach is that the input waveform does not have a sufficiently broad bandwidth to be used for system identification and the non-linearity in the arterial segment can lead to the prediction of an unstable system, which is clearly not the case. As such the use of this method is not popular in clinical application.

The method of principal component analysis (PCA) is similar to Fourier analysis in that the waveform is broken down into its principal components, but a distinct difference exists in that the components are not necessarily sinusoidal. A minimum number of components is sought to best describe the velocity waveform. As this complex method has little advantage over the simple PI it too has remained dormant in the research environment.

More complex methods, such as the Laplace transform analysis (LTA) (Skidmore 1979), have been developed which relate the waveform shape to the arterial physiology. In LTA a simple elastic tube model was developed to represent the arterial tree from the heart down to the periphery. The electrical analogy of this model was then used to develop an equation which relates the flow of current (analogous to blood flow) to a stimulating voltage (analogous to cardiac pressure pulse).

Figure 2.3 shows the simple electrical analogue model of the arterial system. L, R, and C represent the inertia of blood, resistance to flow, and wall compliance, respectively, of the main aorto-iliac trunk of effective length l. R_b represents the resistance of all vessels which run off the main trunk. R_t and L_t are the resistance to flow and inertia in the terminal branch which represent the runoff resistance distal to the inguinal ligament. E represents the impulse delivered by the heart.



Electrical analogue of the arterial system
figure 2.3

With numerous simplifications the model can be reduced to a third order system where parameters relate to arterial wall elasticity, distal impedance and proximal arterial diameter. If the input voltage to the model is assumed to be an impulse then the pole positions on the Argand diagram can be used as an indication of arterial disease.

The system transfer function may be written as

$$H(s) = \frac{k}{(s^2 + 2\alpha s + w_0^2)(s+\gamma)} \qquad (2.20)$$

where :

k = constant (2.21)

$$\alpha = \frac{R}{2L} + \frac{1}{2CR_b} = \text{WoLTD} \qquad (2.22)$$

$$w_0 = \frac{1}{l}\sqrt{\frac{2}{CL}} = LTE \qquad (2.23)$$

$$\gamma = \frac{R_t}{L_t + \frac{Ll}{2}} \qquad (2.24)$$

and

$$LTR = |\gamma| \qquad (2.25)$$

The parameters of the model are calculated from a measured flow waveform by first calculating the spectral content of the waveform and then using a curve fitting procedure in the frequency domain. This model attempted to isolate the different physiological variables and as such seems a very attractive option but several authors have questioned the validity of this approach (Law et al 1984 and Johnston 1984). It was shown by these authors that the flow waveform is a function of peripheral resistance but that the model does not account for continuous flow during the cardiac cycle, as might be the case with distal vasodilation.

A follow-up study by Skidmore et al 1989 conceded that further model refinement should be done but proved that the presence or absence of superficial femoral occlusion did not affect the prediction of iliac disease. The LTA proved a success in detecting stenoses of greater than 50 % with a sensitivity of 92 %. Two areas of concern are stated. First, the LTR variable was proven to be without diagnostic value and secondly that the curve fitting procedure used has difficulty in dealing with positive or negative offsets at the origin i.e. the problem of continuous flow during a given cardiac cycle.

Capper et al (1986) employed an analog system to obtain the mean frequency waveform for analysis. The Laplace Transform parameters and pulsatility index are calculated and compared with angiographically determined aorto iliac disease and pull through measurements. From this pilot study LTD performed better than both the PI and the ankle systolic pressure index (ASPI = systolic pressure at ankle / systolic pressure at brachial artery) in separating significant stenosis from mild disease.

Meara (1984) presented a non-linear regression technique to analytically solve the characteristic equation as proposed by Skidmore. An inverse Laplace transform is applied to Skidmore's frequency based equations. The discrete time equivalent equation thus obtained is used in this study and will be discussed more comprehensively in following sections.

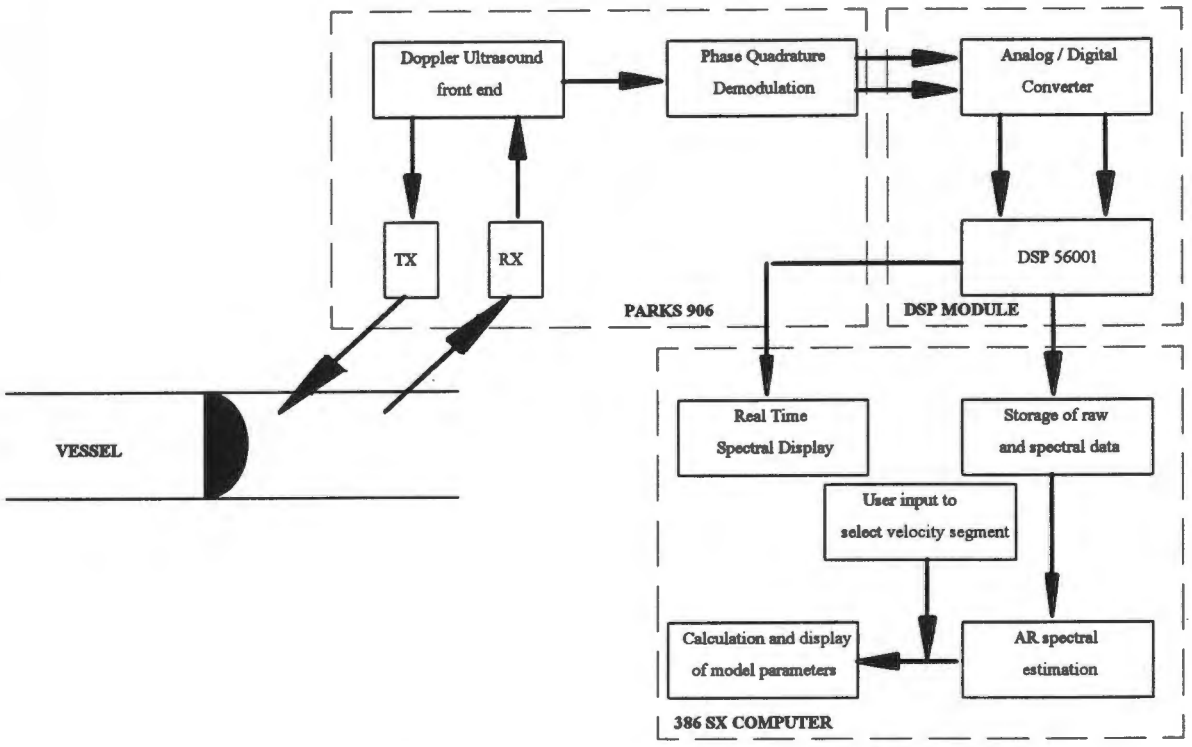
Other parametric methods such as autoregressive (AR) models have proved useful in classification of disease (Alessio et al 1983), being specific in discrimination between signals relating to different pathological states but they have the inherent shortcoming that the parameters extracted have no known physiological meaning.

A great deal of research has been done at the UCT/Groote Schuur Department of Biomedical Engineering and at the Faculty of Medicine, University of Stellenbosch (Bunn and Guelke 1983), into flow in compliant tubes in terms of system parameters such as tube compliance, wall losses and fluid inertia. The approach adopted by these authors is that of an electrical transmission line analogy with specific reference to wave propagation as a function of frequency. This approach has been used to explain cochlea function (Bunn and Guelke 1983, Guelke and Bunn 1984) and the input impedance of the lungs (Capper et al 1990). Applying this resource pool of knowledge to femoral studies necessitated using models which have a direct relationship to arterial pathology.

CHAPTER THREE

3.1 HARDWARE

As real-time implementation and processing is a high priority the hardware has been chosen to facilitate computational power and speed. A schematic representation of the development system is shown.



System data flow diagram
figure 3.1

3.1.1 ULTRASOUND UNIT

The CW Doppler input to the system is from a PARKS 906 bi-directional unit. This unit serves to present two demodulated phase quadrature signals to the DSP unit from which the forward and reverse flow components are calculated. Integrated in the PARKS 906 is the demodulation from the high frequency carrier to baseband audio.

3.1.2 MICRO-CONTROLLER ENVIRONMENT

The DSP board, designed to fit into an IBM compatible PC bus, fulfils the digital processing needs of the system and as such plays an integral part in real-time implementation. The basic composition is a four-channel multiplexed input, 12-bit 1MHz A/D (analog to digital) converter, on board memory and a Motorola DSP56001 fixed point processor. The dual natured architecture of the processor (separate program and data memory space) and the parallel nature in which the program controller, the address generation unit and the ALU operates facilitates benchmarks such as a 1024 point complex FFT calculation in 3.2 ms.

3.1.3 HOST PLATFORM

A PCM 386SX performs the co-ordinating function of the system. This 16MHz PC utilises the following configuration; a 40Mb HDD, 1.2Mb FDD, 2Mb 0 wait state Ram, 16-bit VGA graphics card with 512kB Video Memory. A multi-frequency colour VGA monitor with a resolution of 640 X 480 X 256 colours serves as the VDU.

3.2 SOFTWARE IMPLEMENTATION

3.2.1 MICRO-CONTROLLER ENVIRONMENT

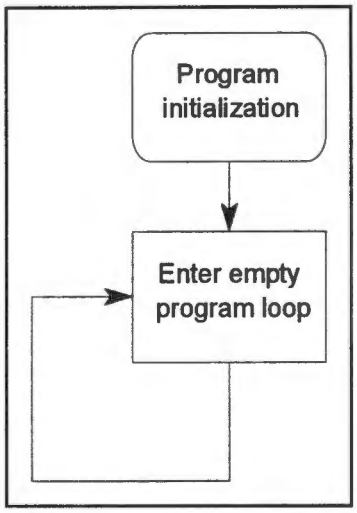
Real or near real time spectral analysis requires considerable processing power which is made feasible with high speed DSP processors. The Motorola DSP56001 fixed point processor used in this project is rated at 10 MIPS (Million instructions per second). To ensure that the greatest speed advantage is attained, software development is done in Motorola DSP56001 assembler.

A complete listing of the DSP assembler code developed is shown in Appendix III. Note the line numbering (Lx.xxxx) system employed. Program comments and debugging lines start with a '%' for MATLAB files, ';' for the assembler files and '//' for the C source code. An explanation of the program philosophy follows.

The process is entirely interrupt driven as a consistent timing sequence is needed. To this end three separate interrupts are used (Int A - L0.0130, Int B - L0.0149, FrameR Int - L0.0169). Int A and Int B are driven by an external source (8254 programmable interval timer) and the FrameR interrupt is derived from the DSP processor itself. Consider carefully the IPL (Interrupt Priority Level) given to them. The DSP processor has three distinct interrupt priority

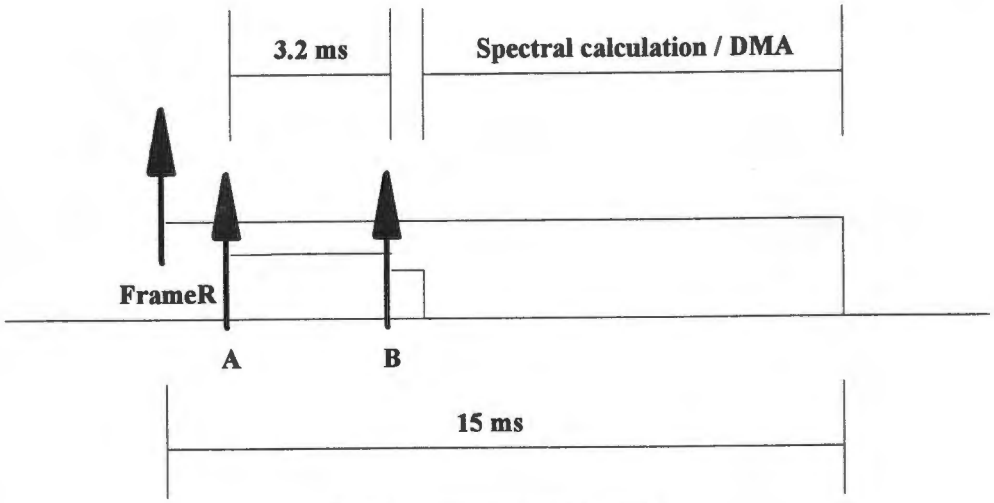
levels. Levels zero through two may be selected by the user. Masking of all lower level IPL's are employed with IPL3 (e.g. a processor reset) having the highest priority.

After program initialisation the program enters an empty loop which allows the interrupts to take control. The following flow diagram serves to illustrate the global perspective.



DSP program loop
figure 3.2

As the sampling frequency is known a priori the FFT-, Hamming window- and carrier lookup table are calculated during program initialisation. The carrier table is used during the demodulation phase of the phase quadrature channels (Appendix II). At this time the interrupt priority is set to IPL3 (L0.1178) which disables all user selectable interrupts. The IPL levels are then set as follows FrameR - IPL2, Int A - IPL0, Int B - IPL0 and the host port to IPL1 (L0.1186). Before the main program loop (L0.1210) is entered the interrupt priority is lowered to IPL2 to allow FrameR to start. The sequence which follows is illustrated below.



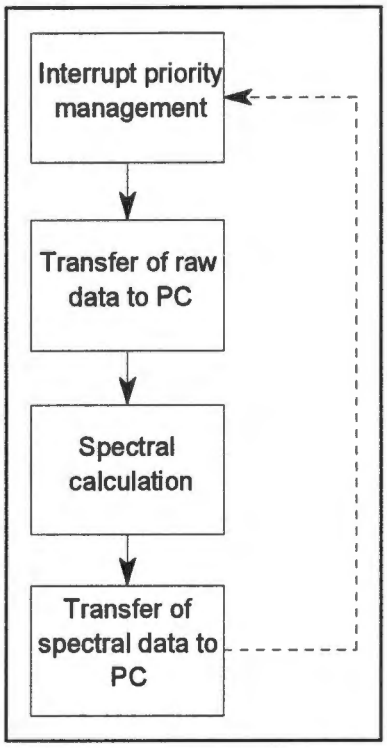
DSP interrupt schedule
figure 3.3

The FrameR interrupt, which is driven by the internal DSP clock, has a period of 15ms. The interrupt initiates the sampling process by setting the 8254 programmable interval timer (which controls Int A and Int B) and unmasking the lower priority interrupts. This enables Int A to start the sampling sequence. The FrameR interrupt then enters an idle state waiting for the IPL to be raised to IPL2 (L0.0175).

Int A samples both input channels into X and Y memory space respectively. During this process the software gain control is introduced through the right shifting of sample data (L0.0135 and L0.0143).

Note that when an interrupt occurs the IPL is automatically raised to disallow any interrupt of the same or lower IPL to interrupt. This process is reversed on interrupt exit (e.g. during the sampling process the IPL alternates between IPL0 and IPL1 when the Int A sequence is first entered, executed and then returned from). When Int B occurs after 3.2ms (i.e. between sampling interrupts) the sampling process is effectively stopped. To prevent Int A from restarting after Int B has finished, we change the exit priority (to IPL2) as stored on the system stack (L0.0150 - L0.0156).

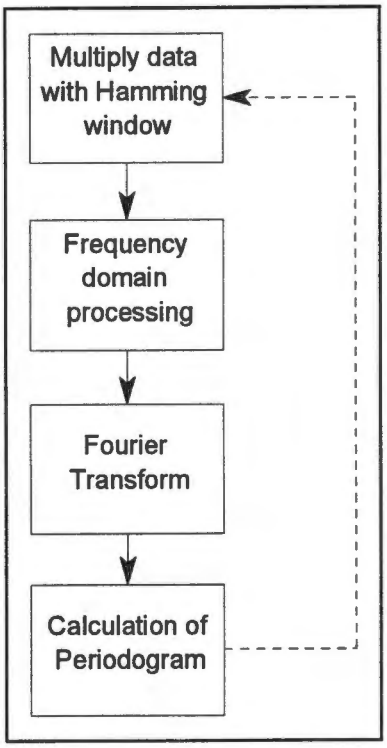
This allows the FrameR interrupt to continue and start the spectral calculation and DMA processes.



FrameR interrupt
figure 3.4

To optimise the DMA transfer process (L0.1275) the raw data is transferred as integer values. No signal resolution is lost during this process as the A/D converters are themselves 12-bit entities. The handshaking between the host and the DSP is implemented through the use of flags in the HCR (Host Control Register) and the HSR (Host Status Register). This is an important process (See DSP macro at L0.1275) which ensures that both the PC and the DSP is at all times aware of what the other party is doing. Consider as an illustration of a potential pitfall the transfer of raw data when the PC expects the spectral values.

The spectral calculation (L0.1360 - L0.1467) comprises the following steps :

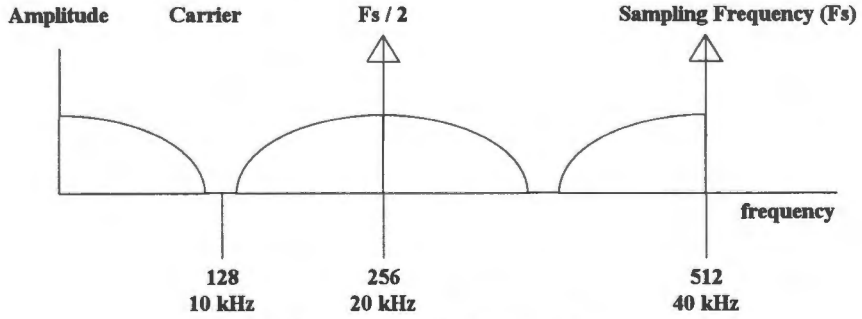


Spectral process flow

figure 3.5

Both direct and quadrature channels are multiplied with a Hamming window to suppress unwanted sidelobes (L0.0356). Frequency domain processing (L0.0382 and L0.0405) follows. This technique (Appendix II) combines the information embedded in the separate input channels into a single memory space. The data is now ready for the application of a single FFT operation which contains all the relevant directional information.

The calculation of the PSD starts with the zero padding from 128 to 512 data points. This process interpolates between data points in the frequency domain. Note that no information is lost during this process. Consider the following diagram :



Spectral distribution

figure 3.6

The replication of the positive and negative frequency bands (Nyquist theorem) and the influence of the frequency domain processing (Appendix II) leaves us with 128 positive and 128 negative data values, offset from the carrier frequency, to display. An FFT and magnitude squared (Periodogram) operation completes the process. The spectral values are then right shifted as only a 8 bit resolution is needed for the spectrogram display. (256 discrete levels proofed sufficient for visual discrimination). The advantage gained in this step is the greater speed of 8 bit DMA.

3.2.2 HOST PLATFORM

The PC fulfils a co-ordinating role and provides the user interface. Software development is done in Turbo C and control over the DSP card is maintained from within this environment.

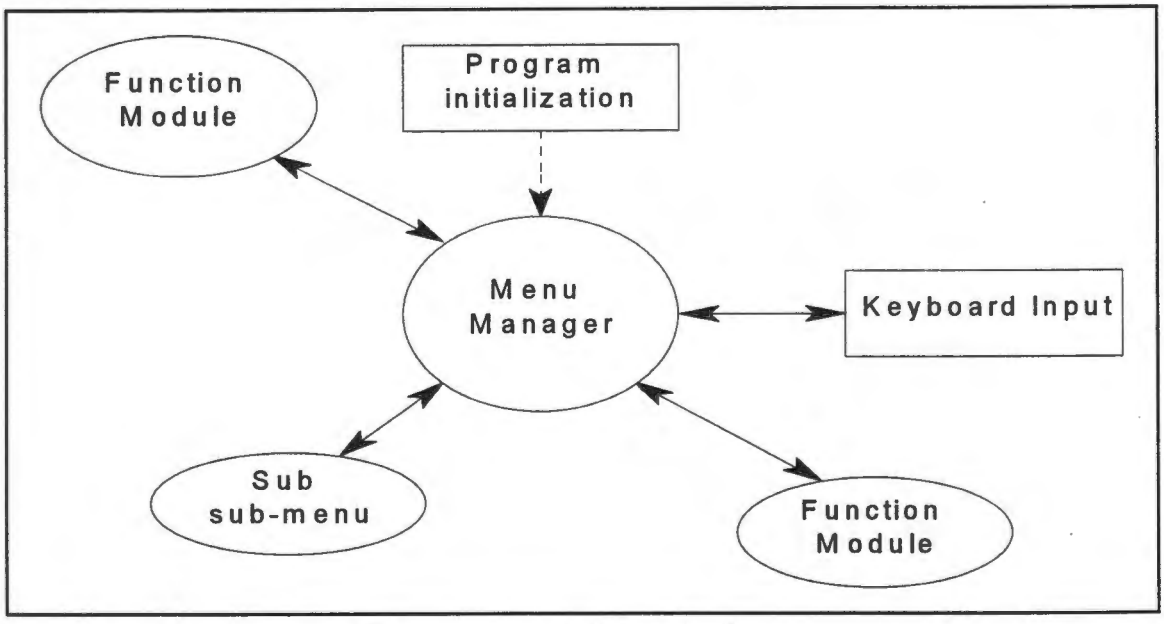
The relative inability of a PC as a real time instrument provided a considerable challenge in ensuring that the DSP be utilised to its full potential. The main culprit proved to be the limited graphics capabilities of the Paradise video adapter. From the outset a decision was made to keep the adapter in graphics mode (640 X 480 X 256 colours) as opposed to continually switching between text and graphics mode. Extensive use is made of the Borland's Graphic Interface (BGI). As the selected mode is not supported by the standard BGI drivers a locally developed software driver specific to the Paradise adapter is used (Paradise.bgi).

The selected mode allows the manipulation of the colour palette and the implementation of a look-up table in which sixteen amplitude ranges are represented by unique colour/intensity combinations. Four colours (Blue, Green, Yellow and Red) are used; each having four different intensity levels. This corresponds to the sixteen amplitude levels. The look-up table is defined with index into the palette starting at colour number sixteen so as not to interfere with the standard IBM defined range for EGA and VGA; zero to fifteen.

To ensure portability and provide a platform for future research work a protocol was established by which data capture and storage could be done. As an aid to this development research done by Evans and Schlindwein (1989) provided valuable input.

A scrolling display provides the user with a real time visual feedback. Upon successful capture the option is provided for permanent storage. A retrieval for comparison or further inspection can be done at any given stage. A record is kept of raw data and the calculated spectrum. This has the advantage of allowing future studies to compare different methods available for spectral analysis and waveform processing.

The system software architecture evolves around a recursive menu manager function (L1.0296 - L1.0324) which, after system initialisation, is called with argument 0 (L2.0076). The significance of this integer is best explained if one considers the MenuWindow struct (L3.0027). All displayable menu items are numbered and provision is made for calling sub levels menus and/or system functions. Menu 0 (L1.0024) is the main menu of the application and from here, given the correct user input, sub menu or implicit function calls are done. As an example consider L1.0028 where the either the **Go** function or the **Menu_Man** function (with an integer argument (L1.0029)) is called. This approach facilitates the addition of functionality with a minimum programming effort. A graphical system flowchart is given below.



Program module control
figure 3.7

When considering system data the most important to note is that two multi dimensional arrays are kept for the forward and reverse channels respectively. A global variable **row** acts as an index to both and is incremented after each

time frame is captured, using the C % (modulus) operator, to allow for a circular buffer structure (L2.0736). All functions have been thoroughly documented (see Appendix III) and can be best understood using this reference provided.

3.2.3 CODE DEVELOPMENT FOR MATLAB

Numerical analysis of the captured data requires complex and intensive computational power. This function is fulfilled by MATLAB. The package supplied by THE MATHWORKS CORPORATION has, in addition to numerous built in functions, a programming structure which allows the user the flexibility to develop his/her own algorithms.

It is proposed that once the waveform analysis is secure and concise, coding will be done in C or DSP assembler and incorporated into the stand alone system. Code development for MATLAB (L4.0000 - L4.0807) is done in a C-like structure which should simplify any cross coding process.

The steps needed to perform waveform analysis (L4.0010 - L4.0029) on given data are as follows :

1. Load raw data as captured (L4.0591 - L4.0598)
2. **Calculate AR spectral estimation (L4.0049 - L4.0090)**
3. Determine blood velocity waveform (L4.0092 - L4.0112)
4. **Apply infinite impulse response (IIR) Yule Walker filter (L4.0740 - L4.0770)**
5. Select a velocity segment to analyse (L4.0686 - L4.0710)
6. Calculate discrete model parameters (L4.0114 - L4.0130)

Note : **Steps printed in bold** are optional.

The raw data is stored in a text file to allow for import into MATLAB. Although this is not optimal in terms of disk space used, DOS based applications are limited in their sharing of data. To partly compensate use any shareware application to compress data files which are not currently used.

The spectral estimation is performed using the well known Durbin algorithm (L4.0392 - L4.0440). The model order selection used is 15. Research done by various authors is used as a guide in this regard. As an alternative to this

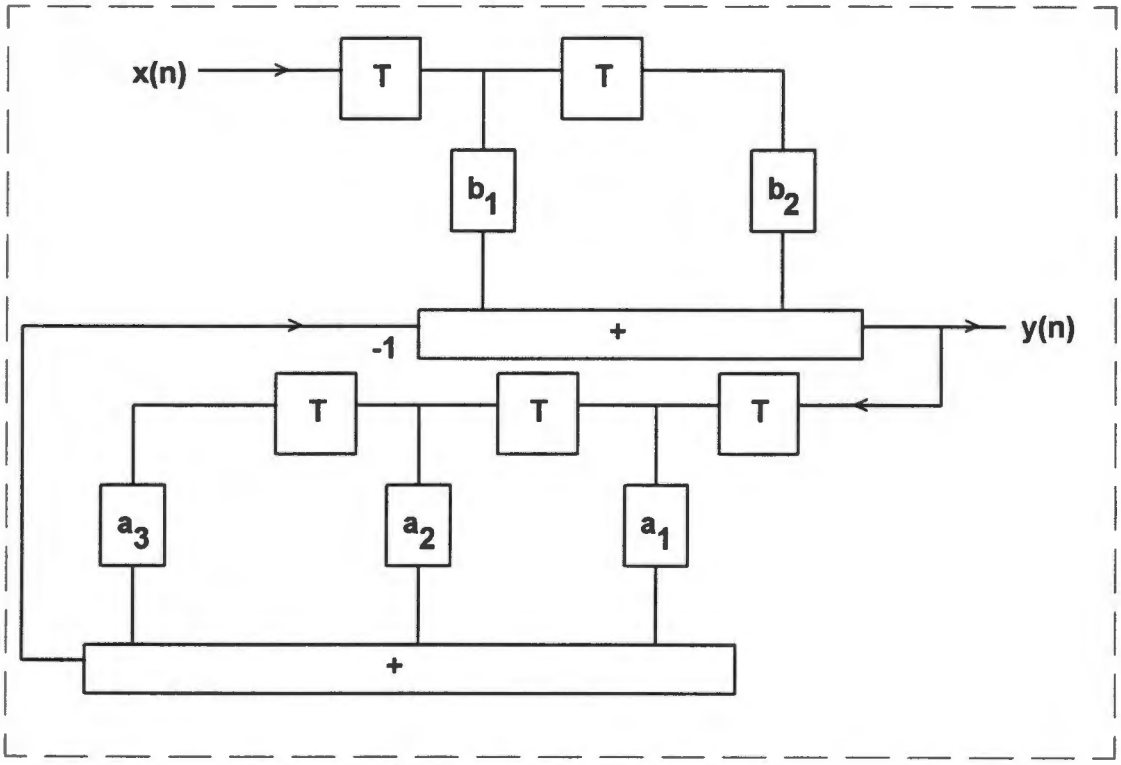
approach the periodogram calculated in real time may be used as a start off point. Pseudo 3-D plots of the calculated spectra are shown in the following chapter.

The mean blood velocity waveform follows from the spectrum. The manner in which we estimate the flow profile is to consider, for each spectral window, that point which has equal amounts of energy to the left and the right. Taking each point thus calculated we construct the waveform profile for the entire period under consideration.

The low pass filtering of the waveform is optional. It has greater importance if Prony's method of system identification is used (L4.0625 - L4.0684). This method is an attractive option for system identification as it assumes an impulse input. Unfortunately this approach is very susceptible to noise variations, a fact stated by Meara et al (1984).

Recall that the objective of the waveform analysis is to identify a system which when driven with an impulse input has a given waveform as output. We therefore need to extract a waveform segment which starts at zero to identify a causal system. (Unfortunately this is not always possible as some physiological phenomena dictate a continuous flow throughout the cardiac cycle.) Using user input as a guide, the start and end of a velocity segment is chosen. The segment amplitude is normalised to unity after subtracting any DC offset component; thus forcing a zero value start. This does not influence the pulsatility of the waveform and is accepted as such until more complete system models are defined.

The discrete equivalent of the continuous s-plane system proposed by Meara et al 1984 is



Discrete system model
figure 3.8

The system transfer function is :

$$H(z) = \frac{b_2 z^{-2} + b_1 z^{-1}}{a_3 z^{-3} + a_2 z^{-2} + a_1 z^{-1}} = \frac{Y(z^{-1})}{X(z^{-1})} \qquad (3.1)$$

Calculating the system parameters given the above model is done in an iterative manner. The difference equation form of the transfer function (L4.0136) is fed to the Nelder-Mead algorithm to calculate the unknown parameters. The algorithm calculates the system output and compares the fitted curve to the selected segment using a mean square error (MSE) as criteria. If not within a given tolerance level the parameters are adjusted and the process repeated.

An important point to consider is the initial guess value of the five unknowns. The theory of applied mathematics warns against local minima, a scenario where the algorithm finds a point other than the absolute minimum of the MSE. To overcome this difficulty the initial guess values are taken from the frequency domain Laplacian fit (L4.0132 - L4.0308), a process which in its own right is used for system identification (see Capper et al 1986).

CHAPTER FOUR

4.1 INTRODUCTION

A small pilot study was done on normal individuals and patients on whom peripheral angiograms have been performed. Mean blood velocity waveforms were gathered from twelve normal limbs and twenty-one limbs of patients presenting with lower limb disease. The normal subjects were all healthy non-smokers with no history or signs of arterial occlusive disease. The patients on the other hand all suffered from lower limb arterial disease ranging from about 30% to almost complete occlusion of these vessels. All the test procedures were completed by the technical staff of the Vascular Laboratory at Groote Schuur Hospital.

The chapter presents the results of the investigation and concludes with screen captures, as presented by our system, showing the results of three individuals.

4.2 METHODS

The subjects were all tested in the supine position. A short time period was taken to explain the procedure to them. The emphasis was to reassure the patients of the very low degree of discomfort which they will experience and as such help them to relax and to allow them to reach a steady heamodynamic state. The laboratory technologist applied the ultrasound gel at the site of the femoral artery pulse, slightly distal to the inguinal ligament. (This procedure ensures close coupling of the transducer probe and the measuring site.) With the computer display as visual feedback they captured three (3) seconds of waveform measurements The waveform analysis was then done off-line, a procedure which takes approximately six minutes per patient.

During the statistical analysis of the subject data, use is made of the Mann Whitney U-test as a quantitative measure of the separation between data groups. The formulae for this index is presented by Mendenhall et al (1986) as

$$Z = \frac{U_A - \left(\frac{n_1 n_2}{2}\right)}{\sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}} \qquad (4.1)$$

with

$$U_A = n_1 n_2 + \frac{n_1(n_1+1)}{2} - W_A \qquad (4.2)$$

where

n_1 = number of observations in sample A (Normals)
n_2 = number of observations in sample B (Patients)
W_A = rank sum of sample A

After calculation of the Z parameter the p value is read from the appropriate statistical map (see reference).

4.3 RESULTS

The following tables show the processed data for the control group of twelve normals and the data for the patient group consisting of twenty one individuals with angiographically determined aorto-iliac arterial disease.

	Frequency domain analysis (Laplace)				**Time domain analysis (Meara)**			
ID	LTE	LTD	LTR	MSE	LTE	LTD	LTR	MSE
n1	18.73	0.161	8.434	0.267	17.86	0.162	9.048	0.116
n2	19.12	0.270	12.91	0.527	16.15	0.174	2.533	0.440
n3	20.25	0.293	7.670	0.977	17.93	0.259	6.875	0.335
n4	17.74	0.128	3.758	0.827	16.77	0.128	3.130	0.353
n5	19.17	0.155	4.916	0.682	19.65	0.237	7.527	0.435
n6	17.35	0.146	3.742	0.662	18.06	0.185	4.701	0.583
n7	20.91	0.353	7.650	1.597	19.74	0.371	8.086	0.145
n8	18.89	0.309	6.289	0.939	18.28	0.310	6.168	0.066
n9	19.63	0.227	5.817	1.290	17.84	0.214	5.527	0.366
n10	17.67	0.162	3.588	0.782	16.19	0.115	2.486	0.341
n11	21.21	0.217	4.862	1.112	18.22	0.193	3.734	0.380
n12	18.28	0.167	3.581	0.463	17.22	0.128	2.600	0.096

Processed data for normal group

Table 4.1

As seen from **Tables 4.1** and **4.2** (below) the fit proposed by Meara has on average a 64%((MSE ave. Meara) - (MSE ave. Laplace))/ (MSE ave. Meara))

advantage over the Laplacian fit in terms of the MSE (Where ave. = geometric mean). Therefore, using the time domain data the statistical properties of the LTD data segment are calculated as :

Mean : 0.206
Median : 0.189

	Frequency domain analysis (Laplace)				Time domain analysis (Meara)			
ID	LTE	LTD	LTR	MSE	LTE	LTD	LTR	MSE
p1	21.77	0.630	3.034	1.139	14.56	0.421	2.328	0.375
p2	26.16	0.799	7.790	1.953	12.76	1.000	41.64	0.820
p3	15.34	0.920	11.70	0.908	14.01	0.403	4.189	0.101
p4	11.76	0.879	34.85	0.674	14.50	0.482	6.107	0.101
p5	29.50	0.386	12.03	1.404	29.81	0.437	13.77	0.315
p6	19.60	0.438	37.10	1.248	19.67	0.419	30.81	0.142
p7	24.01	0.587	8.606	0.149	24.88	0.611	8.853	0.051
p8	16.14	0.875	457.5	1.109	15.99	0.837	442.9	0.106
p9	18.89	0.309	6.289	0.939	18.28	0.310	6.168	0.066
p10	30.24	0.485	14.93	0.231	28.85	0.456	13.87	0.016
p11	13.20	0.703	5.906	0.315	11.34	0.769	7.024	0.113
p12	9.450	0.972	110.2	0.651	10.04	0.998	38.8	0.053
p13	9.510	0.982	101.8	0.641	9.188	0.935	94.58	0.064
p14	18.05	0.512	4.308	0.595	17.47	0.453	3.873	0.045
p15	18.68	0.516	5.268	0.444	15.70	0.497	4.765	0.136
p16	12.69	0.813	62.59	1.884	19.50	0.831	12.43	0.255
p17	24.86	0.485	12.42	1.886	16.82	0.223	4.661	0.670
p18	12.39	0.885	2661	1.042	12.00	0.887	2696	0.149
p19	32.71	0.836	8.621	0.730	16.45	1.000	34.24	0.207
p20	13.86	0.487	714.6	2.074	12.65	0.487	722.5	0.659
p21	12.05	0.645	72.23	1.211	15.30	0.478	9.539	0.281

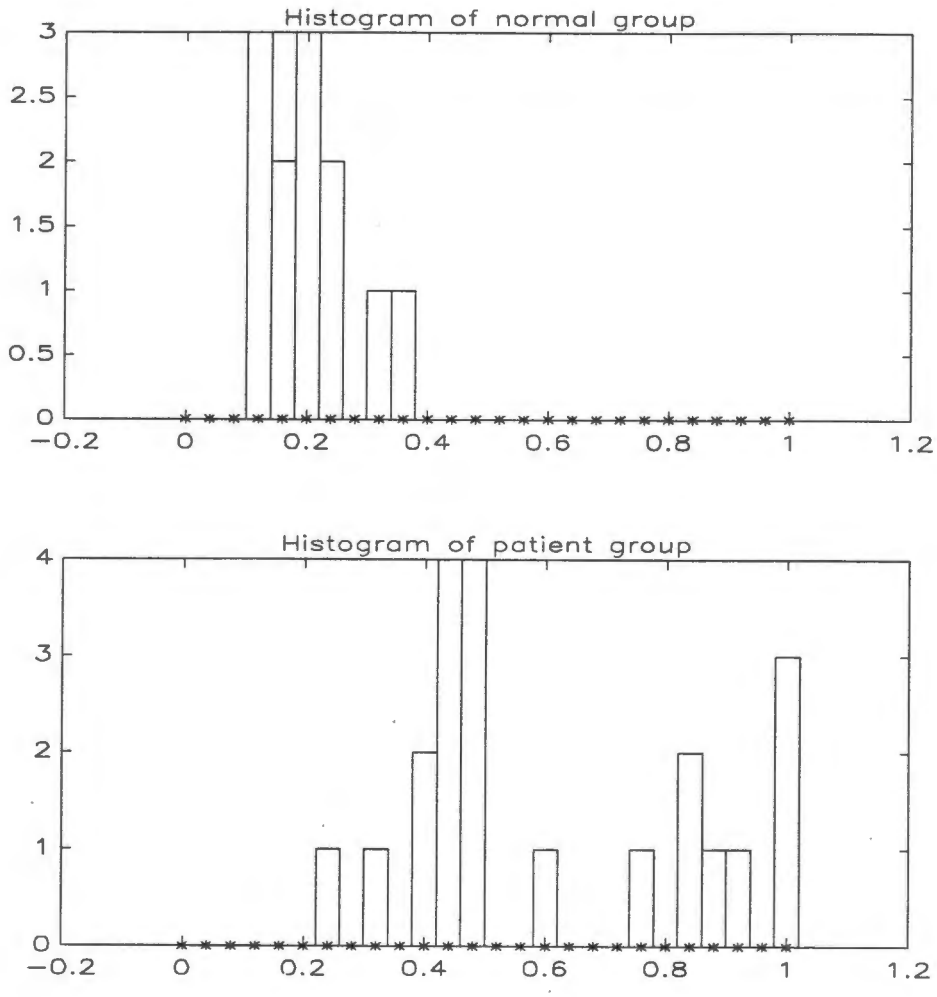
Processed data for patient group
Table 4.2

Using the time domain data the statistical properties of the LTD data segment are calculated as :

Mean : 0.616
Median : 0.487

The median and quartile range of the damping factor for the normal group is 0.19, 0.15-0.25 respectively, while that for the patient group is 0.49, 0.45-0.89 respectively. Separation between the two groups was tested using the Mann Whitney U-test and was found to be highly significant ($p < 0.001$). The

Laplace Transform Damping probability distributions of the two groups are shown in **figure 4.1**. (Where Y-axis = patient number, X-axis = LTD value).



Subject data distribution
figure 4.1

The following graphical displays are of screen captures as presented to the clinician. The first display is of the forward and reverse frequencies as captured. The blood velocity waveform and the curve fitted to this curve are shown in the graph titled **Impulse Response**. The system poles and zeros are shown on the right, both for the discrete z-domain and the continuous s-domain. The system parameters that were calculated are shown together with the values for the LTD, LTR, LTE and the PI. (The figure in brackets adjacent to the PI is the PI for the original unfitted curve)

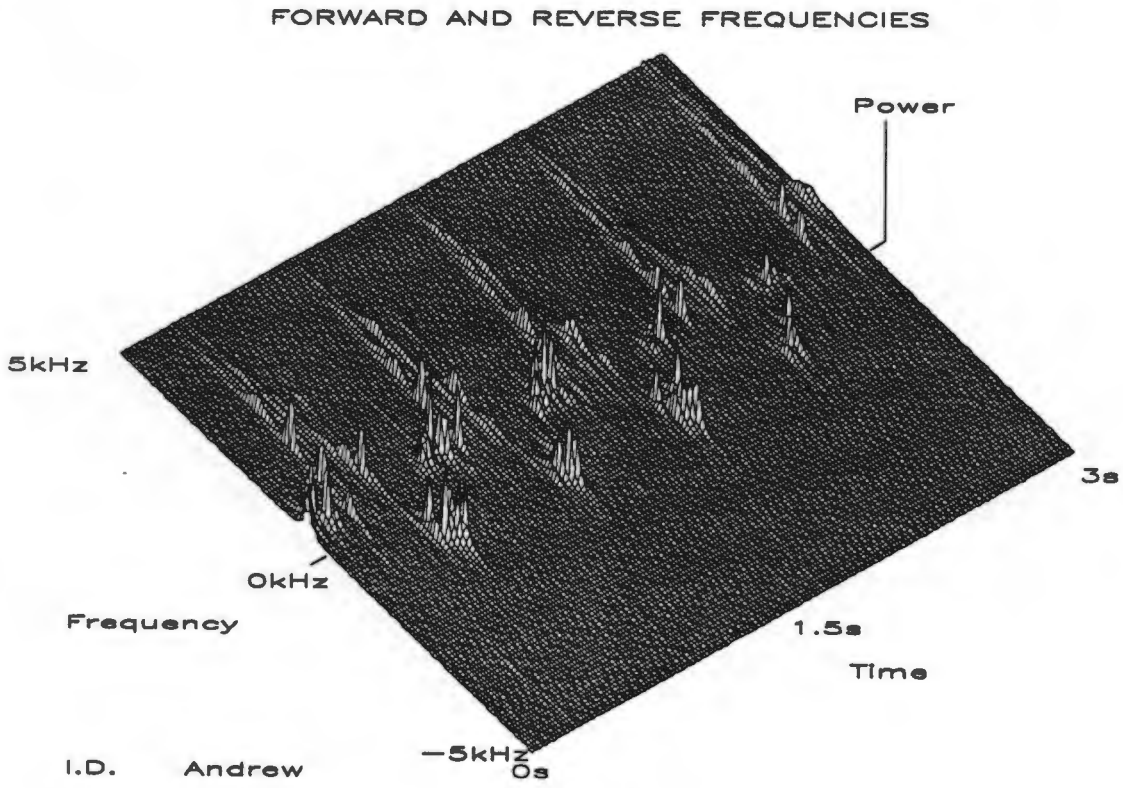


figure 4.2

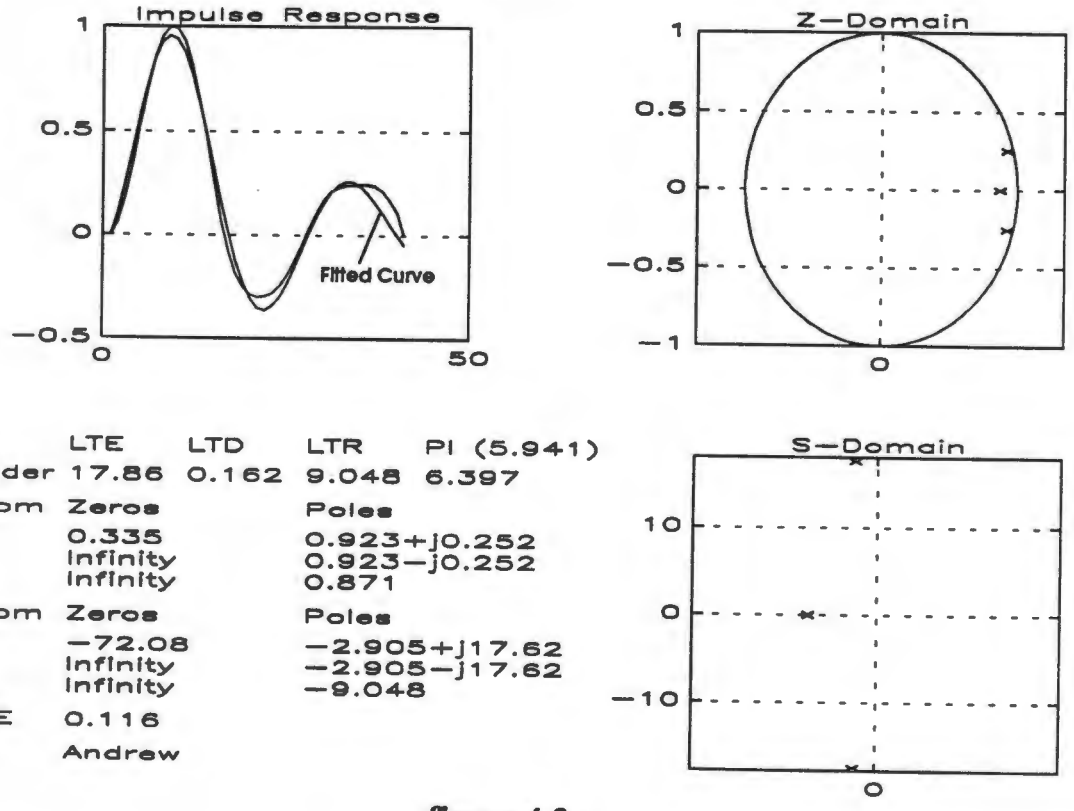


figure 4.3

The graph shown in **figure 4.2** and **4.3** is of a normal healthy 25-year old male. The individual is active in long distance running at a competitive level.

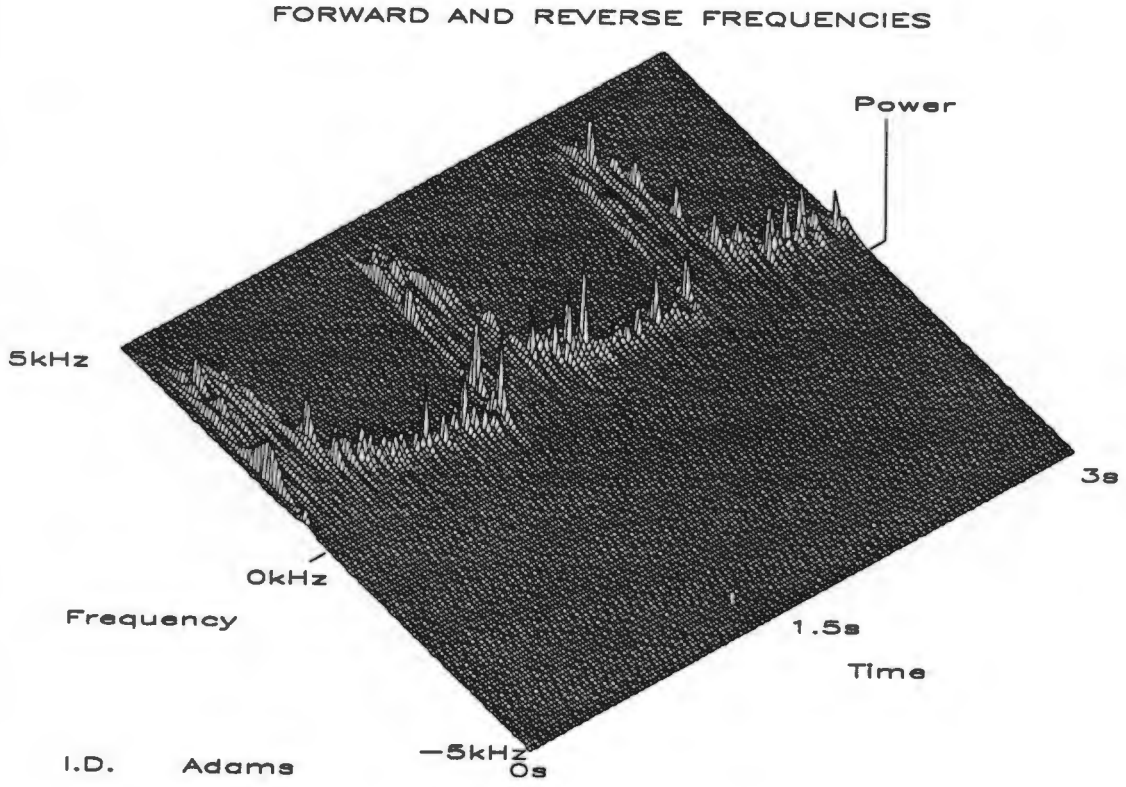
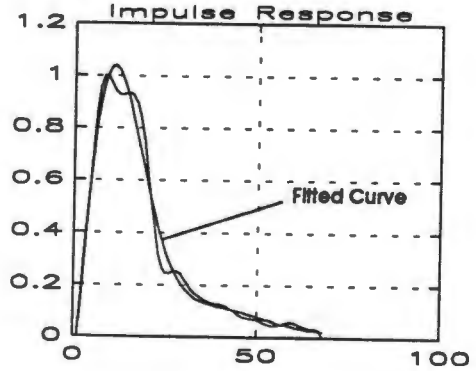
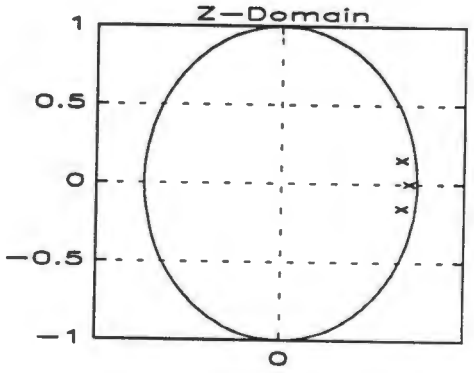


figure 4.4





	LTE	LTD	LTR	PI (3.192)
Nelder	13.4	0.525	3.926	3.314
ZDom	Zeros		Poles	
	0.748		0.942	
	Infinity		0.885+j0.154	
	Infinity		0.885−j0.154	
SDom	Zeros		Poles	
	−19.14		−3.926	
	Infinity		−7.048+j11.4	
	Infinity		−7.048−j11.4	
MSE	0.131			
I.D.	Adams			

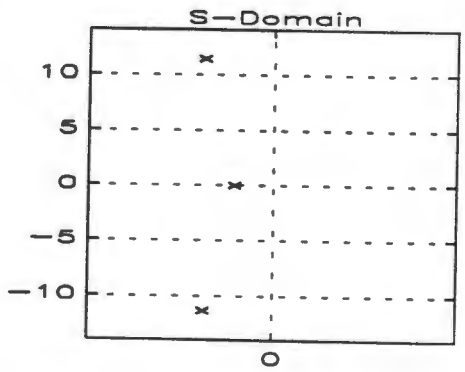


figure 4.5

Figures 4.4 and **4.5** show the results of a 78 year old female (left leg). The peripheral arterial assessment from the department of Radiology reads as follows :

Angiography showed a small infrarenal aneurysm of the abdominal aorta. There was mild disease of all the major vessels of the left leg with a short segment occlusion of the right femoral artery at the adductor hiatus with reconstitution of the popliteals. The trifurcations were patent bilaterally but all crural vessels on both sides were totally occluded shortly below the origin. She was assessed as having unreconstructable disease.

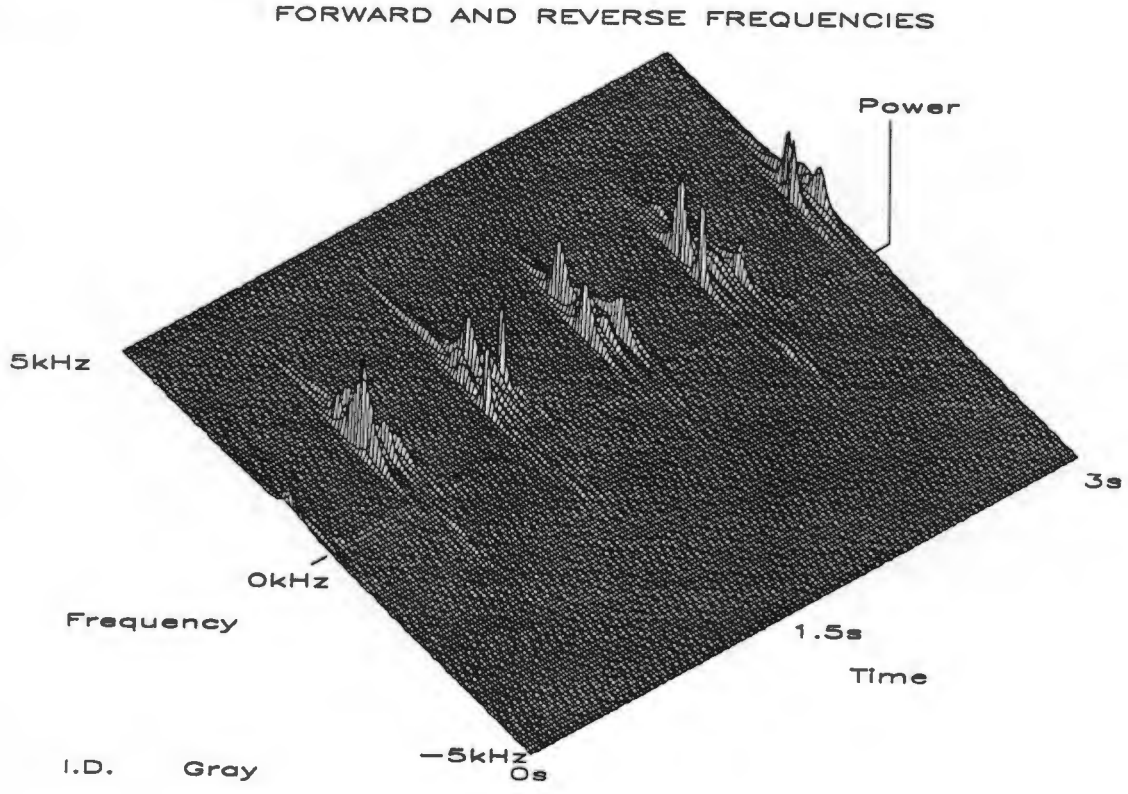
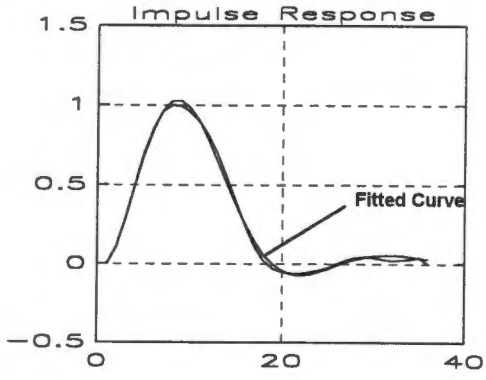
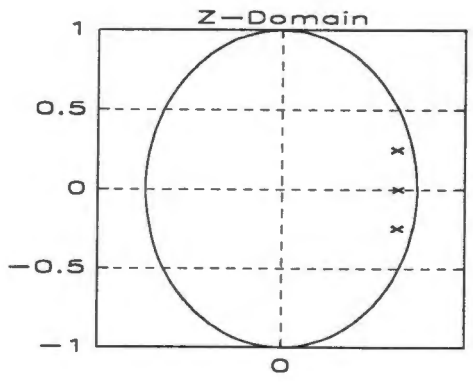


figure 4.6





	LTE	LTD	LTR	PI (3.8)
Nelder	20.24	0.385	9.937	3.914
ZDom	Zeros		Poles	
	−0.146		0.853+j0.248	
	Infinity		0.853−j0.248	
	Infinity		0.86	
SDom	Zeros		Poles	
	−126.8+j207.3		7.808+j18.68	
	Infinity		−7.808−j18.68	
	Infinity		−9.937	
MSE	0.015			
I.D.	Gray			

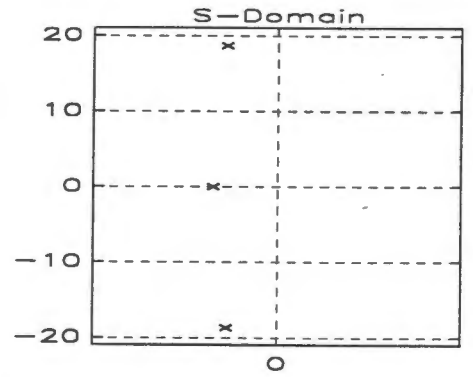


figure 4.7

Figures 4.6 and **4.7** show the results of a 80 year old female (left leg). The peripheral arterial assessment from the department of Radiology reads as follows :

Aorta : Mild infrarenal atheromatous disease. Mild right proximal renal artery stenosis.

Right : Mild common iliac disease. Severe internal iliac disease. Severe profunda/SFA disease with short segment distal occlusion of latter. Heavily diseased popliteal and tight proximal stenosis. Diseased single vessel (peroneal) run-off to ankle.

Left : Mild iliac disease. Severe profunda/SFA disease with thigh grade distal SFA stenosis. Diseased trifurcations. Single vessel (peroneal) run-off to ankle.

CHAPTER FIVE

5.1 DISCUSSION

5.1.1 SYSTEM HARDWARE

The study identified the need for the integration of high speed peripherals. The system hardware that is used has proved to be of an acceptably high quality in its present configuration. With the current trend in operating systems towards graphical user interface based systems the acquisition of a 486-based PC as the host platform should be given consideration.

5.1.2 SYSTEM SOFTWARE

The segmented architecture of the Intel microprocessors used in personal computers today is a major disadvantage if execution speed is of concern. With the application languages and compilers currently only allowing for real mode 16-bit executables all large code and data objects need additional processor overhead to cater for special pointer arithmetic as segment boundaries are crossed. As more sophisticated programming tools become available the application developer will be able to operate the 386-based microprocessors in its protected mode configuration. This will do away with the *segment:offset* pointer arithmetic used in real mode and provide a 32-bit flat address space, with a theoretical limit of 4Giga byte per application. Should the need arise one may recompile the developed code for this mode of operation. This process should prove seamless as the application code in its present state adheres to the ANSI C standard.

5.1.3 SYSTEM MODEL

The analogous electrical circuit model used in this study relates the circuit current (equivalent to mean blood flow) in the terminal branch to the input voltage (equivalent to pressure). The mathematical formula is simplified by assuming the input voltage to be an impulse. The characteristic equation is expressed in the form of poles on the Argand diagram, the poles being the points where the equation becomes infinite. The model has three poles. Two of the poles are a complex conjugate pair, while the third is real. The pole positions express the dynamic behaviour of the system. The two complex poles are used to calculate the damping factor, which relates to the resistance

proximal to the site of measurement and a natural resonant frequency, which is indicative of arterial compliance. The real pole on the other hand relates to the runoff resistance distal to the site of measurement

5.1.4 RESULTS

The use of blood flow waveform analysis to characterise the arterial system has great potential in the non-invasive assessment of arterial disease. This study has highlighted the importance of waveform selection for impulse response analysis. Firstly, the impulse response assumes that the waveform starts at zero. This is not the case when the peripheral resistance is low or when the heart rate is high, as these situations result in a continuous forward flow at the end of diastole. Secondly, fluctuations in the heart rate cause the flow velocity at the end of diastole to differ from that at the beginning of systole. Further work is presently underway to address these problems.

The damping factor has been shown to be a good indicator of lower limb arterial disease, and it is able to separate normals from patients with arterial disease ($p < 0.001$ Mann Whitney U-test). The parameter which is related to runoff resistance is, as expected, highly dependent on peripheral vasodilatation. In the case of distal disease, the decrease in resistance caused by vasodilatation compensates for the increased resistance due to stenoses in all but the very severe cases.

A caution raised by Campbell et al (1984) regarding waveform reproducibility, with specific reference to operator experience, proved most applicable in our study where, although familiar with Doppler CW units the technologist had to be cautioned on the intrusive manner in which measurements were done. With patients that are particularly obese and/or patients with severe occlusions the notion is to apply excessive pressure to obtain waveform readings. As a consequence this acts as a disturbance to the existing flow and inaccurate results are obtained.

5.2 CONCLUSION

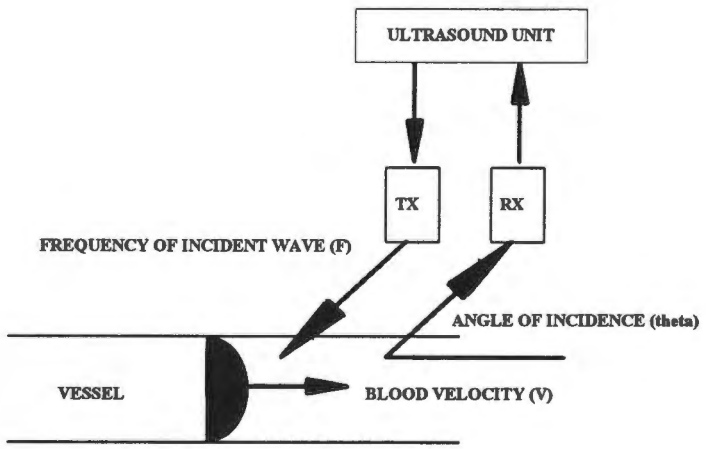
The study is considered to have reached the objectives set. The advantage of having done the initial phase one development has enabled the author to clearly identify potential problems and make recommendations for future research. Proposals which have been made at work group meetings and which may be investigated as post-graduate projects in their own right include research into

the spectral estimation techniques applied to the flow variation, improvement of the Skidmore arterial model and the application of the core unit to different vessel groups. Research in progress involves the gathering of a more comprehensive database and the refinement of the method of waveform selection.

APPENDIX I

DOPPLER PHYSICS

The relative change in frequency due to reflections of acoustic waves off moving structures is well known. A schematic representation of the application to arterial flow is shown below.



Doppler Physics

The change in frequency due to the movement of blood is :

$$\Delta F = \frac{2VF\cos(theta)}{C}$$

where :

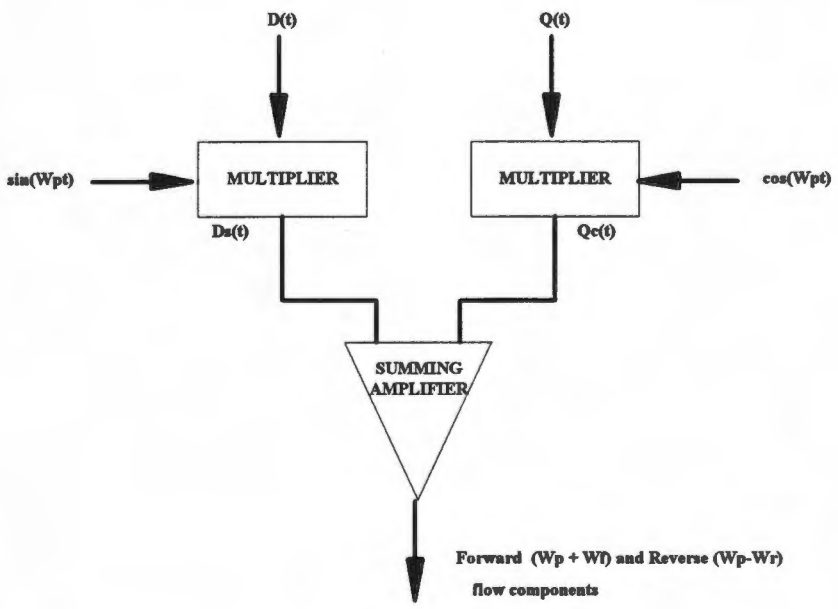
C = velocity of ultrasound in the medium

The transmitted frequency, **F** , is usually selectable at either 5MHz for deeper vessels or 10MHz for superficial vessels. The Doppler frequency shift falls within the audio band for the range of blood cell velocities found in blood vessels. Since blood cells close to the vessel walls move at a lower velocity than those near the vessel centre a range of Doppler shift frequencies is obtained from any given vessel. These are resolved using spectral analysis of the composite Doppler signal. (Note that it is very important to insonate the total cross-section of the blood vessel during examination).

APPENDIX II

FREQUENCY DOMAIN PROCESSING

In order to distinguish between forward and reverse flow components phase quadrature processing has been used. Frequency domain processing shifts the Doppler frequencies to a pilot frequency. The forward and reverse flow components are then identified from their offset from this reference. The following block diagram serves to illustrate.



Frequency domain processing

If we consider the simplistic example of the simultaneous uniform flow of blood in both a forward and reverse direction resulting in frequency shift of W_f and W_r respectively (Atkinson and Woodcock, p69),

$$D(t) = \frac{1}{2}[A\cos\Phi_c + B_f\cos(W_f t + \Phi_f) + B_r\cos(W_r t - \Phi_r)]$$

$$Q(t) = -\frac{1}{2}[A\sin\Phi_c + B_f\sin(W_f t + \Phi_f) + B_r\sin(W_r t - \Phi_r)]$$

where :

B_f, B_r = amplitude of forward and reverse flow respectively

$W_f \Phi_f$ = frequency shift and phase due to forward flowing blood

$W_r \Phi_r$ = frequency shift and phase due to reverse flowing blood

$A\cos\Phi_c$ = DC component due to the demodulation of the carrier

then mathematically the operation alters the direct D(t) and quadrature Q(t) channels as follows :

$$D_s(t) = \frac{1}{2}[A\cos\Phi_c \sin W_p t + B_f \cos(W_f t + \Phi_f)\sin W_p t + B_r \cos(W_r t - \Phi_r)\sin W_p t]$$

$$Q_c(t) = -\frac{1}{2}[A\sin\Phi_c \cos W_p t + B_f \sin(W_f t + \Phi_f)\cos W_p t + B_r \sin(W_r t - \Phi_r)\cos W_p t]$$

$$D_s(t) + Q_c(t) = \frac{1}{2}A(\sin W_p t\cos\Phi_c + \cos W_p t\sin\Phi_c) + \frac{1}{2}B_f[\sin W_p t\cos(W_f t + \Phi_f) + \cos W_p t\sin(W_f t + \Phi_f)] + \frac{1}{2}B_r[\sin W_p t\cos(W_r t - \Phi_r) - \cos W_p t\sin(W_r t - \Phi_r)]$$

$$= A\sin(W_p + \Phi_c) + B_f \sin[(W_f + W_p)t + \Phi_f] + B_r \sin[(W_p - W_r)t + \Phi_r]$$

It thus follows that the forward and reverse frequencies lie on either side of W_p, the pilot frequency. Note that the greater the reverse frequency the **lower** its presence on an absolute frequency scale. The clutter component can be easily eliminated due to its fixed frequency location.

APPENDIX III

DSP SOURCE CODE LISTING (0.0000-0.1372)

```
0.0000 ; ***************************************************
0.0001 ;                ACQUISITION AND ANALYSIS OF
0.0002 ;                LOWER LIMB FLOW WAVEFORMS
0.0003 ; ***************************************************
0.0004 ; ASM for Motorola DSP 56001
0.0005 ;
0.0006 ; Copyright(C) L.SMITH
0.0007 ; Department of Biomedical Engineering
0.0008 ; University of Cape Town
0.0009 ; Observatory
0.0010 ; Cape Town
0.0011 ; New South Africa
0.0012 ; (021) 471250 x 235
0.0013
0.0014 ;***************************************************
0.0015
0.0016 ;SAMPLES QUADRATURE INPUTS FROM BI-DIRECTIONAL
0.0017 ;DOPPLER UNIT. VALUES ARE TRANSFERRED VIA DMA TO
0.0018 ;HOST PC.
0.0019 ;
0.0020 ;
0.0021 ;NOTE THE FOLLOWING :
0.0022 ;1.The ADC is connected to D23..D12. That means that
0.0023 ;  the samples taken are in the most significant 12 bits.
0.0024 ;
0.0025 ;2.With 1 sign bit and 23 significant bits the maximum
0.0026 ;  numerical value received by the PC is 8388608.
0.0027 ;
0.0028 ;3.N Samples are taken per channel at 40kHz sampling rate.
0.0029 ;
0.0030 ;4.DMA transfer at approx 200 kbytes/s.
0.0031 ;  (24 bit transfers are done)
0.0032 ;
0.0033 ;5.R0 is used to keep the loading addresses of
0.0034 ;  channel0 and channel2
0.0035 ;
0.0036 ;6.IRQA is connected to counter 0 and IRQB to counter 1.
0.0037 ;  (Hardware selection on DSP board)
0.0038 ;
0.0039 ;7.Fractional x and / is NOT the same as integer x and / !!
0.0040 ;  Care MUST therefore be taken to ensure that overflow is detected
0.0041 ;  during INT multiplication.
0.0042 ;
0.0043 ;8.HF2 and HF0 is used to synchronize DMA transfer to the PC.
0.0044 ;
0.0045 ;9.The Direct channel is sampled into X and the Quadrature
0.0046 ;  channel into Y memory.
0.0047 ;
0.0048 ;PROBLEMS :
0.0049 ;1.Sampling into anything but the lower 256 positions produces
0.0050 ;  a "clipped" waveform. The fast INT and external memory have
0.0051 ;  been tested. They are not at fault. The problem prevent us
0.0052 ;  from running say sampling and DMA transfers concurrently as
0.0053 ;  the "loading" area then contains a corrupted signal.
0.0054 ;
0.0055 ;2.The FFT algorithm is sensitive to where the data is placed
0.0056 ;  in X and Y memory. In addition the FFT argument appears to
0.0057 ;  dislike a high memory placement.
```

```
0.0058 ;
0.0059 ;************************************************
0.0060
0.0061              opt     fc,mu,s,w,mex  ;ASM COMMANDS
0.0062
0.0063    Port    equ       $8000          ;A/D WRITE PORT
0.0064    ADC     equ       $8001          ;A/D READ PORT
0.0065
0.0066    HRX     equ       $FFEB          ;PERIPHERAL REGISTERS
0.0067    HTX     equ       $FFEB
0.0068    HSR     equ       $FFE9
0.0069    HCR     equ       $FFE8
0.0070    PBC     equ       $FFE0
0.0071    SCR     equ       $FFF0
0.0072    SCCR    equ       $FFF2          ;SCI CLOCK CONTROL REGISTER
0.0073    IPR     equ       $FFFF
0.0074    CT0     equ       $9000          ;8254 TIMER
0.0075    CT1     equ       $9001
0.0076    CT2     equ       $9002
0.0077    CREG    equ       $9003
0.0078    Delay   equ       30             ;MUX
0.0079
0.0080    A_Addr  equ       $600           ;FFT ARGUMENT ADDRESS
0.0081    Ca_Add  equ       $800           ;CARRIER ADDRESS
0.0082    HamA    equ       $1000          ;HAMMING WINDOW
0.0083
0.0084    T0      set       128            ;COUNTER 0 DIVIDE BY (FOR Fs)
0.0085    Fs      set       40e3           ;SAMPLING FREQUENCY (USED IN INITARG)
0.0086    InNum   set       256            ;NUMBER OF SAMPLES RECIEVED
0.0087    Wsp     set       $2000          ;ADDITIONAL WORKSPACE
0.0088    N       set       128            ;SAMPLE NUMBER
0.0089    N_FFT   set       512            ;NUMBER OF FFT DONE
0.0090    Frate   set       $2258          ;FRAMERATE (SEE 11-62) ($2258=15ms)
0.0091                                     ;(10Hz => 40kHz OR 80Hz => 320kHz)
0.0092
0.0093 ;LOADING AREA
0.0094    WkAddr  set  $0                  ;SEE PROBLEM 1 AND RawToPC
0.0095    DMAIn1  set  (WkAddr+InNum-1)    ;TEST SIGNAL RECEIVED FROM PC
0.0096    DMAIn2  set  (Wsp+InNum-1)       ;TEST SIGNAL RECEIVED FROM PC
0.0097
0.0098 ;DISPLAY AREA (NB!! TRANSFER REVERSED)
0.0099    DsAddr1 set  (WkAddr+$3F)
0.0100    DsAddr2 set  (WkAddr+$40+$3F)
0.0101    RawDisp set  (WkAddr+N+$7F)
0.0102
0.0103 ;SHIFT INPUT VALUES DOWN BY ...
0.0104 ;REP #0 NOT ALLOWED (IF 0 DESIRED REMOVE ASR IN IrqA)
0.0105 ;   ANOTHER OPTION IS THE USE OF
0.0106 ;   @CVI(@LOG(N)/@LOG(2)) AS THE MAX VALUE
0.0107 ;   RECIEVED FROM A FFT IS A SUMMATION OF N
0.0108 ;   NUMBERS AND WILL THUS NOT EXCEED 1.
0.0109
0.0110    In_SHFT set  3
0.0111
0.0112              org       p:$0000
0.0113              jmp       BEGIN
0.0114              org       p:$08
0.0115              jsr       IntA         ;ATTACHED TO COUNTER 0 (SAMPLING)
```

```
0.0116              org     p:$0A
0.0117              jsr     IntB            ;ATTACHED TO COUNTER 1 (WINDOW TAKEN)
0.0118              org     p:$001C         ;SCI TIMER INT
0.0119              jsr     FrameR          ;FRAMERATE CONTROL (SCI TIMER 11-61)
0.0120              org     p:$20           ;THE FOLLOWING 2 ARE FAST INT ROUTINES
0.0121              movep   x:HRX,x:(r3)-   ;RECIEVE VALUES (SEE dmarx)
0.0122              org     p:$22
0.0123              movep   x:(r3)-,x:HTX   ;WRITE TO HOST (NOTE DIRECTION OF Rn!!)
0.0124
0.0125              org     p:$0080
0.0126
0.0127
0.0128 ;********** INTERRUPT ROUTINES **********
0.0129 ;ISSUED Fs TIMES PER SECOND (EXECUTION TIME APPROX 12us)
0.0130    IntA      move    #$0A,a1         ;SET UP CH2
0.0131              move    a1,p:Port  ;
0.0132              rep     #Delay          ;ALLOW MUX TO SETTLE
0.0133              nop                     ;(APPROX 2.5us DELAY BETWEEN CHANNELS)
0.0134              move    p:ADC,a
0.0135              rep     #In_SHFT        ;SHIFT DOWN !!(SEE LIMITING)
0.0136              asr     a
0.0137              move    a,y:(r0)        ;READ CH0 INTO Y:MEM
0.0138              move    #$08,a1         ;SET UP CH0
0.0139              move    a1,p:Port       ;
0.0140              rep     #Delay          ;ALLOW MUX TO SETTLE
0.0141              nop
0.0142              move    p:ADC,a
0.0143              rep     #In_SHFT
0.0144              asr     a
0.0145              move    a,x:(r0)+       ;READ CH2 INTO X:MEM
0.0146              rti
0.0147
0.0148 ;ISSUED EVERY (N*T)sec i.e AFTER EVERY TIME WINDOW CAPTURED
0.0149    IntB                              ;IPL RAISED (INT A DISABLED) 8-19
0.0150              movec   ssl,x1          ;DESIRED INT STATE AFTER RTI
0.0151              move    #>$FCFF,a1      ;INT IPL0 NOT ALLOWED
0.0152              move    #>$100,x0
0.0153              and     x1,a
0.0154              or      x0,a
0.0155              move    a1,x1
0.0156              movec   x1,ssl
0.0157              nop
0.0158
0.0159 ;   USE THE FOLLOWING IF FrameR INT NOT USED
0.0160 ;            jclr    #3,x:HSR,_HNR   ;HF0 (HOST NOT READY TO RECIEVE DMA)
0.0161 ;            jsr     RawToPC         ;TRANSFER RAW DATA
0.0162 ;            jsr     WORK            ;PROCEED WITH CALCULATIONS
0.0163 ;            jsr     ToPC            ;TRANSFER DATA
0.0164 ;_HNR        jsr     CNTSetup        ;USE THE FOLLOWING IF FrameR INT
0.0165 ;            jsr     DisCard         ;NOT USED
0.0166 ;            move    #WkAddr,r0
0.0167              rti
0.0168
0.0169    FrameR                            ;IPL RAISED TO IPL3
0.0170              jsr     CNTSetup        ;RESET EXTERNAL TIMERS
0.0171              jsr     DisCard         ;INITIAL VALUES FROM ADC
0.0172              move    #WkAddr,r0
0.0173
```

```
0.0174                andi    #$FC,mr             ;UNMASK INTERRUPTS
0.0175    _busy       jclr    #8,SR,_busy         ;WAIT FOR COMPLETION OF SAMPLING
0.0176                jclr    #3,x:HSR,_HNR       ;HF0 (HOST NOT READY TO RECIEVE DMA)
0.0177                jsr     RawToPC             ;TRANSFER RAW DATA
0.0178                jsr     WORK                ;PROCEED WITH CALCULATIONS
0.0179                jsr     ToPC                ;TRANSFER DATA
0.0180    _HNR
0.0181 ;              move    r5,p:Port           ;TOGGLE PORT TO SEE THAT INT WORKS
0.0182                                            ;NOTE PIN 11 ON IC574
0.0183                rti                         ;SR POPPED (ONLY IPL2 ALLOWED)
0.0184
0.0185 ;************* MACROS ******************
0.0186    Lsp2Wsp MACRO LAddr,WAddr
0.0187
0.0188 ;LOAD DATA INTO WORKING AREA
0.0189 ;
0.0190 ;NB!! NOTE :
0.0191 ;  DATA TRANSFER TO PC AND SAMPLING HAPPENS CONCURRENTLY.
0.0192 ;  BE THEREFORE CAREFUL AS TO WHICH REGISTERS ARE USED.
0.0193
0.0194                move    #LAddr,r2
0.0195                move    #WAddr,r4
0.0196                move    r2,r5
0.0197                move    r4,r1
0.0198
0.0199                do      #N,_xym
0.0200                move    x:(r2)+,x0 y:(r5)+,y0
0.0201                move    x0,x:(r1)+ y0,y:(r4)+
0.0202    _xym
0.0203                ENDM
0.0204
0.0205
0.0206 ;***************************************
0.0207    BitRev  MACRO  I_addr,T_addr,Num
0.0208
0.0209 ;MACRO TO BIT REVERSE FFT OUTPUT
0.0210 ;
0.0211 ;where :
0.0212 ;I_addr = input data pointer
0.0213 ;T_addr = temporary buffer to hold "unscrambled" data
0.0214 ;Num    = Number of values to be RETURNED from
0.0215 ;      temporary buffer
0.0216
0.0217                move    #0,m0
0.0218                move    m0,m4
0.0219                move    #T_addr,r1
0.0220                move    r1,r5
0.0221                move    #N_FFT/2,n0
0.0222                move    n0,n4
0.0223                move    #I_addr,r0
0.0224                move    r0,r4
0.0225
0.0226 ;BIT REVERSE DATA
0.0227                do      #N_FFT,_l1
0.0228                move    x:(r0)+n0,x0        y:(r4)+n4,y0
0.0229                move    x0,x:(r1)+          y0,y:(r5)+
0.0230    _l1
0.0231
```

```
0.0232 ;RESET LINEAR ADDRESS MODE
0.0233                  move    #$FFFF,m0
0.0234                  move    m0,m4
0.0235                  move    #I_addr,r0
0.0236                  move    r0,r4
0.0237                  move    #T_addr,r1
0.0238                  move    r1,r5
0.0239
0.0240 ;PUT DATA VALUES BACK
0.0241                  do      #Num,_l2
0.0242                  move    x:(r1)+,x0              y:(r5)+,y0
0.0243                  move    x0,x:(r0)+ y0,y:(r4)+
0.0244      _l2
0.0245
0.0246                  ENDM
0.0247
0.0248 ;****************************************
0.0249      sincos  macro   points,coef
0.0250
0.0251 ;MACRO TO ESTABLISH ARGUMENTS FOR FFT
0.0252 ;
0.0253 ;where :
0.0254 ;points  = FFT number that will be done.
0.0255 ;coeff   = base address of the arguments
0.0256 ;negative cosine value in X:MEM
0.0257 ;negative sine value in Y:MEM
0.0258
0.0259
0.0260      pi          set     3.141592654
0.0261      freq        set     2.0*pi/@cvf(points)
0.0262
0.0263                  org     x:coef
0.0264      count       set     0
0.0265                  dup     points/2
0.0265                  dc      -@cos(@cvf(count)*freq)
0.0266      count       set     count+1
0.0267                  endm
0.0268
0.0269                  org     y:coef
0.0270      count       set     0
0.0271                  dup     points/2
0.0272                  dc      -@sin(@cvf(count)*freq)
0.0273      count       set     count+1
0.0274                  endm
0.0275                  org         p:
0.0276                  endm
0.0277
0.0278 ;****************************************
0.0279      Carrier  macro   addr
0.0280
0.0281 ;MACRO TO ESTABLISH CARRIER WAVES FOR FREQUENCY
0.0282 ;DOMAIN PROCESSING.
0.0283 ;
0.0284 ;where :
0.0285 ;addr = base address of the arguments
0.0286 ;
0.0287 ;    cos(wt) in X:MEM     (f = 10khz (Fs\4))
0.0288 ;    sin(wt) in Y:MEM
```

```
0.0289
0.0290    freqc     set       5e3
0.0291    pi        set       3.141592654
0.0292    omega     set       (2.0*pi*freqc/Fs)
0.0293
0.0294              org       x:addr
0.0295    count     set       0
0.0296              dup       N
0.0297              dc        0.9999999*@cos(@cvf(count)*omega)
0.0298    count     set       count+1
0.0299              endm
0.0300
0.0301              org       y:addr
0.0302    count     set       0
0.0303              dup       N
0.0304              dc        0.9999999*@sin(@cvf(count)*omega)
0.0305    count     set       count+1
0.0306              endm
0.0307              org       p:
0.0308              endm
0.0309
0.0310 ;***************************************
0.0311    HAMMING macro  points,addr
0.0312
0.0313 ;MACRO TO DETERMINE WINDOW WEIGHTS
0.0314 ;
0.0315 ;w(n) = 0.54-0.46cos(2*pi*n/N)
0.0316 ;where :
0.0317 ;points  = N (Sample number taken)
0.0318 ;addr   = base address of the weights
0.0319
0.0320    pi        set       3.141592654
0.0321    freq      set       2.0*pi/@cvf(points)
0.0322
0.0323              org       x:addr
0.0324    count     set       0
0.0325              dup       points
0.0326              dc        0.9999999*(0.54-0.46*@cos(@cvf(count)*freq))
0.0327    count     set       count+1
0.0328              endm
0.0329
0.0330              org       p:
0.0331              endm
0.0332
0.0333 ;***************************************
0.0334    MulHAM MACRO  Mem,Addr
0.0335
0.0336 ;MULTIPLY TIME WINDOW WITH HAMMING WINDOW
0.0337 ;A window lenght of N is assumed.
0.0338 ;
0.0339 ;where :
0.0340 ;Mem  = X or Y memory
0.0341 ;Addr = address pointer
0.0342 ;Fraction * Integer
0.0343
0.0344              move      #Addr,r0
0.0345              move      #HamA,r1
0.0346
```

```
0.0347                  do      #N,_win
0.0348                  move    \Mem:(r0),y0
0.0349                  move    x:(r1)+,y1
0.0350                  mpyr    y0,y1,b
0.0351                  move    b,\Mem:(r0)+
0.0352      _win
0.0353                  ENDM
0.0354
0.0355 ;*****************************************
0.0356      HAM_XY  MACRO  Addr
0.0357
0.0358 ;MULTIPLY TIME WINDOW WITH HAMMING WINDOW
0.0359 ;BOTH X AND Y MEM ARE MULTIPLIED.
0.0360
0.0361 ;A window lenght of N is assumed.
0.0362 ;where :
0.0363 ;Addr = address pointer
0.0364 ;Fraction * Integer
0.0365
0.0366                  move    #Addr,r0
0.0367                  move    #HamA,r1
0.0368                  move    r0,r4
0.0369                  move    r0,r2
0.0370                  move    r0,r5
0.0371
0.0372                  move    x:(r1)+,x0
0.0373                  move    x:(r0)+,x1  y:(r4)+,y1
0.0374                  do      #N,_win
0.0375                  mpyr    x0,x1,a             x:(r0)+,x1
0.0376                  mpyr    x0,y1,b             x:(r1)+,x0  y:(r4)+,y1
0.0377                  move    a,x:(r2)+   b,y:(r5)+
0.0378      _win
0.0379                  ENDM
0.0380
0.0381 ;*****************************************
0.0382      MulCar  MACRO
0.0383
0.0384 ;MULTIPLY TIME WINDOW WITH CARRIER WAVES
0.0385 ;A WINDOW LENGTH OF N IS ASSUMED.
0.0386 ;
0.0387 ;DIRECT CHANNEL IN X MEM * SINE WAVE
0.0388 ;QUADRATURE CHANNEL IN Y MEM * COS WAVE
0.0389 ;
0.0390                  move    #WkAddr,r0
0.0391                  move    #2,n0
0.0392                  move    #Ca_Add,r4
0.0393                  move    r0,r5
0.0394
0.0395                  move    x:(r0),x0   y:(r4),y0
0.0396                  move    x:(r4)+,x1  y:(r0)+,y1
0.0397                  do      #N,_cwin
0.0398                  mpyr    x0,y0,a             x:(r0),x0   y:(r4),y0
0.0399                  mpyr    y1,x1,b             x:(r4)+,x1  y:(r0)-,y1
0.0400                  move    a,x:(r0)+n0         b,y:(r5)+
0.0401      _cwin
0.0402                  ENDM
0.0403
0.0404 ;*****************************************
```

```
0.0405     Add_DQ MACRO
0.0406
0.0407 ;MULTIPLY TIME WINDOW WITH CARRIER WAVES
0.0408 ;A WINDOW LENGTH OF N IS ASSUMED.
0.0409 ;
0.0410 ;DIRECT CHANNEL IN X MEM * SINE WAVE
0.0411 ;QUADRATURE CHANNEL IN Y MEM * COS WAVE
0.0412 ;Y MEM ZEROED AFTER USE.
0.0413
0.0414                move      #WkAddr,r0
0.0415                move      r0,r4
0.0415                clr       b                    #WkAddr,r1
0.0416
0.0417                move      x:(r0),x0
0.0418                move      y:(r4)+,a
0.0419                do        #N,_awin
0.0420                add       x0,a                 x:(r4),x0   b,y:(r1)+
0.0421                move      a,x:(r0)+   y:(r4)+,a
0.0422
0.0423     _awin
0.0424                ENDM
0.0425
0.0426 ;****************************************
0.0427     Mirr_ZP MACRO Raddr
0.0428
0.0429 ;MIRROR AND ZEROPAD EXTRAPOLATED Rxx
0.0430 ;IT IS ASSUMED THAT N_FFT/2 POINTS EXIST.
0.0431 ;
0.0432 ;where :
0.0433 ;Raddr = Rxx address
0.0434
0.0435                move      #Raddr,r0
0.0436                clr       a                    #Raddr-1,r4
0.0437                move      a,y:(r0)+
0.0438
0.0439                do        #(N_FFT/2-1),_mirror
0.0440                move      x:(r0),b             a,y:(r4)
0.0441                move      b,x:(r4)-            a,y:(r0)+
0.0442     _mirror
0.0443                move      b,x:(r4)             ;COMPENSATE FOR 2N-1 POINT Rxx
0.0444                move      a,y:(r4)             ;THROUGH DUPLICATING LAST POINT
0.0445
0.0446                ENDM
0.0447
0.0448 ;****************************************
0.0449     ForRev     MACRO  Daddr,Qaddr,P_N
0.0450
0.0451 ;MACRO TO CALCULATE FORWARD AND REVERSE FLOW COMPONENTS
0.0452 ;
0.0453 ;where :
0.0454 ;Daddr = Direct channel address
0.0455 ;Qaddr = Quadrature channel address
0.0456 ;P_N  = 1 for doing posotive AND negative frequencies
0.0457 ;     0 for doing only positive.
0.0458 ;
0.0459 ;THE INPUT TO THE MACRO IS THE FREQUENCY SPECTRUM OF BOTH
0.0460 ;CHANNELS IN TERMS OF THEIR REAL AND IMAGINARY COMPONENTS
0.0461 ;AS CALCULATED PREVIOUSLY.
```

```
0.0462 ;
0.0463 ;PHASE DOMAIN PROCESSING IS USED AS FOLLOWS :
0.0464 ;FORWARD FLOW = DIRECT CHANNEL + QUADRATURE CHANNEL PHASE
0.0465 ;                SHIFTED BY PI/2
0.0466 ;REVERSE FLOW = QUADRATURE CHANNEL + DIRECT CHANNEL PHASE
0.0467 ;                SHIFTED BY PI/2
0.0468 ;
0.0469 ;USE IS MADE OF THE FOLLOWING MATHEMATICAL ENTITY :
0.0470 ;POSITIVE FREQUENCIES :
0.0471 ;    (Re + jIm)j = (-Im + jRe)
0.0472 ;                  = Phase advance by PI/2
0.0473 ;
0.0474 ;NEGATIVE FREQUENCIES :
0.0475 ;    (Re + jIm)(-j) = (-Im + jRe)
0.0476 ;                    = Phase delay by PI/2
0.0477 ;
0.0478 ;
0.0479                  move    #Daddr,r0
0.0480                  move    #Qaddr,r4
0.0481
0.0482                  do      #N_FFT/2,_posloop
0.0483
0.0484 ;FORWARD FLOW
0.0485                  move    x:(r0),a    y:(r4),y0
0.0486                  sub     y0,a        x:(r4),b    y:(r0),y0
0.0487                  add     y0,b        a,y0
0.0488                  move    b,x0
0.0489
0.0490 ;REVERSE FLOW
0.0491                  move    x:(r4),a    y:(r0),y1
0.0492                  sub     y1,a        x:(r0),b    y:(r4),y1
0.0493                  add     y1,b        x0,x:(r0)   a,y:(r4)
0.0494                  move    b,x:(r4)+   y0,y:(r0)+
0.0495    _posloop
0.0496
0.0497                  clr  b  #>P_N,x0
0.0498                  cmp     x0,b
0.0499                  jeq     _negloop
0.0500
0.0501                  do      #N_FFT/2,_negloop
0.0502
0.0503 ;FORWARD FLOW
0.0504                  move    x:(r0),a    y:(r4),y0
0.0505                  add     y0,a        x:(r4),x0   y:(r0),b
0.0506                  sub     x0,b        a,y0
0.0507                  move    b,x0
0.0508
0.0509 ;REVERSE FLOW
0.0510                  move    x:(r4),a   y:(r0),y1
0.0511                  add     y1,a        x:(r0),x1   y:(r4),b
0.0512                  sub     x1,b        x0,x:(r0)   a,y:(r4)
0.0513                  move    y0,y:(r0)+ b,x:(r4)+
0.0514    _negloop
0.0515
0.0516                  ENDM
0.0517
0.0518 ;****************************************
0.0519    AveSub  MACRO    Addr
```

```
0.0520
0.0521 ;FIR FILTER. THIS LEAVES Rxx(O)=VARIANCE OF SEQUENCE.
0.0522 ;DETERMINE THE AVERAGE VALUE PER WINDOW AND SUBTRACT
0.0523 ;THAT VALUE FROM EACH SAMPLE FOR BOTH X AND Y MEM.
0.0524 ;
0.0525 ;where :
0.0526 ;Addr = starting address
0.0527
0.0528    SRT       set       @CVI(@LOG(N)/@LOG(2))
0.0529
0.0530              move      #Addr,r4
0.0531              clr       a                    #Addr,r0
0.0532              clr       b
0.0533              move      x:(r0)+,x0 y:(r4)+,y0
0.0534
0.0535              do        #N,_ave
0.0536              add       x0,a                 x:(r0)+,x0
0.0536              add       y0,b                 y:(r4)+,y0
0.0537    _ave
0.0538              rep       #SRT
0.0539              asr       a
0.0540              rep       #SRT
0.0541              asr       b
0.0542
0.0543              move      #Addr,r0
0.0544              move      #Addr,r4
0.0545              move      a,x0
0.0546              move      b,y0
0.0547
0.0548              do        #N,_sub
0.0549              move      x:(r0),a     y:(r4),b
0.0550              sub       x0,a
0.0551              sub       y0,b
0.0552              move      a,x:(r0)+    b,y:(r4)+
0.0553    _sub
0.0554
0.0555              ENDM
0.0556
0.0557 ;****************************************
0.0558    ZeroPad  MACRO  addr,numz
0.0559
0.0560 ;MACRO TO INTERPOLATE FFT FREQUENCIES
0.0561 ;
0.0562 ;where :
0.0563 ;addr = from where do we start zeropadding
0.0564 ;numz = how many zeros do we add
0.0565 ;
0.0566 ;NB! ZERO "do" LOOPS NOT ALLOWED.
0.0567
0.0568              clr       b                    #>numz,x0
0.0569              cmp       x0,b                 #addr,r4
0.0570              jeq       _padloop
0.0571              clr       a                    #addr,r0
0.0572              do        #numz,_padloop
0.0573              move      a,x:(r0)+            b,y:(r4)+
0.0574    _padloop
0.0575
0.0576               ENDM
```

```
0.0577
0.0578 ;****************************************
0.0579      ExtrRxx MACRO C_Addr,R_Addr
0.0580
0.0581 ;MACRO TO EXTRAPOLATE AUTOCORR TO N_FFT/2 POINTS.
0.0582 ;
0.0583 ;       p
0.0584 ;Rxx(n)=-äa(k)*Rxx(n-k)   for |n|>p
0.0585 ;      k=1
0.0586 ;
0.0587 ;p = order of AR filter.
0.0588 ;a = LPC coeff of AR filter.
0.0589 ;LPC IN Y MEM
0.0590 ;Rxx IN X MEM (SEE AUTOCOR)
0.0591 ;
0.0592 ;where :
0.0593 ;C_Addr = LPC coeff address.
0.0594 ;R_Addr = Rxx address.
0.0595
0.0596                  move      #(R_Addr+N_Aut-1),r4
0.0597                  move      #(R_Addr+N_Aut),r1
0.0598                  move      #(C_Addr+1),r0
0.0599                  move      #(N_AR+2),n4
0.0600
0.0601                  do        #N_Pred,_Outl
0.0602                  clr       a
0.0603                  move      x:(r4)-,x0  y:(r0)+,y0
0.0604                  do        #N_AR,_Inl
0.0605                  mac       -x0,y0,a    x:(r4)-,x0          y:(r0)+,y0
0.0606      _Inl
0.0607                  move      a,x:(r1)+                       y:(r4)+n4,y1        ;Y MEM DUMMY READ
0.0608                  move      #(C_Addr+1),r0
0.0609      _Outl
0.0610
0.0611                   ENDM
0.0612
0.0613 ;****************************************
0.0614      sqrt3   MACRO   Prec
0.0615
0.0616 ; INTEGER SQRT
0.0617 ; Full (Prec) bit precision square root routine using
0.0618 ; a successive approximation technique.
0.0619 ;
0.0620 ; y  = double precision (48 bit) positive input number
0.0621 ; b  = 24 bit output root
0.0622 ;
0.0623 ; a  = temporary storage
0.0624 ; x0 = guess
0.0625 ; x1 = bit being tested
0.0626 ; y1:y0 = input number
0.0627
0.0628                  clr       b           #<$40,x0            ;init root and guess
0.0629                  move      x0,x1                           ;init bit to test
0.0630
0.0631                  do        #Prec,_endl                     ;START OF LOOP
0.0632                  mpy       -x0,x0,a                        ;square and negate the guess
0.0633                  asr       a
0.0634                  add       y,a                             ; compare to double precision input
```

```
0.0635              tge     x0,b                ; update root if input >= guess
0.0636              tfr     x1,a                ; get bit to test
0.0637              asr     a                   ; shift to next bit to test
0.0638              add     b,a     a,x1        ; form new guess
0.0639              move    a,x0                ; save new guess
0.0640    _endl                                 ;END OF LOOP
0.0641              ENDM
0.0642
0.0643 ;*************************************
0.0644    Magn  MACRO Addr,Nump
0.0645
0.0646 ;MACRO TO TAKE ABS OF FFT
0.0647 ;
0.0648 ;ABS = SQRT(SQR(REAL)+SQR(IMAG))
0.0649 ;
0.0650 ;where :
0.0651 ;Addr = pointer to buffer space
0.0652 ;Nump = number of points to do
0.0653
0.0654              move    #Addr,r0
0.0655              move    r0,r4
0.0656              move    #Addr,r1
0.0656
0.0657              do      #Nump,_abs
0.0658              move    x:(r0)+,x0          y:(r4)+,y0
0.0659              mpy     x0,x0,a
0.0660              mac     y0,y0,a
0.0661
0.0662              asr     a                   ;INT * (Result in a0)
0.0663              move    a0,y0
0.0664              move    a1,y1
0.0665              sqrt3   13                  ;TAKE INT SQRT
0.0666
0.0667              move    b,x:(r1)+
0.0668    _abs
0.0669              ENDM
0.0670
0.0671 ;*************************************
0.0672    ChrShft MACRO     IAddr,Num
0.0673
0.0674 ;PREPARE FOR CHAR (8 BIT) TRANSFER.
0.0675
0.0676              move    #IAddr,r0
0.0677
0.0678              do      #Num,_chr
0.0679              move    x:(r0),a
0.0680              rep     #16
0.0681              asr     a
0.0682              move    a,x:(r0)+
0.0683    _chr
0.0684              ENDM
0.0685
0.0686 ;*************************************
0.0687    PerG      MACRO Addr,Nump
0.0688
0.0689 ;MACRO TO TAKE ABS^2 OF FFT
0.0690 ;
0.0691 ;ABS = SQR(REAL)+SQR(IMAG)
```

```
0.0692 ;
0.0693 ;where :
0.0694 ;Addr = pointer to buffer space
0.0695 ;Nump = number of points to do
0.0696
0.0697              move    #Addr,r0
0.0698              move    r0,r4
0.0699              move    r0,r1
0.0700
0.0701              move    x:(r0)+,x0          y:(r4)+,y0
0.0702              do      #Nump,_per
0.0703              mpy     x0,x0,a             x:(r0)+,x0
0.0704              mac     y0,y0,a             y:(r4)+,y0
0.0705
0.0706              move    a,x:(r1)+
0.0707    _per
0.0708              ENDM
0.0709
0.0710 ;***************************************
0.0711    AutoCor MACRO  NumC,Ms1,InA,Ms2,OutA
0.0712
0.0713 ;MACRO TO ESTIMATE BIASED AUTOCORRELATION FUNCTION
0.0714 ;
0.0715 ;                  n=N-m-1
0.0716 ;C(m) =       1/N ∑ x(n)*x(n+m)   m=0..NumC
0.0717 ;                  n=0
0.0718 ;
0.0719 ;NumC = number of coeff to be calculated
0.0720 ;MsX  = memory space being used
0.0721 ;InA  = vector address of input array
0.0722 ;OutA = vector address of autocorr coeff
0.0723 ;
0.0724 ;INSERT THE FOLLOWING IF INT MULTIPLICATION
0.0725 ;IS DESIRED.
0.0726 ;           rep     #(SHTCNT+1)                 ;DIVIDE BY N + INT *
0.0727 ;           asr     a
0.0728 ;           move    a0,\Ms2:(r1)+               ;TRANSFER OUT
0.0729
0.0730    SHTCNT  SET     @CVI(@LOG(N)/@LOG(2))
0.0731
0.0732              move    #OutA,r1
0.0733              move    #N,r2
0.0734              move    #0,r4                       ;r4=m
0.0735
0.0736              do      #NumC,_outl
0.0737              clr     a                           #InA,r0
0.0738              move    r4,n0
0.0739
0.0740              do      r2,_inl                     ;r2=N-m-1
0.0741              move    \Ms1:(r0),x0                ;x(n)
0.0742              move    \Ms1:(r0+n0),y0             ;x(n+m)
0.0743              macr    x0,y0,a     (r0)+
0.0744    _inl
0.0745              move    a,\Ms2:(r1)+
0.0746              move    x:(r2)-,x1          y:(r4)+,y1          ;dummy read
0.0747    _outl
0.0748              ENDM
0.0749
```

```
0.0750 ;****************************************
0.0751     InterP  MACRO   INum,addr
0.0752
0.0753 ;TO INTERPOLATE BETWEEN FREQUENCIES. THE OPERATION IS
0.0754 ;EQUIVALENT TO ZEROPADDING WHEN DOING AN FFT.
0.0755 ;
0.0756 ;TWO SUCCESSIVE NUMBERS ARE COMPARED AND A LINEAR INTERPOLATION IS
0.0757 ;MADE. THE FINAL SEGMENT DECAYS TO ZERO.
0.0758 ;where :
0.0759 ;INum = initial number count
0.0760 ;FNum = final number count
0.0761
0.0762     IncN        set         8
0.0763     IncD        set         @CVI(@LOG(IncN)/@LOG(2))
0.0764
0.0765                 clr         a                   #WkAddr,r0
0.0766                 move        #addr,r1
0.0767                 move        a,x:(WkAddr+INum)
0.0768
0.0769                 move        x:(r0)+,x0
0.0770                 move        x:(r0),a
0.0771
0.0772                 do          #INum,_betw
0.0773                 sub         x0,a                x0,x:(r1)+
0.0774                 rep         #IncD
0.0775                 asr         a                   x0,b
0.0776                 move        x:(r0)+,x0 a,y1
0.0777
0.0778                 add         y1,b
0.0778                 do          #IncN-1,_ins
0.0779                 add         y1,b                b,x:(r1)+
0.0780     _ins
0.0781                 move        x:(r0),a
0.0782     _betw
0.0783
0.0784 ;               do          #IncN,_last
0.0785 ;               add         y1,b                b,x:(r1)+
0.0786 ;_last
0.0787                 ENDM
0.0788
0.0789 ;****************************************
0.0790     BartL  MACRO        Num
0.0791
0.0792 ;SMOOTHING OF SPECTRUM THROUGH AVERAGING
0.0793 ;where Num = number of points for each spectrum
0.0794
0.0795                 move        #WkAddr,r0
0.0796                 move        #WkAddr+M,r4
0.0797                 move        #M,n2
0.0798
0.0799                 do          #Num,_add
0.0800                 move        r4,r2
0.0801                 move        x:(r0),a
0.0802                 move        x:(r2)+n2,x0
0.0803
0.0804                 do          #BartN-1,_smooth
0.0805                 add         x0,a                        x:(r2)+n2,x0
0.0806     _smooth
```

```
0.0807              move    a,x:(r0)+           y:(r4)+,y0 ;dummy read
0.0808    _add
0.0809
0.0810              ENDM
0.0811
0.0812 ;*************************************
0.0813    Spect     MACRO
0.0814
0.0815 ;CALULATE SPECTRA OF THE TIME SIGNAL. BARTLETT SMOOTHING
0.0816 ;OF THE PERIODOGRAM FOLLOWS. NOTE THAT THE VARIANCE OF THE FINAL
0.0817 ;SPECTRUM APPROACHES ZERO AS THE SAMPLE NUMBER INCREASES
0.0818 ;
0.0819 ;USE THE FOLLOWING FOR BARTLETT SPECTRAL ESTIMATION
0.0820 ;BartN     set     4           ;BARTLETT BLOCKS
0.0821 ;M         set     128         ;SAMPLES PER BLOCK
0.0822 ;
0.0823 ;          Spect                               ;CALCULATE SPECTRA
0.0824 ;          BartL   N_FFT/4                     ;DO BARTLETT SMOOTHING
0.0825 ;          InterP  N_FFT/4,2*N_FFT             ;INTERPOLATE FREQ POINTS (32->256)
0.0826
0.0827            fftr2c  N_FFT,WkAddr,A_Addr         ;DO COMPLEX FFT
0.0828            BitRev  WkAddr,Wsp,N_FFT/4          ;BITREV OUTPUT OF FFT
0.0829            PerG    WkAddr,N_FFT/4              ;DETERMINE MAGNITUDE
0.0830
0.0831            fftr2c  N_FFT,WkAddr+M,A_Addr
0.0832            BitRev  WkAddr+M,Wsp,N_FFT/4
0.0833            PerG    WkAddr+M,N_FFT/4
0.0834
0.0835            fftr2c  N_FFT,WkAddr+2*M,A_Addr
0.0836            BitRev  WkAddr+2*M,Wsp,N_FFT/4
0.0837            PerG    WkAddr+2*M,N_FFT/4
0.0838
0.0839            fftr2c  N_FFT,WkAddr+3*M,A_Addr
0.0840            BitRev  WkAddr+3*M,Wsp,N_FFT/4
0.0841            PerG    WkAddr+3*M,N_FFT/4
0.0842
0.0843            ENDM
0.0844
0.0845 ;*************************************
0.0846    Leroux    MACRO  nk,Raddr,Kaddr
0.0847
0.0848 ;USE THE FOLLOWING FOR AR SPECTRAL ESTIMATION
0.0849 ;  N_AR     set     15                  ;NEED ONE MORE Rxx THAN ORDER REQUIRED
0.0850 ;  N_Aut    set     64                  ;IN DURBIN ALGORITHM.
0.0851 ;  N_Pred   set     (N_FFT/4-N_Aut)     ;Rxx POINTS PREDICTED.
0.0852 ;  N_Zero   set     N_FFT/4             ;NUMBER OF POINTS TO ZEROPAD
0.0853
0.0854 ;          AutoCor N_Aut,x,WkAddr,x,WkAddr+N
0.0855 ;          Leroux  N_AR,WkAddr+N,WkAddr        ;REFLECTION COEFF
0.0856 ;          Durbin  N_AR,WkAddr+N,WkAddr        ;AR COEFF
0.0857
0.0858 ;          ExtrRxx WkAddr,WkAddr+N             ;EXTRAPOLATE Rxx
0.0859 ;          ZeroPad WkAddr+N+N_FFT/4,N_Zero ;ZEROPAD Rxx
0.0860 ;          Mirr_ZP (WkAddr+N)                  ;MIRROR AND ZERO Y MEM
0.0861 ;          fftr2c  N_FFT,WkAddr,A_Addr         ;DO COMPLEX FFT
0.0862 ;          BitRev  WkAddr,Wsp,N_FFT/4          ;BITREV OUTPUT OF FFT
0.0863 ;          Magn    WkAddr,N_FFT/4
0.0864
```

```
0.0865 ; Leroux-Gueguen Solution for PARCOR (LPC) Coefficients
0.0866 ;
0.0867 ;    Implements the Leroux-Gueguen algorithm.
0.0868 ;
0.0869 ;X MEMORY VALUES
0.0870    fk        set       Kaddr              ;LPC reflection coefficients
0.0871    r         set       Raddr
0.0872
0.0873              org       y:Kaddr
0.0874    bim1      ds        nk+1
0.0875    bi        ds        nk
0.0876 ;
0.0877              org       p:
0.0878
0.0879 ;    Start of LPC analysis
0.0880 ;    The following registers correspond to the FORTRAN variables:
0.0881 ;
0.0882 ;    r0 - r[i]  = irptr     r4 - bim1[j] = jbm1
0.0883 ;    r1 - fk[i] = ifk       r5 - bim1[i] = ibim1
0.0884 ;    r2 - fk[j] = jfk       r6 - bi[i]   = ibi
0.0885 ;    r3 - bi[j] = jbi       r7 - loop counter = im1
0.0886 ;
0.0887 ;    Approximate analysis times:
0.0888 ;
0.0889 ;    8th  order: 46.2  uS
0.0890 ;    10th order: 61.6  uS
0.0891 ;    16th order: 117.2 uS
0.0892 ;
0.0893 ;    Initialization
0.0894 ;
0.0895        move   #r,r0                        ;r(0) points to r[0]
0.0896        move   #fk,r1                       ;r(1) points to fk[1]
0.0897        move   #bim1,r5                     ;r(5) points to bim1[1]
0.0898        move   #bi+1,r6                     ;r(6) points to bi[2]
0.0899 ;
0.0899 ;    Begin Leroux - Gueguen Algorithm
0.0900 ;
0.0901        move          x:(r0)+,x0            ;get r[0]
0.0902        move          x:(r0)+,a             ;get r[1], r(0) points to r[2]
0.0903        abs           a          a,b        ;get abs(r[1]), copy r[1] to b
0.0904        eor           x0,b       b,y0       ;N = sign bit, copy r[1] to y0
0.0905        and           #$fe,ccr              ;clear quotient sign bit
0.0906        rep           #24                   ;set up for 24-bit quotient
0.0907        div           x0,a                  ;get abs(r[1])/r[0]
0.0908        jmi           L1                    ;check sign bit N
0.0909        neg           a                     ;negate quotient if needed
0.0910    L1  move          a0,a                  ;move -r[1]/r[0] to a
0.0911        move          a,x:(r1)+  a,y1             ;copy -r[1]/r[0] to fk[1] and y1
0.0912        tfr           x0,b       y0,y:(r5)+       ;bim1[1] = r[1], copy r[0] to b
0.0913        macr          y1,y0,b    #1,r7            ;r[0] + fk[1] * r[1]; loop count = 1
0.0914        move          b,y:(r5)                    ;bim1[2] = r[0] + (fk[1] * r[1])
0.0915 ;
0.0916 ;    outer do loop
0.0917 ;
0.0918        do            #nk-1,L4
0.0919        move          #fk,r2                ;r(2) points to fk[1]
0.0920        move          #bi,r3                ;r(3) points to bi[1]
0.0921        move          #bim1,r4              ;r(4) points to bim1[1]
```

```
0.0922          move            x:(r0)+,b   y:(r5),y1            ;yi = r[i], get bim1[i]
0.0923          move            b,x1        b,y:(r3)            ;copy yi to x1, bi[1] = yi
0.0924 ;
0.0925 ;    inner do loop
0.0926 ;
0.0927          do              r7,L2                           ;begin inner do loop
0.0928          move            x:(r2)+,x0          y:(r4),a    ;get fk[j] and bim1[j]
0.0929          macr            x1,x0,a             a,x1 y:(r3)+,y0 ;bim1[j]+(fk[j]*yi)
0.0930          macr            x1,x0,b             a,y:(r3)    ;save bi[j+1], yi=yi+(fk[j]*bim1[j])
0.0931          move            b,x1                y0,y:(r4)+  ;copy yi to x1, save bim1[j]
0.0932 ;
0.0933 ;    end of inner do loop (note: b = x1 = yi)
0.0934 ;
0.0935    L2    abs             b           b,a     ;get abs(yi), copy yi to a
0.0936          eor             y1,a        (r7)+   ;N = sign bit, inc. outer loop counter
0.0937          and             #$fe,ccr            ;clear quotient sign bit
0.0938          rep             #24                 ;set up for 24-bit quotient
0.0939          div             y1,b                ;get abs(yi)/bim1[i]
0.0940          jmi             L3                  ;check sign bit N
0.0941          neg             b                   ;negate quotient if needed
0.0942    L3    move            b0,x0                               ;move -yi/bim1[i] to x0
0.0943          tfr             y1,b        x0,x:(r1)+ y:(r6)+,y0   ;get bim1[i],save fk[i],get bi[i]
0.0944          macr            x1,x0,b     y0,y:(r5)+              ;bim1[i]+(fk[i]*yi), bim1[i] = bi[i]
0.0945          move            b,y:(r5)                            ;bim1[i+1] = bim1[i]*(fk[i]*yi)
0.0946 ;
0.0947 ;    end of outer do loop
0.0948 ;    end of analysis
0.0949    L4
0.0950
0.0951              ENDM
0.0952
0.0953 ;*************************************
0.0954    Durbin  MACRO   nk,Raddr,Kaddr
0.0955
0.0956 ;MACRO TO ESTIMATE AR COEFFICIENTS GIVEN REFLECTION COEFF
0.0957 ;
0.0958 ;nk          = number of AR reflection coeff
0.0959 ;Raddr       = autocorrelation address
0.0960 ;Kaddr       = AR coefficient address
0.0961
0.0962    acoef   set     Kaddr               ;LPC filter coeff (nk+1 values)
0.0963    refl    set     Kaddr               ;Reflection coeff (nk values)
0.0964    anew    set     (Kaddr+nk+1)        ;updated LPC filter coeff (nk values)
0.0965    error   set     (anew+nk);error
0.0966    alpha   set     (error+1)  ;alpha
0.0967
0.0968            move    #refl,r1    ;r1 points to k[0]
0.0969            move    #Raddr,r0   ;r0 points to r[0]
0.0970            move    #anew,r3    ;r3 points to anew[0]
0.0971            move    #acoef,r4   ;r4 points to a[0]
0.0972            move    #error,r6   ;r6 points to error
0.0973
0.0974            move    #$7FFFFF,a
0.0975            move    a,y:(r4)+                   ;define a[0]=1.0
0.0976            move    a,y:(r3)                    ;anew[0]=1.0
0.0977
0.0978            move    #2,r2                       ;r2=i (SEE BELOW)
0.0979            move    x:(r1)+,a                   ;k[0]
```

```
0.0980              move    x:(r0)+,x0 a,y:(r4)+   ;r[0] in x0, a[1]=k[0]
0.0981
0.0982              clr     b                      a,y0                ;k[0] in y0
0.0983              mpy     y0,y0,a                x0,b                ;r[0] in b
0.0984              move    a,y0                                       ;k[0]^2 in y0
0.0985              mac     -y0,x0,b               #2,n2               ;alpha in b
0.0986              move    b,x1                                       ;alpha in x1
0.0987
0.0988              move    #refl,n1                                   ;n1 points to k[0]
0.0989              move    #Raddr,n0                                  ;n0 points to r[0]
0.0990              move    #acoef,n5
0.0991
0.0992              do      #nk-1,_L1                                  ;OUTER LOOP
0.0993              move    r2,r0
0.0994              clr     a                      #acoef,r4   ;r4 points to a[0]
0.0995              move    (r0)+n0
0.0996
0.0997              do      r2,_L2                                     ;DO i TIMES
0.0998              move    x:(r0)-,x0             y:(r4)+,y0
0.0999              mac     x0,y0,a                                    ;error in a
0.1000    _L2
0.1001              move    (r2)-                                      ;r2=i-1
0.1002              move    r2,r1
0.1003              move    a,y:(r6)+                                  ;error out
0.1004
0.1005              move    x:(r1+n1),y0                               ;k[i-1]
0.1006              mpy     y0,y0,a                x1,b                ;y0=k[i-1]
0.1007              move    a,y1
0.1008              mac     -y1,x1,b               y0,y:(r4)           ;a[i-1]=k[i-1]
0.1009              move    b,x1                   b,y:(r6)-           ;alpha in x1
0.1010
0.1011              move    #(anew+1),r3
0.1012              move    #(acoef+1),r4                              ;r4=a[1]
0.1013              move    r2,r5                                      ;r5=i-1
0.1014
0.1015              do      r2,_L3                                     ;DO i-1 TIMES
0.1016              move    y:(r4)+,b
0.1017              move    y:(r5+n5),x0                               ;a[i-j+1]
0.1018              mac     x0,y0,b     (r5)-
0.1019              move    b,y:(r3)+
0.1020
0.1020    _L3
0.1021              do      r2,_L4                                     ;DO i-1 TIMES
0.1022              move    y:-(r3),b
0.1023              move    b,y:-(r4)
0.1024    _L4
0.1025              move    (r2)+n2                                    ;INC i
0.1026    _L1
0.1027
0.1028              ENDM
0.1029
0.1030 ;***********************************
0.1031    fftr2c  MACRO   points,data,coef
0.1032
0.1033 ;NB! IF COEF IS TO LARGE A OVERFLOW OCCURS WHEN THE OFFSET
0.1034 ;n6 IS ADDED !!!
0.1035 ;
0.1036 ;MACRO TO AID IN CALCULATION OF FREQUENCY CONTENT
```

```
0.1037 ;OF A SIGNAL. THE SAME DATA BUFFER IS USED TO HOLD
0.1038 ;INPUT AND OUTPUT DATA
0.1039 ;
0.1040 ; Radix 2 Decimation in Time In-Place Fast Fourier Transform Routine
0.1041 ;   Complex input and output data
0.1042 ;      Real data in X memory
0.1043 ;      Imaginary data in Y memory
0.1044 ;   Normally ordered input data
0.1045 ;   Bit reversed output data
0.1046 ;      Coefficient lookup table
0.1047 ;      -Cosine values in X memory
0.1048 ;      -Sine values in Y memory
0.1049 ;
0.1050 ; Macro Call - fftr2c   points,data,coef
0.1051 ;      points    number of points (16-32768, power of 2)
0.1052 ;      data      start of data buffer
0.1053 ;      coef      start of sine/cosine table
0.1054 ;
0.1055 ; Alters Data ALU Registers
0.1056 ;      x1    x0    y1    y0
0.1057 ;      a2    a1    a0    a
0.1058 ;      b2    b1    b0    b
0.1059 ;
0.1060 ; Alters Address Registers
0.1061 ;      r0    n0    m0
0.1062 ;      r1    n1    m1
0.1063 ;            n2
0.1064 ;      r4    n4    m4
0.1065 ;      r5    n5    m5
0.1066 ;      r6    n6    m6
0.1067 ;
0.1068 ; Alters Program Control Registers
0.1069 ;      pc    sr
0.1070 ;
0.1071 ; Uses 6 locations on System Stack
0.1072 ;
0.1073          move   #data,r0              ;initialize input pointer
0.1074          move   #points/4,n0          ;initialize butterflies per group
0.1075          move   n0,n4                 ;initialize pointer offsets
0.1076          move   n0,n6                 ;initialize coefficient offset
0.1077          move   #points-1,m0          ;initialize address modifiers
0.1078          move   m0,m1                 ;for modulo(points) addressing
0.1079          move   m0,m4
0.1080          move   m0,m5
0.1081 ;
0.1082 ; Do first and second Radix 2 FFT passes
0.1083                 move     x:(r0)+n0,x0
0.1084                 tfr      x0,a       x:(r0)+n0,y1
0.1085
0.1086                 do       n0,_twopass
0.1087                 tfr      y1,b       x:(r0)+n0,y0
0.1088                 add      y0,a       x:(r0),x1                ;ar+cr
0.1089                 add      x1,b       r0,r4                    ;br+dr
0.1090                 add      a,b        (r0)+n0                  ;ar'=(ar+cr)+(br+dr)
0.1091                 subl     b,a        b,x:(r0)+n0              ;br'=(ar+cr)-(br+dr)
0.1092                 tfr      x0,a       a,x0        y:(r0),b
0.1093                 sub      y0,a       y:(r4)+n4,y0             ;ar-cr
0.1094                 sub      y0,b       x0,x:(r0)                ;bi-di
```

```
0.1095              add     a,b     y:(r0)+n0,x0    ;cr'=(ar-cr)+(bi-di)
0.1096              subl    b,a     b,x:(r0)        ;dr'=(ar-cr)-(bi-di)
0.1097              tfr     x0,a    a,x0            y:(r4),b
0.1098              add     y0,a    y:(r0)+n0,y0    ;bi+di
0.1099              add     y0,b    x0,x:(r0)+n0    ;ai+ci
0.1100              add     b,a     y:(r0)+,x0      ;ai'=(ai+ci)+(bi+di)
0.1101              subl    a,b     a,y:(r4)+n4     ;bi'=(ai+ci)-(bi+di)
0.1102              tfr     x0,a    b,y:(r4)+n4
0.1103              sub     y0,a    x1,b            ;ai-ci
0.1104              sub     y1,b    x:(r0)+n0,x0    ;dr-br
0.1105              add     a,b     x:(r0)+n0,y1    ;ci'=(ai-ci)+(dr-br)
0.1106              subl    b,a     b,y:(r4)+n4     ;di'=(ai-ci)-(dr-br)
0.1107              tfr     x0,a    a,y:(r4)+
0.1108  _twopass
0.1109 ;
0.1110 ; Do all FFT passes but first, second and last pass
0.1111              move    #points/8,n0            ;initialize butterflies per group
0.1112              move    #4,n2                   ;initialize groups per pass
0.1113              move    #0,m6                   ;initialize coefficient address modifier
0.1114                                              ;for reverse carry (bit reversed) addressing
0.1115
0.1116              do      #@cvi(@log(points)/@log(2)-2.5),_end_pass
0.1117              move    #data,r0                ;initialize A input pointer
0.1118              move    r0,r4                   ;initialize A output pointer
0.1119              lua     (r0)+n0,r1              ;initialize B input pointer
0.1120              move    #coef,r6                ;initialize C input pointer
0.1121              lua     (r1)-,r5                ;initialize B output pointer
0.1122              move    n0,n1                   ;initialize pointer offsets
0.1123              move    n0,n4
0.1124              move    n0,n5
0.1125
0.1126              do      n2,_end_grp
0.1127              move    x:(r1),x1       y:(r6),y0       ;lookup -sine value
0.1128              move    x:(r5),a        y:(r0),b
0.1129              move    x:(r6)+n6,x0                    ;lookup -cosine value
0.1130
0.1131              do      n0,_end_bfy                     ;Radix 2 DIT butterfly kernel with constant
0.1132                                                      ;twiddle factor
0.1133              mac     x1,y0,b         y:(r1)+,y1
0.1134              macr    -x0,y1,b        a,x:(r5)+       y:(r0),a
0.1135              subl    b,a             x:(r0),b        b,y:(r4)
0.1136              mac     -x1,x0,b        x:(r0)+,a       a,y:(r5)
0.1137              macr    -y1,y0,b        x:(r1),x1
0.1138              subl    b,a             b,x:(r4)+       y:(r0),b
0.1139  _end_bfy
0.1140              move    a,x:(r5)+n5     y:(r1)+n1,y1    ;dummy load of x1 and y1
0.1141              move    x:(r0)+n0,x1    y:(r4)+n4,y1
0.1141  _end_grp
0.1142              move    n0,b1
0.1143              lsr     b               n2,a1           ;divide butterflies per group by two
0.1144              lsl     a               b1,n0           ;multiply groups per pass by two
0.1145              move    a1,n2
0.1146  _end_pass
0.1147
0.1148 ; Do last FFT pass
0.1149              move    n1,n0                   ;correct pointer offset for last pass
0.1150              move    #data,r0                ;initialize A input pointer
0.1151              move    r0,r4                   ;initialize A output pointer
```

```
0.1152                  lua     (r0)+,r1                        ;initialize B input pointer
0.1153                  move    #coef,r6                        ;initialize C input pointer
0.1154                  lua     (r1)-n1,r5                      ;initialize B output pointer
0.1155                  move    x:(r1),x1       y:(r6),y0
0.1156                  move    x:(r5),a        y:(r0),b
0.1157
0.1158                  do      n2,_lastpass                    ;Radix 2 DIT butterfly kernel with one
0.1159                                                          ;butterfly per group
0.1160                  mac     x1,y0,b         x:(r6)+n6,x0    y:(r1)+n1,y1
0.1161                  macr    -x0,y1,b        a,x:(r5)+n5     y:(r0),a
0.1162                  subl    b,a             x:(r0),b        b,y:(r4)
0.1163                  mac     -x1,x0,b        x:(r0)+n0,a     a,y:(r5)
0.1164                  macr    -y1,y0,b        x:(r1),x1       y:(r6),y0
0.1165                  subl    b,a             b,x:(r4)+n4     y:(r0),b
0.1166  _lastpass
0.1167                  move    a,x:(r5)+n5
0.1168
0.1169                  move    #$FFFF,m0       ;DEFAULT LINEAR ARITHMETIC SELECTED
0.1170                  move    m0,m1
0.1171                  move    m0,m4
0.1172                  move    m0,m5
0.1173                  move    m0,m6
0.1174
0.1175                  ENDM
0.1176
0.1177 ;********* PROGRAM START **********
0.1178  BEGIN   ori     #$3,mr          ;ONLY INT WITH IPL3 ALLOWED
0.1179          bclr    #2,omr          ;DISABLE ROM TABLES
0.1180          movep   #1,x:PBC        ;CONFIGURE PORT B AS HOST PORT
0.1181                                  ;(AS OPPOSED TO GENERAL-PURPOSE I/O)
0.1182          bset    #2,x:HCR        ;ENABLE VECTORED HOST INTERRUPTS
0.1183          bset    #1,x:HCR        ;ENABLE XMT INT
0.1184          bset    #0,x:HCR        ;ENABLE RCV INT
0.1185
0.1186          movep   #$C809,x:IPR    ;GIVE IRQA IPL0, IRQB IPL0, HI IPL1
0.1187                                  ;AND SCI TIMER IPL2(SEE ALSO 8.17)
0.1188          sincos  N_FFT,A_Addr    ;INITLIAZE THE VARIOUS LOOKUP TABLES
0.1189          Carrier Ca_Add
0.1190          HAMMING N,HamA
0.1191
0.1192 ;...AND ACTION...
0.1193          movep   #$2000,x:SCR    ;ENABLE TIMER INT
0.1194          movep   #Frate,x:SCCR   ;INT RATE = Fosc[20.48MHz]/(512*Frate)
0.1195          andi    #$FC,mr
0.1196          ori     #$2,mr          ;ONLY INT WITH IPL2,3 ALLOWED
0.1197
0.1198 ;  USE THE FOLLOWING IF FrameR INT NOT USED
0.1199 ;       jsr     CNTSetup        ;SETUP OF IRQA AND IRQB
0.1200 ;       jsr     DisCard         ;START UP A/D CONVERTER
0.1201 ;       move    #WkAddr,r0      ;DATA ADDRESS
0.1202 ;       andi    #$FC,mr         ;UNMASK INTERRUPTS
0.1203
0.1204 ;       andi    #$FC,mr
0.1205 ;       ori     #$1,mr          ;INT WITH IPL1,2,3 ALLOWED
0.1206 ;       jsr     FromPC          ;USE WITH To_DSP ON PC
0.1207
0.1208
0.1209 ;## MAIN PROGRAM LOOP ##
```

```
0.1210    MAIN_LOOP
0.1211 ;            jsr       ToPC
0.1212              jmp       MAIN_LOOP
0.1213
0.1214 ;########## SUBROUTINES ##########
0.1215 ;SETUP OF PROGRAMABLE INTERVAL TIMER
0.1216    CNTSetup
0.1217
0.1218 ;SET WAIT STATES TO SLOW SPEED FOR 8254-2 DEVICE
0.1219              movep     #$2222,x:$FFFE    ;(x,y,p,io)
0.1220
0.1221 ;TIMING SETUP IRQA
0.1222              move      #%110100,r0       ;COUNTER 0 IN MODE 2
0.1223              move      #CREG,r1          ;CONTROL REG ADDR
0.1224              jsr       wrtimer
0.1225              clr       a                 #>$FF,x0
0.1226              move      #>T0,a1           ;INT PERIOD:5.12MHz/128=40kHz
0.1227              and       x0,a              #CT0,r1
0.1228              move      a1,r0
0.1229              jsr       wrtimer           ;LSB TO COUNTER
0.1230              clr       a                 #>$FF00,x0
0.1231              move      #>T0,a1
0.1232              and       x0,a
0.1233              rep       #8
0.1234              asr       a
0.1235              move      a1,r0
0.1236              jsr       wrtimer           ;MSB TO COUNTER
0.1237
0.1238 ;TIMING SETUP IRQB
0.1239              move      #%1110100,r0      ;COUNTER 1 IN MODE 2
0.1240              move      #CREG,r1          ;CONTROL REG ADDR
0.1241              jsr       wrtimer
0.1242              clr       a                 #>$FF,x0
0.1243              move      #>N,a1            ;INT PERIOD:40kHz/N
0.1244              and       x0,a              #CT1,r1
0.1245              move      a1,r0
0.1246              jsr       wrtimer           ;LSB TO COUNTER
0.1247              clr       a                 #>$FF00,x0
0.1248              move      #>N,a1
0.1249              and       x0,a
0.1250              rep       #8
0.1251              asr       a
0.1252              move      a1,r0
0.1253              jsr       wrtimer           ;MSB TO COUNTER
0.1254
0.1255 ;SET WAIT STATES FOR ADDED SPEED
0.1256              movep     #$0010,x:$FFFE    ;(x,y,p,io)
0.1257              rts
0.1258
0.1259 ;################################
0.1260 ;WRITING TO 8254 TIMER
0.1260    wrtimer   move      r0,p:(r1)
0.1261              rep   #4
0.1262              nop
0.1263              rts
0.1264
0.1265 ;################################
0.1266 ;DISCARD INITIAL VALUES FROM THE ADC
```

```
0.1267   DisCard  move    #$08,a1
0.1268            move    a1,p:Port          ;SET UP CH0
0.1269            rep     #Delay             ;ALLOW MUX TO SETTLE
0.1270            nop                        ;(APPROX 2.5us DELAY BETWEEN CHANNELS)
0.1271            move    p:ADC,a1           ;READ CH0 AND DISCARD VALUE
0.1272            rts
0.1273
0.1274 ;###################################
0.1275   RawToPC  jclr    #3,x:HSR,_NReI     ;HF0 (HOST NOT READY)
0.1276
0.1277 ;TRANSFER OF RAW DATA. (INTEGERS ARE TRANSFERED)
0.1278 ;FIRST X AND THEN Y MEM IS SHIFTED FROM WkAddr TO Wk_Addr+N
0.1279 ;IN X MEM AND THEN SHIFTED DOWN ACCORDING TO THE VALUE OF
0.1280 ;In_SHFT. THE DESIRED POSITION IS THE LOWEST
0.1281
0.1282            move    #WkAddr,r2
0.1283            move    #WkAddr+N,r4
0.1284
0.1285            do      #N,_xm
0.1286            move    x:(r2)+,a
0.1287            rep     #12-In_SHFT
0.1288            asr     a
0.1289            move    a,x:(r4)+
0.1290   _xm
0.1291            move    #RawDisp,r3
0.1292            jsr     dmatx                      ;TRANSFER DATA
0.1293
0.1294            move    #WkAddr,r2
0.1295            move    #WkAddr+N,r4
0.1296
0.1297            do      #N,_ym
0.1298            move    y:(r2)+,a
0.1299            rep     #12-In_SHFT
0.1300            asr     a
0.1301            move    a,x:(r4)+
0.1302   _ym
0.1303            move    #RawDisp,r3
0.1304            jsr     dmatx                      ;TRANSFER DATA
0.1305
0.1306   _NReI
0.1307            rts
0.1308
0.1309 ;###################################
0.1310   ToPC     jclr    #3,x:HSR,_NRe              ;HF0 (HOST NOT READY)
0.1311            move    #DsAddr1,r3
0.1312            jsr     dmatx                      ;TRANSFER DATA
0.1313
0.1314            move    #DsAddr2,r3
0.1315            jsr     dmatx                      ;TRANSFER DATA
0.1316   _NRe
0.1317            rts
0.1318
0.1319 ;###################################
0.1320 ;DMA TRANSFER TO PC
0.1321   dmatx    bset    #3,x:HCR                   ;HF2 (HOST MAY REQUEST DMA)
0.1322   _bf      jclr    #7,x:HSR,_tf               ;WAIT FOR DMA PROCESS TO BE INITIATED
0.1323            jmp     _bg                        ;DMA PROCESS INITIATED
0.1324   _tf      jset    #3,x:HSR,_bf               ;HOST DO WANT DMA
```

```
0.1325    _bg       jset      #7,x:HSR,_bg                ;WAIT FOR COMPLETION OF DMA
0.1326              bclr      #3,x:HCR                    ;HF2 (NO DMA TRANSFER)
0.1327              rts
0.1328
0.1329 ;################################
0.1330    FromPC    move      #DMAIn1,r3
0.1331              jsr       dmarx
0.1332              move      #DMAIn2,r3
0.1333              jsr       dmarx
0.1334
0.1335              move      #Wsp,r0
0.1336              move      #WkAddr,r1
0.1337              do        #InNum,_mv
0.1338              move      x:(r0)+,a
0.1339              move      a,y:(r1)+
0.1340    _mv
0.1341
0.1342              jsr       WORK
0.1343              rts
0.1344
0.1345 ;################################
0.1346 ;DMA TRANSFER FROM PC
0.1347    dmarx
0.1348    _bs       jclr      #7,x:HSR,_bs                ;WAIT FOR DMA PROCESS TO BE INITIATED
0.1349    _bt       jset      #7,x:HSR,_bt                ;WAIT FOR COMPLETION OF DMA
0.1350              rts
0.1351
0.1352 ;################################
0.1353    ENDIMEM                                         ;END OF INTERNAL PROGRAM MEM
0.1354              org       p:$E000                     ;PROGRAM CONTINUES ON EXTERNAL
0.1355                                                    ;MEMORY
0.1356
0.1357 ;####### DO NUMBERCRUNCHING ######
0.1358
0.1359    WORK
0.1360              HAM_XY  WkAddr                        ;MPY WITH HAMMING WINDOW
0.1361              MulCar                                ;SHIFT UP IN FREQ
0.1362              Add_DQ                                ;ADD DIRECT AND QUADRATURE
0.1363
0.1364              ZeroPad   WkAddr+N,(N_FFT-N)          ;ZEROPAD
0.1365              fftr2c    N_FFT,WkAddr,A_Addr         ;DO COMPLEX FFT
0.1366              BitRev    WkAddr,Wsp,N_FFT/4          ;BIT REVERSE OUTPUT
0.1367              PerG      WkAddr,N_FFT/4              ;PERIODOFRAM
0.1368              ChrShft   WkAddr,N_FFT/4              ;PREPARE FOR TRANSFER
0.1369
0.1370              rts
0.1371    END
0.1372 ;************ PROGRAM END ***************
```

PC SOURCE CODE LISTING (1.0000-3.0067)

```
1.0000 /* ************************************************
1.0001 /*
1.0002 /*                          CW DOPPLER ULTRASOUND
1.0003 /*
1.0004 /* ************************************************
1.0005
1.0006 // Copyright(C) L.SMITH
1.0006 // Department of Biomedical Engineering
1.0007 // University of Cape Town
1.0008 // Observatory
1.0009 // Cape Town
1.0010 // New South Africa
1.0011 // (021) 471250 x 235
1.0012
1.0013 // MISC FUNCTIONS FOR WA_MAIN.C
1.0014
1.0015 #include "intface.h"
1.0016 #include "wa_main.h"
1.0017
1.0017 //*************** GLOBAL VARIABLES **************
1.0018 extern unsigned long int StartT,ElapsT;
1.0019 extern int far TW,TH;
1.0020 extern int MaxX,MaxY,clip,Ret_Main;
1.0021
1.0022 //*********** STRUCTURE DECLARATIONS ***********
1.0023
1.0024 struct MenuWindow Menu[MNum]=
1.0025     {          //MAIN MENU (0)
1.0026                { " Capture Save Import MeanF LookW Ex Quit",
1.0027                  0,0,639,28,14,6,2,
1.0028                  "CSIMLEQ",
1.0028                  {&Go,&Menu_Man,&Get_FileN,&MeanF,&LookW,&Menu_Man,&Quit},
1.0029                  {NULL,3,1,NULL,NULL,1,NULL},
1.0030                  NULL
1.0031                },
1.0032
1.0033                //SUB MENU (1)
1.0034                { "Sub\nMenu\nExample",
1.0035                  10,40,90,100,14,6,1,
1.0036                  "SME",
1.0037                  {&Menu_Man,&Menu_Man,&Menu_Man},
1.0038                  {2,2,2},
1.0039                  NULL
1.0039                },
1.0040
1.0041                //SUB_SUB MENU (2)
1.0042                { "Another\nLevel\nDown",
1.0043                  60,120,140,180,14,6,1,
1.0044                  "ALD",
1.0045                  {&Test,&Test,&Test},
1.0046                  {NULL,NULL,NULL},
1.0047                  NULL
1.0048                },
1.0049
1.0050                //SAVE SUB MENU (3)
1.0050                { "Overwrite\nNew",
1.0051                  67,40,157,85,14,6,1,
1.0052                  "ON",
```

```
1.0053                  {&WriteFile,&Get_FileN},
1.0054                  {NULL,0},
1.0055                  NULL
1.0056                 }
1.0057
1.0058 };
1.0059
1.0060 struct DisplayWindow DisW[DWNum]=
1.0061     {
1.0061                 {1,"SPECTROGRAM","Time (sec)","Frequency (kHz)",
1.0062                  14,0,15,NULL,
1.0063                  75,76,560,418,     // viewing area 2x256x400
1.0064                  {NULL,NULL,NULL,NULL},
1.0065                  {NULL,NULL,NULL,NULL}, //XOFF (graph wraparound)
1.0066                  NULL
1.0067                 },
1.0068
1.0069                 {1,"DSP56001 Output 1","","",
1.0070                  14,0,15,6,
1.0071                  5,329,319,479,
1.0071                  {NULL,NULL,NULL,NULL},
1.0072                  {NULL,NULL,NULL,NULL}, //ymax,xoff,width
1.0073                  NULL
1.0074                 },
1.0075
1.0076                 {1,"DSP56001 Output 2","","",
1.0077                  14,0,15,6,
1.0078                  322,329,636,479,
1.0079                  {NULL,NULL,NULL,NULL},
1.0080                  {NULL,NULL,NULL,NULL}, //ymax,xoff,width
1.0081                  NULL
1.0082                 },
1.0082
1.0083                 {1,"X-Memory","Time","Amplitude",
1.0084                  14,0,15,6,
1.0085                  5,64,191,214,
1.0086                  {NULL,NULL,NULL,NULL},
1.0087                  {NULL,NULL,NULL,NULL}, //ymax,xoff,width
1.0088                  NULL
1.0089                 },
1.0090
1.0091                 {1,"Y-Memory","Time","Amplitude",
1.0092                  14,0,15,6,
1.0093                  5,280,191,430,
1.0093                  {NULL,NULL,NULL,NULL},
1.0094                  {NULL,NULL,NULL,NULL}, //ymax,xoff,width
1.0095                  NULL
1.0096                 },
1.0097
1.0098                 {1,"Spectrum-F","Frequency","Amplitude",
1.0099                  14,0,15,6,
1.0100                  205,64,391,214,
1.0101                  {NULL,NULL,NULL,NULL},
1.0102                  {NULL,NULL,NULL,NULL}, //ymax,xoff,width
1.0103                  NULL
1.0104                 },
1.0104
1.0105                 {1,"Spectrum-R","Frequency","Amplitude",
```

```
1.0106                  14,0,15,6,
1.0107                  205,280,391,430,
1.0108                  {NULL,NULL,NULL,NULL},
1.0109                  {NULL,NULL,NULL,NULL},  //ymax,xoff,width
1.0110                  NULL
1.0111                 }
1.0112
1.0113 };
1.0114
1.0115 //################################################
1.0115 void Init_Graph(void)
1.0116 {
1.0117 #define SVGA                    134 // Code given by BGI toolkit
1.0118 #define SVGA640x480v5F          2    // Mode 5Fh 640x480x256
1.0119
1.0120     int GraphDriver,GraphMode,ErrorCode;
1.0121
1.0122     GraphDriver = SVGA;
1.0123     GraphMode = SVGA640x480v5F;
1.0124     installuserdriver("PARADISE",NULL);
1.0125     initgraph(&GraphDriver,&GraphMode,"");
1.0126     ErrorCode = graphresult();
1.0126     if (ErrorCode != grOk)
1.0127     {
1.0128                 sound(1000);
1.0129                 delay(500);
1.0130                 nosound();
1.0131                 exit(20);
1.0132     }
1.0133
1.0134     return;
1.0135 }
1.0136
1.0136 //################################################
1.0137 void Test(void)
1.0138 // DEBUGGING
1.0139 {
1.0140     Ret_Main = 1;
1.0141     return;
1.0142 }
1.0143
1.0144 //################################################
1.0145 void Listprint(int l,int t,int fcol,char *list,char *PossKey)
1.0146 // PRINTS THE GIVEN OPTIONS AND HIGHLIGHTS THE HOTKEY.
1.0147 {
1.0147     char  str[200];
1.0148     int   xl,xloc,yloc;
1.0149     char  *strptr,*charptr,*keyptr,shortstr[2];
1.0150
1.0151     xl = l+TW;
1.0152     xloc = xl;
1.0153     yloc = t+(int)(TH/2);
1.0154
1.0155     stpcpy(str,list);
1.0156     setcolor(fcol);
1.0157
1.0158     charptr = str;
1.0158     keyptr = PossKey;
```

```
1.0159    strptr = strtok(str,"\n");
1.0160
1.0161    do
1.0162    {
1.0163              do
1.0164              {
1.0165                        shortstr[0] = *charptr;
1.0166                        shortstr[1] = '\0';
1.0167
1.0168                        if (*keyptr == shortstr[0])
1.0169                        {
1.0169                                  setcolor(15);
1.0170                                  keyptr++;
1.0171                        }
1.0172
1.0173                        outtextxy(xloc,yloc,shortstr);
1.0174                        setcolor(fcol);
1.0175
1.0176                        xloc += TW;
1.0177                        charptr++;
1.0178              }while(*charptr != '\0');
1.0179
1.0180              xloc = xl-TW;  //ALLOW FOR EXTRA NULL CHAR INSERTED
1.0180              yloc += TH+2;
1.0181              strptr = strtok(NULL,"\n");
1.0182
1.0183    }while(strptr != NULL);
1.0184
1.0185    return;
1.0186 }
1.0187
1.0188 //################################################
1.0189 /*
1.0190 TYPE OF OPERATIONS THAT MAY BE PERFORMED ON THE VARIOUS WINDOWS :
1.0191
1.0191      (1) OPEN WITH SINGLE BORDER
1.0192      (2) OPEN WITH DOUBLE BORDER
1.0193      (3) "SOLID" BORDER AROUND WINDOW
1.0194      (4) "BROKEN" BORDER AROUND WINDOW
1.0195      (5) CLEARING THE WINDOW
1.0196      (6) RECTANGLE OF GIVEN COLOUR
1.0197
1.0198 */
1.0199
1.0200 void GrWin(int l,int t,int r,int b,int fcol,int bcol,int type_op)
1.0201 {
1.0202    setcolor(fcol);
1.0202    switch (type_op)
1.0203    {
1.0204              case 1 : {
1.0205                                  setfillstyle(SOLID_FILL,bcol);
1.0206                                  bar(l,t,r,b);
1.0207                                  rectangle(l,t,r,b);
1.0208                        }break;
1.0209
1.0210              case 2 : {
1.0211                                  setfillstyle(SOLID_FILL,bcol);
1.0212                                  bar(l,t,r,b);
```

```
1.0212                                  rectangle(l,t,r,b);
1.0213                                  rectangle(l+3,t+3,r-3,b-3);
1.0214                          }break;
1.0215
1.0216          case 3 : {
1.0217                                  setlinestyle(0,0,NORM_WIDTH);
1.0218                                  rectangle(l,t,r,b);
1.0219                          }break;
1.0220
1.0221          case 4 : {
1.0222                                  setcolor(bcol);
1.0223                                  rectangle(l,t,r,b);
1.0223                                  setcolor(fcol);
1.0224                                  setlinestyle(3,0,NORM_WIDTH);
1.0225                                  rectangle(l,t,r,b);
1.0226                          }break;
1.0227
1.0228          case 5 : {
1.0229                                  setfillstyle(SOLID_FILL,bcol);
1.0230                                  bar(l+1,t+1,r-1,b-1);
1.0231                          }break;
1.0232
1.0233          case 6 : {
1.0234                                  setcolor(fcol);
1.0234                                  rectangle(l,t,r,b);
1.0235                          }break;
1.0236
1.0237  }
1.0238  setlinestyle(0,0,NORM_WIDTH);
1.0239  return;
1.0240 }
1.0241
1.0242 //###############################################
1.0243 char far *GrSave(int l,int t,int r,int b)
1.0244 // SAVE THE BACKGROUND TO ENABLE RESTORATION OF THE
1.0245 // ORIGINAL IMAGE.
1.0245 {
1.0246  unsigned size;
1.0247  char far *Bitmapptr;
1.0248
1.0249  if ((size = imagesize(l,t,r,b)) <= 0)
1.0250  {
1.0251          sound(500);
1.0252          delay(500);
1.0253          nosound();
1.0254          Bitmapptr=NULL;
1.0255          return(Bitmapptr);
1.0256  }
1.0256  if ((Bitmapptr = (char far *)farmalloc(size)) == (char far *)NULL)
1.0257  {
1.0258          sound(500);
1.0259          delay(500);
1.0260          nosound();
1.0261          return(Bitmapptr);
1.0262  }
1.0263  getimage(l,t,r,b,Bitmapptr);
1.0264  return(Bitmapptr);
1.0265 }
```

```
1.0266
1.0267 //##############################################
1.0267 void GrRestore(int l,int t,char far *Bitmapptr)
1.0268 // RESTORE THE BACKGROUND IMAGE.
1.0269 {
1.0270      putimage(l,t,Bitmapptr,COPY_PUT);
1.0271      farfree(Bitmapptr);
1.0272      return;
1.0273 }
1.0274
1.0275 //##############################################
1.0276 void DefaultWin(void)
1.0277 // OPENS DEFAULT DISPLAY WINDOWS.
1.0277 // NOTE THAT IMAGESIZE() WILL NOT ALLOW US TO SAVE WINDOWS
1.0278 // LARGER THAN 64k
1.0279 {
1.0280 #define D DisW
1.0281      GrWin(D[0].L,D[0].T,D[0].R,D[0].B,D[0].fcol,D[0].bcol,1);
1.0282      Spec_Prep(0);
1.0283
1.0284      /* WINDOWS USED FOR DEBUGGING
1.0285      GrWin(D[1].L,D[1].T,D[1].R,D[1].B,D[1].fcol,D[1].bcol,1);
1.0286      XY_Prep(1);
1.0287
1.0288      GrWin(D[2].L,D[2].T,D[2].R,D[2].B,D[2].fcol,D[2].bcol,1);
1.0288      XY_Prep(2);
1.0289      */
1.0290
1.0291      return;
1.0292 #undef D DisW
1.0293 }
1.0294
1.0295 //##############################################
1.0296 void Menu_Man(int Num)
1.0297 // MENU MANAGER. THE VARIOUS MENU WINDOWS ARE CALLED FROM
1.0298 // HERE IN A RECURSIVE MANNER.
1.0299 {
1.0299 #define Mn Menu[Num]
1.0300      char Option;
1.0301      int  OptionNr;
1.0302
1.0303      if ((Mn.Mapptr = GrSave(Mn.L,Mn.T,Mn.R,Mn.B)) != NULL)
1.0304      {
1.0305                GrWin(Mn.L,Mn.T,Mn.R,Mn.B,Mn.fcol,Mn.bcol,Mn.type_op);
1.0306                Listprint(Mn.L,Mn.T,Mn.fcol,Mn.Menulist,Mn.KeyList);
1.0307                if (Num == 0)DefaultWin();
1.0308                while (((Option = toupper(getch())) != 27) ||
1.0309                          ((Option == 27) && (Num == 0)))
1.0310                {
1.0310                     if (Option == 0x0)
1.0311                               Option = getch();
1.0312                     OptionNr = strcspn(Mn.KeyList,&Option);
1.0313                     if(OptionNr < strlen(Mn.KeyList))
1.0314                     {
1.0315                               (*Mn.ProcPointer[OptionNr])(Mn.ArgList[OptionNr]);
1.0316                               if(Num == 0) Ret_Main = 0;
1.0317                               if(Ret_Main == 1)break;
1.0318                     }
```

```
1.0319                  }
1.0320                  if (Num != 0)GrRestore(Mn.L,Mn.T,Mn.Mapptr);
1.0321      }
1.0321
1.0322      return;
1.0323 #undef Mn
1.0324 }
1.0325
1.0326 //################################################
1.0327 void BackWin(int col)
1.0328 // BACKGROUND DISPLAY.
1.0329 {
1.0330      setfillstyle(HATCH_FILL,col);
1.0331      bar(0,36,MaxX,MaxY);
1.0332      setcolor(col);
1.0332      rectangle(0,36,MaxX,MaxY);
1.0333      setfillstyle(SOLID_FILL,col);
1.0334      return;
1.0335 }
1.0336
1.0337 //################################################
1.0338 void Spec_Prep(int n)
1.0339 // PREPARE THE SPECTROGRAM DISPLAY WINDOW. AXIS,TITLE AND
1.0340 // THE COLORBAR ARE DISPLAYED.
1.0341 {
1.0342 #define D DisW
1.0342
1.0343      int dist,*intptr,ylabpos,xlabpos,titlepos;
1.0344      int xst,xdiv,yst,ydiv,i,j,m;
1.0345      char val[3],str[80];
1.0346
1.0347      intptr = D[n].Act;
1.0348      dist = 3*TH;
1.0349
1.0350      xlabpos = (int)((D[n].R+D[n].L-textwidth(D[n].xlabel))/2);
1.0351      titlepos = (int)((D[n].R+D[n].L-textwidth(D[n].title))/2);
1.0352      ylabpos = (int)((D[n].B+D[n].T+textwidth(D[n].ylabel))/2);
1.0353
1.0353      setcolor(D[n].fcol);
1.0354      rectangle(D[n].L+dist,D[n].T+dist,D[n].R-dist,D[n].B-dist);
1.0355
1.0356      setcolor(D[n].Icol);
1.0357      outtextxy(xlabpos,D[n].B-TH-1,D[n].xlabel);
1.0358      outtextxy(titlepos,D[n].T+TH,D[n].title);
1.0359      settextstyle(DEFAULT_FONT,VERT_DIR,0);
1.0360      outtextxy(D[n].L+TH+1,ylabpos,D[n].ylabel);
1.0361
1.0362      *intptr = D[n].L+dist+1;  intptr++; // SET ACTIVE VIEWING COORD.
1.0363      *intptr = D[n].T+dist+1;  intptr++;
1.0364      *intptr = D[n].R-dist-1;  intptr++;
1.0364      *intptr = D[n].B-dist-1;
1.0365
1.0366      rectangle(D[n].Act[2]+10,D[n].Act[1]+47,D[n].R-10,D[n].Act[3]-48);
1.0367      yst = D[n].Act[3]-49;
1.0368      for(i = PalOff;i < PalOff+ColLevels; i++)
1.0369      {
1.0370                  for(m = 0;m < 10; m++)
1.0371                  {
```

```
1.0372                          for(j = D[n].Act[2]+11;j < D[n].R-11; j++)
1.0373                          {
1.0374                                  putpixel(j,yst,i);
1.0375                          }
1.0375                          yst--;
1.0376                  }
1.0377  }
1.0378
1.0379  setcolor(D[n].fcol);
1.0380  settextstyle(DEFAULT_FONT,HORIZ_DIR,0);
1.0381
1.0382  D[n].MisC[0] = D[n].Act[0];
1.0383  line(D[n].Act[0],(D[n].Act[1]+D[n].Act[3])/2,
1.0384  D[n].Act[2],(D[n].Act[1]+D[n].Act[3])/2);
1.0385
1.0386  setcolor(D[n].Icol);
1.0386  yst = (int)((D[n].Act[1]+D[n].Act[3])/2-5);
1.0387  ydiv = (int)((D[n].Act[3]-D[n].Act[1])/10);
1.0388  for(i = 0;i <= 5; i++)
1.0389  {
1.0390          outtextxy(D[n].Act[0]-18,yst,itoa(i,val,10));
1.0391          yst -= ydiv;
1.0392  }
1.0393
1.0394  yst = (int)((D[n].Act[1]+D[n].Act[3])/2-5);
1.0395  for(i = 1;i <= 5; i++)
1.0396  {
1.0397          yst += ydiv;
1.0397          outtextxy(D[n].Act[0]-26,yst,itoa(-i,val,10));
1.0398  }
1.0399
1.0400  xst = D[n].Act[0]-3;
1.0401  xdiv = (int)((D[n].Act[2]-D[n].Act[0])/6);
1.0402  for(i = 0;i <= 30;i += 5)
1.0403  {
1.0404          outtextxy(xst,D[n].Act[3]+12,gcvt((double)(i)/10,4,str));
1.0405          xst += xdiv;
1.0406  }
1.0407
1.0407  return;
1.0408 #undef D DisW
1.0409 }
1.0410
1.0411 //################################################
1.0412 void XY_Prep(int n)
1.0413 // PREPARES X-Y DISPLAY WINDOWS.
1.0414 {
1.0415 #define D DisW
1.0416  int dist,*intptr,ylabpos,xlabpos,titlepos;
1.0417
1.0418  intptr = D[n].Act;
1.0418  dist = 2*TH;
1.0419
1.0420  xlabpos = (int)((D[n].R+D[n].L-textwidth(D[n].xlabel))/2);
1.0421  titlepos = (int)((D[n].R+D[n].L-textwidth(D[n].title))/2);
1.0422  ylabpos = (int)((D[n].B+D[n].T+textwidth(D[n].ylabel))/2);
1.0423
1.0424  setcolor(D[n].fcol);
```

```
1.0425      rectangle(D[n].L+dist,D[n].T+dist,D[n].R-dist,D[n].B-dist);
1.0426
1.0427      setcolor(D[n].Icol);
1.0428      outtextxy(xlabpos,D[n].B-TH-1,D[n].xlabel);
1.0429      outtextxy(titlepos,D[n].T+(TH/2),D[n].title);
1.0429      settextstyle(DEFAULT_FONT,VERT_DIR,0);
1.0430      outtextxy(D[n].L+TH+1,ylabpos,D[n].ylabel);
1.0431
1.0432      *intptr = D[n].L+dist+1;  intptr++; // SET ACTIVE VIEWING COORD.
1.0433      *intptr = D[n].T+dist+1;  intptr++;
1.0434      *intptr = D[n].R-dist-1;  intptr++;
1.0435      *intptr = D[n].B-dist-1;
1.0436
1.0437      setcolor(D[n].fcol);
1.0438      settextstyle(DEFAULT_FONT,HORIZ_DIR,0);
1.0439
1.0440      D[n].MisC[0] = (int)((D[n].Act[3]-D[n].Act[1])/2);
1.0440      D[n].MisC[1] = D[n].Act[0];
1.0441
1.0442      if((n == 3) | (n == 4))
1.0443                  D[n].MisC[2] = (int)((D[n].Act[2]-D[n].Act[0])/NRaw);
1.0444      if((n == 5) | (n == 6))
1.0445                  D[n].MisC[2] = (int)((D[n].Act[2]-D[n].Act[0])/NR);
1.0446
1.0447      return;
1.0448 #undef D DisW
1.0449 }
1.0450
1.0451 //################################################
1.0451 unsigned long int TimeElapsed(void)
1.0452 // TIME CALCULATION USED DURING DEBUGGGING PROCESS.
1.0453 {
1.0454      unsigned long int t,*tix;
1.0455
1.0456      tix = (unsigned long int *)0x46C;
1.0457      t = (*tix)*55;
1.0458
1.0459      return(t-StartT);
1.0460 }
1.0461
1.0462 //################################################
1.0462 void TimerStart(void)
1.0463 // TIME CALCULATION USED DURING DEBUGGGING PROCESS.
1.0464 {
1.0465      unsigned long int *tix;
1.0466
1.0467      tix = (unsigned long int *)0x46C;
1.0468      StartT = (*tix)*55;
1.0469      return;
1.0470 }
1.0471
1.0472 //################################################
1.0472 void Quit(void)
1.0473 {
1.0474      fcloseall();
1.0475      cleardevice();
1.0476      closegraph();
1.0477      exit(0);
```

```
1.0478 }
1.0479
1.0480 //##############################################
```

```
2.0000 //*************************************************
2.0001 //                ACQUISITION AND ANALYSIS OF
2.0002 //                LOWER LIMB FLOW WAVEFORMS
2.0003 //*************************************************
2.0003
2.0004 // Copyright(C) L.SMITH
2.0005 // Department of Biomedical Engineering
2.0005 // University of Cape Town
2.0006 // Observatory
2.0006 // Cape Town
2.0007 // New South Africa
2.0008 // (021) 471250 x 235
2.0008
2.0009 /* PLEASE NOTE THE FOLLOWING :
2.0010
2.0010   1.FONT=DEFAULT_FONT
2.0011   2.USERCHARSIZE = 8X14 FOR MENUS
2.0011   3.SETUSERCHARSIZE DOES NOT WORK WITH PARADISE DRIVER !!
2.0012   4.VGA RESOLUTION : 640X480X256 COLOURS IS USED (MODE $5F)
2.0013   5.THE REVERSE FREQUENCY ARRAY IS REVERSED : DUE TO TYPE OF
2.0013     DEMODULATION USED ON DSP BOARD.
2.0014
2.0014 */
2.0015
2.0016 #include "intface.h"
2.0016 #include "wa_main.h"
2.0017 #define        MaxRow 200
2.0017
2.0018 //****************** CONSTANTS ********************
2.0019 //CARDADDRESS
2.0019 const word CardAddr = 0x2b0;
2.0020
2.0021 //2^23 : 24 BIT TRANSFERS FROM DSP56001
2.0021 const unsigned long MAXVALUE = 8388608;
2.0022
2.0022 typedef union _addr
2.0023 {
2.0024     dword l;
2.0024     struct
2.0025     {
2.0025                word lo;
2.0026                word hi;
2.0027     }w;
2.0027 }_addrtype;
2.0028
2.0028 //*************** GLOBAL VARIABLES **************
2.0029 int            far TW,TH,srow,frow;
2.0030 int            MaxX,MaxY,clip,Ret_Main;
2.0030 int            Shft,ChkISR,HF0,HF2,row;
2.0031 int            ICReg = 0;
2.0032 char           FileName[10] = "";
2.0032 int            Bvel_flag = 0;
2.0033 unsigned long int StartT,ElapsT,maxval;
2.0033
2.0034 int            BVel[MaxRow];
2.0035 char           ForFreq[MaxRow][NR];
2.0035 char           RevFreq[MaxRow][NR];
2.0036 char           *FFreq[MaxRow];
```

```
2.0036 char             *RFreq[MaxRow];
2.0037 int              far XRaw[MaxRow][NRaw];
2.0038 int              far YRaw[MaxRow][NRaw];
2.0038 int              far *Xmem[MaxRow];
2.0039 int              far *Ymem[MaxRow];
2.0039
2.0040 //long          huge ForFreq[NR];    // USEFUL FOR DEBUGGING (24 BIT TRANSFER)
2.0041 //long          huge RevFreq[NR];
2.0041
2.0042 //***************** STRUCTURES *****************
2.0043 struct palette
2.0043 {
2.0044     int color[ColLevels][3];
2.0044 }pal;
2.0045
2.0046 extern struct MenuWindow Menu[MNum];
2.0046 extern struct DisplayWindow DisW[DWNum];
2.0047
2.0047 //############## MAIN PROGRAM START ############
2.0048 void main(void)
2.0049 {
2.0049     int ChrMax = 128;
2.0050
2.0050     Init_Graph();
2.0051     cleardevice();
2.0052     InitPalette();
2.0052
2.0053     // DEFAULT VALUES USED
2.0053     TW = 8;
2.0054     TH = 14;
2.0055     MaxX = 639;
2.0055     MaxY = 479;
2.0056     maxval = MAXVALUE;
2.0057
2.0057     // INITIALIZE POINTERS
2.0058     for(row=0;row<MaxRow;row++)
2.0058     {
2.0059      FFreq[row] = &ForFreq[row][0];
2.0060      RFreq[row] = &RevFreq[row][0];
2.0060      Xmem[row] = &XRaw[row][0];
2.0061      Ymem[row] = &YRaw[row][0];
2.0061     }
2.0062     row=0;
2.0063
2.0063     // USE THE FOLLOWING WHEN 24 BIT TRANSFERS ARE DONE
2.0064     // Shft = (int)(log(((float)(maxval))/(ColLevels))/log(2));
2.0064     Shft = (int)(log(((float)(ChrMax))/(ColLevels))/log(2));
2.0065
2.0066     // INITIALIZE DSP BOARD
2.0066     do
2.0067     {
2.0068      Reset56(CardAddr,0);
2.0068      ChkISR = inportb(CardAddr+2);
2.0069      if(ChkISR != 6)
2.0069      {
2.0070                  sound(500);
2.0071                  delay(100);
2.0071                  nosound();
```

```
2.0072      }
2.0072     }while(ChkISR != 6);
2.0073     Load56(CardAddr,"dsp_dop");
2.0074     Go56(CardAddr);
2.0074
2.0075     // STARTUP OF USER INTERFACE
2.0075     BackWin(3);
2.0076     Menu_Man(0);
2.0077     return;
2.0077 }
2.0078
2.0079 //################ FUNCTIONS USED #################
2.0079 void InitPalette(void)
2.0080
2.0080 // SET UP PALETTE FOR USE BY SPECTROGRAM
2.0081 // INTENSITIES RANGE FROM
2.0082 //   LOW BLUE=>GREEN=>YELLOW=>RED (HIGH)
2.0082 {
2.0083     unsigned int x,k,m,i;
2.0083
2.0084     x = (int)((MaxColInt-MinColInt)/(ColLevels>>2));
2.0085     k = x;
2.0085     m = ColLevels/4;
2.0086
2.0086     for(i=0;i<ColLevels;i++)
2.0087     {
2.0088      pal.color[i][0] = 0;
2.0088      pal.color[i][1] = 0;
2.0089      pal.color[i][2] = 0;
2.0089     }
2.0090
2.0091     for(i=0;i<ColLevels>>2;i++)
2.0091     {
2.0092      pal.color[i][2] = MinColInt+k;
2.0093      pal.color[i+m][1] = MinColInt+k;
2.0093      pal.color[i+2*m][0] = MinColInt+k;
2.0094      pal.color[i+2*m][1] = MinColInt+k;
2.0094      pal.color[i+3*m][0] = MinColInt+k;
2.0095      k += x;
2.0096     }
2.0096
2.0097     for(i = 0;i < ColLevels; i++)
2.0097     {
2.0098                setrgbpalette(PalOff+i,pal.color[i][0],pal.color[i][1],
2.0099                pal.color[i][2]);
2.0099     }
2.0100     return;
2.0100 }
2.0101
2.0102 //###############################################
2.0102 void SOUND(void)
2.0103 // SOUND A TONE. USED FOR BEGUGGING.
2.0104 {
2.0104     sound(500);
2.0105     delay(100);
2.0105     nosound();
2.0106     return;
2.0107 }
```

```
2.0107
2.0108 //##############################################
2.0108 void ValWindow(void)
2.0109
2.0110 // DISPLAY WINDOW OF MAXIMUM VALUE RECIEVED. USED FOR BEGUGGING.
2.0110 {
2.0111     char value[20];
2.0111
2.0112     GrWin(540,35,638,57,15,0,1);
2.0113     outtextxy(550,40,ultoa(maxval,value,10));
2.0113     return;
2.0114 }
2.0115
2.0115 //##############################################
2.0116 void ScaleVal(void)
2.0116
2.0117 // DISPLAY CURRENT MAXIMUM VALUE. USED FOR BEGUGGING.
2.0118 {
2.0118     char kbchar,value[20];
2.0119     long chval = 1000000;
2.0119     int  test = 1,l = 540,t = 35,r = 638,b = 57;
2.0120
2.0121     GrWin(l,t,r,b,15,0,4);
2.0121     outtextxy(550,40,ultoa(maxval,value,10));
2.0122     while(!kbhit()){}
2.0122
2.0123     while(test)
2.0124     {
2.0124               if (kbhit())
2.0125               {
2.0126                         kbchar = getch();
2.0126                         if(kbchar == 0xD)
2.0127                                   test = 0;
2.0127                         else
2.0128                         {
2.0129                                   switch(kbchar)
2.0129                                   {
2.0130                                   case 0x49 :{
2.0130                                                      GrWin(l,t,r,b,15,0,5);
2.0131                                                      if(maxval<(MAXVALUE-chval)) maxval += chval;
2.0132                                                      outtextxy(l+10,t+5,ultoa(maxval,value,10));
2.0132                                                      break;
2.0133                                                                }
2.0133
2.0134                                   case 0x51 :{
2.0135                                                      GrWin(l,t,r,b,15,0,5);
2.0135                                                      if(maxval>chval)
2.0136                                                                maxval -= chval;
2.0136                                                      else
2.0137                                                      {
2.0138                                                                if(maxval>100000)
2.0138                                                                          maxval -= 100000;
2.0139                                                                else
2.0140                                                                          if(maxval>10000)
2.0140                                                                                    maxval -= 10000;
2.0141                                                      }
2.0141                                                      outtextxy(l+10,t+5,ultoa(maxval,value,10));
2.0142                                                      break;
```

```
2.0143                                          }
2.0143
2.0144                          case 0x47 :{
2.0144                                          GrWin(l,t,r,b,15,0,5);
2.0145                                          maxval = MAXVALUE;
2.0146                                          outtextxy(l+10,t+5,ultoa(maxval,value,10));
2.0146                                          break;
2.0147                                  }
2.0147
2.0148                          case 0x4F :{
2.0149                                          GrWin(l,t,r,b,15,0,5);
2.0149                                          maxval = 2048;
2.0150                                          outtextxy(l+10,t+5,ultoa(maxval,value,10));
2.0151                                          break;
2.0151                                  }
2.0152
2.0152                          }
2.0153                  }
2.0154          }
2.0154  }
2.0155
2.0155  Shft = (int)(log(((float)(maxval))/(ColLevels))/log(2));
2.0156  GrWin(l,t,r,b,15,0,3);
2.0157  return;
2.0157 }
2.0158
2.0158 //################################################
2.0159 void Get_FileN(int type_op)
2.0160
2.0160 // GET FILE NAME TO BE IN/EXPORTED.
2.0161 // CHARACTERS BETWEEN VALUES 47 AND 91 ARE ALLOWED.
2.0162 // ENTER WILL ACCEPT OPTION AND ESC WILL RETURN.
2.0162 {
2.0163  char far *Map,fname[2],*ptr;
2.0163  int  l = 122,t = 45,r = 212,b = 90,x = l+4,y = t+5;
2.0164
2.0165  if ((Map = GrSave(l,t,r,b)) != NULL)
2.0165          GrWin(l,t,r,b,14,6,1);
2.0166  else return;
2.0166
2.0167  setcolor(14);
2.0168  gprintf(&x,&y,"Filename ?");
2.0168  setcolor(15);
2.0169
2.0169  sprintf(FileName,"");
2.0170  fname[1] = '\0';
2.0171
2.0171  while(((fname[0] = getch()) != 0x0D) && (strlen(FileName) < 8))
2.0172  {
2.0173          if((toupper(fname[0]) > 47) && (toupper(fname[0]) < 91))
2.0173          {
2.0174                  outtextxy(x+5,y+5,fname);
2.0174                  x += TW;
2.0175                  strcat(FileName,fname);
2.0176          }
2.0176
2.0177          if(fname[0] == 27)
2.0177          {
```

```
2.0178                          GrRestore(l,t,Map);
2.0179                          return;
2.0179                  }
2.0180
2.0180                  if(fname[0] == 0x08)
2.0181                  {
2.0182                          if(strlen(FileName) > 0)
2.0182                          {
2.0183                                  setcolor(6);
2.0183                                  x -= TW;
2.0184                                  fname[0] = 219;
2.0185                                  outtextxy(x+5,y+5,fname);
2.0185                                  setcolor(15);
2.0186                                  ptr = strrchr(FileName,'\0');
2.0187                                  ptr--;
2.0187                                  *ptr = '\0';
2.0188                          }
2.0188                  }
2.0189  }
2.0190
2.0190  switch(type_op)
2.0191  {
2.0191                  case 0 :WriteFile();break;
2.0192                  case 1 :ReadFile();break;
2.0193  }
2.0193
2.0194  GrRestore(l,t,Map);
2.0194  return;
2.0195 }
2.0196
2.0196 //###############################################
2.0197 void WriteFile(void)
2.0198
2.0198 // WRITE THE OUPUT FILES. ASCII VALUES ARE SAVED TO ENABLE
2.0199 // INPUT TO MATLAB. THE SELECTED FREQUNCY VALUES ARE STORED
2.0199 // IN .SPC EXTENSION AND THE RAW DATA IN A .RAW FILE.
2.0200 // NB! REMEMBER THAT DUE TO THE TYPE OF DEMODULATION USED THE
2.0201 //    REVERSE FREQUENCIES ARE IN REVERSED ORDER.
2.0201 {
2.0202  FILE    *stream;
2.0202  int     i,j;
2.0203  char    Path_Name[80] = "",path[50] = "",fext[4] = "",mext[4] = "";
2.0204
2.0204  sprintf(path,"c:\\bc\\files\\patdata\\");
2.0205  sprintf(fext,".spc");
2.0205  strcat(Path_Name,path);
2.0206  strcat(Path_Name,FileName);
2.0207  strcat(Path_Name,fext);
2.0207
2.0208  if((stream = fopen(Path_Name,"wt")) == NULL)
2.0209  {
2.0209          sound(100);
2.0210          delay(500);
2.0210          nosound();
2.0211          return;
2.0212  }
2.0212
2.0213  for(i = 0;i < MaxRow; i++)
```

```
2.0213    {
2.0214              for(j = 0;j < NR; j++)
2.0215                        fprintf(stream,"%d ",FFreq[i][j]);
2.0215              fprintf(stream,"\n");
2.0216    }
2.0216
2.0217    fprintf(stream,"\n");
2.0218    for(i = 0; i < MaxRow; i++)
2.0218    {
2.0219              for(j = 0; j<NR; j++)
2.0220                        fprintf(stream,"%d ",RFreq[i][j]);
2.0220              fprintf(stream,"\n");
2.0221    }
2.0221    fclose(stream);
2.0222
2.0223    sprintf(Path_Name,"");
2.0223    sprintf(mext,".raw");
2.0224    strcat(Path_Name,path);
2.0224    strcat(Path_Name,FileName);
2.0225    strcat(Path_Name,mext);
2.0226
2.0226    if((stream = fopen(Path_Name,"wt")) == NULL)
2.0227    {
2.0227              sound(100);
2.0228              delay(500);
2.0229              nosound();
2.0229              return;
2.0230    }
2.0230
2.0231    for(i = 0;i < MaxRow; i++)
2.0232    {
2.0232              for(j = 0;j < NRaw; j++)
2.0233                        fprintf(stream,"%d ",Xmem[i][j]);
2.0234              fprintf(stream,"\n");
2.0234    }
2.0235
2.0235    fprintf(stream,"\n");
2.0236    for(i = 0;i < MaxRow; i++)
2.0237    {
2.0237              for(j = 0;j < NRaw; j++)
2.0238                        fprintf(stream,"%d ",Ymem[i][j]);
2.0238              fprintf(stream,"\n");
2.0239    }
2.0240
2.0240    fclose(stream);
2.0241    SOUND();
2.0241
2.0242    Ret_Main = 1;
2.0243    return;
2.0243 }
2.0244
2.0245 //################################################
2.0245 void ReadFile(void)
2.0246
2.0246 // READ THE SELECTED INPUT FILES. THE FUNCTIONS ARE THE
2.0247 // REVERSE OF THOSE USED IN WriteFile().
2.0248 {
2.0248    FILE      *stream;
```

```
2.0249  int         i,j,raw_available = 1;
2.0249  char        Path_Name[80] = "",path[50] = "",fext[4] = "",mext[4] = "";
2.0250
2.0251  sprintf(path,"c:\\bc\\files\\patdata\\");
2.0251  sprintf(fext,".spc");
2.0252  strcat(Path_Name,path);
2.0252  strcat(Path_Name,FileName);
2.0253  strcat(Path_Name,fext);
2.0254
2.0254  if((stream=fopen(Path_Name,"rt"))==NULL)
2.0255  {
2.0256          sound(100);
2.0256          delay(500);
2.0257          nosound();
2.0257          return;
2.0258  }
2.0259
2.0259  for(i = 0;i < MaxRow; i++)
2.0260  {
2.0260   for(j = 0;j < NR; j++)
2.0261          fscanf(stream,"%d ",&FFreq[i][j]);
2.0262   fscanf(stream,"\n");
2.0262  }
2.0263
2.0263  fscanf(stream,"\n");
2.0264  for(i = 0;i < MaxRow; i++)
2.0265  {
2.0265   for(j = 0;j < NR; j++)
2.0266          fscanf(stream,"%d ",&RFreq[i][j]);
2.0267   fscanf(stream,"\n");
2.0267  }
2.0268  fclose(stream);
2.0268
2.0269  sprintf(Path_Name,"");
2.0270  sprintf(mext,".raw");
2.0270  strcat(Path_Name,path);
2.0271  strcat(Path_Name,FileName);
2.0271  strcat(Path_Name,mext);
2.0272
2.0273  if((stream = fopen(Path_Name,"rt")) == NULL)
2.0273  {
2.0274          sound(100);
2.0274          delay(500);
2.0275          nosound();
2.0276          for(i = 0;i < MaxRow; i++)
2.0276          {
2.0277           for(j = 0;j < NRaw; j++)
2.0277           {
2.0278                  Xmem[i][j] = 0;
2.0279                  Ymem[i][j] = 0;
2.0279           }
2.0280          }
2.0281          raw_available = 0;
2.0281  }
2.0282
2.0282  if(raw_available == 1)
2.0283  {
2.0284          for(i=0;i<MaxRow;i++)
```

```
2.0284                  {
2.0285                          for(j = 0;j < NRaw; j++)
2.0285                                  fscanf(stream,"%d ",&Xmem[i][j]);
2.0286                          fscanf(stream,"\n");
2.0287                  }
2.0287
2.0288                  fscanf(stream,"\n");
2.0288                  for(i = 0;i < MaxRow; i++)
2.0289                  {
2.0290                          for(j = 0;j < NRaw; j++)
2.0290                                  fscanf(stream,"%d ",&Ymem[i][j]);
2.0291                          fscanf(stream,"\n");
2.0292                  }
2.0292          }
2.0293
2.0293          fclose(stream);
2.0294          row = 0;
2.0295          Bvel_flag = 1;
2.0295          ReDraw();
2.0296          return;
2.0296 }
2.0297
2.0298 //################################################
2.0298 void ReDraw(void)
2.0299
2.0299 // FOLLOWING USER SELECTION THE CIRCULAR ARRAYS ARE
2.0300 // LINEARIZED TO ALLOW DATA DISPLAY FROM LEFT TO RIGHT.
2.0301 {
2.0301          int             i;
2.0302          int             far *Rptr[MaxRow];
2.0303          char            *Tptr[MaxRow];
2.0303
2.0304          if(row != 0)
2.0304          {
2.0305                  for(i = 0;i < MaxRow; i++)
2.0306                  {
2.0306                          (row == (MaxRow-1)) ? (row = 0) : (row++);
2.0307                          Tptr[i] = FFreq[row];
2.0307                          Rptr[i] = Xmem[row];
2.0308                  }
2.0309
2.0309                  for(i = 0;i < MaxRow; i++)
2.0310                  {
2.0310                          FFreq[i] = Tptr[i];
2.0311                          Xmem[i] = Rptr[i];
2.0312                  }
2.0312
2.0313                  for(i = 0;i < MaxRow; i++)
2.0313                  {
2.0314                          (row == (MaxRow-1)) ? (row = 0) : (row++);
2.0315                          Tptr[i] = RFreq[row];
2.0315                          Rptr[i] = Ymem[row];
2.0316                  }
2.0317
2.0317                  for(i = 0;i < MaxRow; i++)
2.0318                  {
2.0318                          RFreq[i] = Tptr[i];
2.0319                          Ymem[i] = Rptr[i];
```

```
2.0320                  }
2.0320      }
2.0321
2.0321      DisW[0].MisC[0]=DisW[0].Act[0];
2.0322      for(row=0;row<MaxRow;row++)
2.0323                  Spectrogram(row);
2.0323
2.0324      row = 0;
2.0324      SOUND();
2.0325      return;
2.0326 }
2.0326
2.0327 //################################################
2.0328 void Spectrogram(int row)
2.0328
2.0329 // DISPLAY THE SPECTROGRAM IN NEAR REAL TIME ON THE USER SCREEN.
2.0329 // NOTE DIRECTION OF REVERSE DISPLAY. THIS IS DUE TO THE
2.0330 // FREQUENCY DOMAIN PROCESS USED DURING DEMODULATION.
2.0331 // DUE TO GRAPHIC SPEED LIMITATIONS THE ENTIRE SPECTRUM WINDOW
2.0331 // CAN NOT BE SCROLLED IN THE DESIRED TIME FRAME AND THEREFORE
2.0332 // A CIRCULAR SCROLLING TECHNIQUE IS USED.
2.0332 {
2.0333      int         i,j = 0,x = DisW[0].MisC[0];
2.0334      int         y1 = DisW[0].Act[1],y2 = DisW[0].Act[3];
2.0334      char        *TF = FFreq[row];
2.0335      char        *TR = RFreq[row];
2.0335
2.0336      for(i = 0;i < (NR << 1);i += 2)
2.0337      {
2.0337                  putpixel(x,y1+i,(TF[NR-j-1]>>Shft)+PalOff);
2.0338                  putpixel(x,y2-i,(TR[j]>>Shft)+PalOff);
2.0339                  j++;
2.0339      }
2.0340
2.0340      (x == (DisW[0].Act[0] + VLine_DWn-2)) ? (DisW[0].MisC[0] =
2.0341      DisW[0].Act[0]) : (DisW[0].MisC[0] += 2);
2.0342      return;
2.0342 }
2.0343
2.0343 //################################################
2.0344 void signal_view(long huge *signal,int n,int nr)
2.0345
2.0345 // DISPLAY DATA (32 BIT) RECEIVED FROM DSP BOARD IN A SELECTED
2.0346 // WINDOW. USED DURING DEBUGGING.
2.0346 {
2.0347      int         i,xoff;
2.0348      long        yval[NRaw];
2.0348
2.0349      setviewport(DisW[n].Act[0],DisW[n].Act[1],DisW[n].Act[2],
2.0350      DisW[n].Act[3],clip);
2.0350      clearviewport();
2.0351      setviewport(0,0,MaxX,MaxY,clip);
2.0351      xoff = DisW[n].MisC[1];
2.0352
2.0353      for(i = 0;i < nr; i++)
2.0353      {
2.0354                  yval[i] = DisW[n].Act[1]+(long)(-((((float)(signal[i])/maxval)*
2.0354                  DisW[n].MisC[0])-DisW[n].MisC[0]));
```

```
2.0355    }
2.0356
2.0356    setcolor(DisW[n].Dcol);
2.0357    moveto(xoff,yval[0]);
2.0357    xoff += DisW[n].MisC[2];
2.0358
2.0359    for (i = 1;i < nr; i++)
2.0359    {
2.0360              lineto(xoff,yval[i]);
2.0360              xoff+=DisW[n].MisC[2];
2.0361    }
2.0362    return;
2.0362 }
2.0363
2.0364 //################################################
2.0364 void I_signal_view(int *signal,int n,int nr)
2.0365
2.0365 // DISPLAY DATA (16 BIT) RECEIVED FROM DSP BOARD IN A SELECTED
2.0366 // WINDOW. USED DURING DEBUGGING.
2.0367 {
2.0367    int  i,xoff;
2.0368    int yval[NRaw],A_Dmax = 2048;
2.0368
2.0369    setviewport(DisW[n].Act[0],DisW[n].Act[1],DisW[n].Act[2],
2.0370    DisW[n].Act[3],clip);
2.0370    clearviewport();
2.0371    setviewport(0,0,MaxX,MaxY,clip);
2.0371    xoff = DisW[n].MisC[1];
2.0372
2.0373    for (i = 0;i < nr; i++)
2.0373    {
2.0374              yval[i]=DisW[n].Act[1]+(int)(-((((float)(signal[i])/A_Dmax)*
2.0375              DisW[n].MisC[0])-DisW[n].MisC[0]));
2.0375    }
2.0376
2.0376    setcolor(DisW[n].Dcol);
2.0377    moveto(xoff,yval[0]);
2.0378    xoff += DisW[n].MisC[2];
2.0378
2.0379    for (i = 1;i < nr; i++)
2.0379    {
2.0380              lineto(xoff,yval[i]);
2.0381              xoff += DisW[n].MisC[2];
2.0381    }
2.0382    return;
2.0382 }
2.0383
2.0384 //################################################
2.0384 void C_signal_view(char *signal,int n,int nr)
2.0385
2.0386 // DISPLAY DATA (8 BIT) RECEIVED FROM DSP BOARD IN A SELECTED
2.0386 // WINDOW. USED DURING DEBUGGING.
2.0387 {
2.0387    int  i,xoff;
2.0388    int yval[NRaw],Cmax = 128;
2.0389
2.0389    setviewport(DisW[n].Act[0],DisW[n].Act[1],DisW[n].Act[2],
2.0390    DisW[n].Act[3],clip);
```

```
2.0390    clearviewport();
2.0391    setviewport(0,0,MaxX,MaxY,clip);
2.0392    xoff = DisW[n].MisC[1];
2.0392
2.0393    for (i = 0;i < nr; i++)
2.0393    {
2.0394              yval[i]=DisW[n].Act[1]+(int)(-(((((float)(signal[i])/Cmax)*
2.0395              DisW[n].MisC[0])-DisW[n].MisC[0]));
2.0395    }
2.0396
2.0397    setcolor(DisW[n].Dcol);
2.0397    moveto(xoff,yval[0]);
2.0398    xoff += DisW[n].MisC[2];
2.0398    for (i = 1;i < nr; i++)
2.0399    {
2.0400              lineto(xoff,yval[i]);
2.0400              xoff += DisW[n].MisC[2];
2.0401    }
2.0401    return;
2.0402 }
2.0403
2.0403 //################################################
2.0404 void DMA_Done(word Timeout)
2.0404
2.0405 // POLL DMA CONTROLLER TO ASSERTAIN DMA STATUS. THIS FUNCTION
2.0406 // WAS TAKEN FROM THE DSP DRIVER SOFTWARE AND OPTIMISED TO
2.0406 // GAIN A SPEED ADVANTAGE.
2.0407 {
2.0407    word pollstatus,timecount = 0;
2.0408    byte TCMask = 1 << DMA_Chan,DMask = DMA_Chan+4,State;
2.0409
2.0409    State = inportb(CardAddr);
2.0410    do
2.0411    {
2.0411              pollstatus = inportb(8);
2.0412    }while (((pollstatus & TCMask) == 0) && (++timecount <= Timeout));
2.0412    outportb(0xA,DMask);
2.0413    outportb(CardAddr,State&0x18);      //DISABLE DMA MODE
2.0414    if (timecount > Timeout)
2.0414    {
2.0415              printf("%d ",row);
2.0415              sound(100);
2.0416              delay(1);
2.0417              nosound();
2.0417    }
2.0418    return;
2.0418 }
2.0419
2.0420 //################################################
2.0420 int InTPoll56(word Timeout)
2.0421
2.0422 // POLL DMA CONTROLLER TO ASSERTAIN DMA STATUS. THIS FUNCTION
2.0422 // WAS TAKEN FROM THE DSP DRIVER SOFTWARE AND OPTIMISED TO
2.0423 // GAIN A SPEED ADVANTAGE.
2.0423 {
2.0424 #define DMAstatus 8
2.0425 #define DMAmaskS 0xA
2.0425
```

```
2.0426      word pollstatus,timecount = 0;
2.0426      byte TCMask = 1<<DMA_Chan,DMask = DMA_Chan+4;
2.0427
2.0428      do
2.0428      {
2.0429                  pollstatus=inportb(DMAstatus);
2.0429      }while (((pollstatus & TCMask) == 0) && (++timecount <= Timeout));
2.0430      outportb(DMAmaskS,DMask);
2.0431      return !(timecount<Timeout);
2.0431 }
2.0432
2.0433 //##############################################
2.0433 void InTDMA(byte icr,word DMApageno,word DMAoffset,word count,byte ICRmask)
2.0434
2.0434 // SETUP OF DMA CONTROLLER. THIS FUNCTION WAS TAKEN FROM THE
2.0435 // DSP DRIVER SOFTWARE AND OPTIMISED TO GAIN A SPEED ADVANTAGE.
2.0436 // (SEE IBM TECHNICAL REFERENCE MANUAL FOR DETAILS).
2.0436 {
2.0437      word PageOffs = 3,DMAcount,DMAaddr;
2.0437      byte ICR = icr,State;
2.0438
2.0439      State = inportb(CardAddr);
2.0439      outportb(0x80+PageOffs,DMApageno);
2.0440      outportb(0xB,DMA_Chan+4+(DMAdown<<5));
2.0440      outportb(0xA,DMA_Chan+4);
2.0441      outportb(0xC,0xFF);
2.0442      count--;
2.0442      DMAaddr = DMA_Chan<<1;
2.0443      DMAcount = DMAaddr+1;
2.0444      outportb(DMAaddr,DMAoffset&0xFF);
2.0444      outportb(DMAaddr,DMAoffset>>8);
2.0445      outportb(DMAcount,count&0xFF);
2.0445      outportb(DMAcount,count>>8);
2.0446      outportb(0xA,DMA_Chan);
2.0447      inportb(8);
2.0447      ICR |= ICRmask|0x80;
2.0448      ICR |= State&0x18;     // PRESERVE FLAGS IN DSP
2.0448      if(ICRmask) outportb(CardAddr,ICR);
2.0449 }
2.0450
2.0450 //##############################################
2.0451 void DMAIn56(byte icr,word segmentpart,word offsetpart,word count)
2.0451
2.0452 // SETUP OF DMA CONTROLLER. THIS FUNCTION WAS TAKEN FROM THE
2.0453 // DSP DRIVER SOFTWARE AND OPTIMISED TO GAIN A SPEED ADVANTAGE.
2.0453 // (SEE IBM TECHNICAL REFERENCE MANUAL FOR DETAILS).
2.0454 {
2.0454      _addrtype addr;
2.0455      word DMAoffset,DMApageno,DMAcount;
2.0456      byte ICRmask = 1;
2.0456
2.0457      addr.l = (dword)offsetpart+((dword)segmentpart<<4);
2.0458      DMApageno = addr.w.hi;
2.0458      DMAoffset = addr.w.lo;
2.0459
2.0459      if((DMAoffset+1)<count)
2.0460      {
2.0461                  DMAcount = DMAoffset+1;
```

```
2.0461                  InTDMA(icr,DMApageno,DMAoffset,DMAcount,ICRmask);
2.0462                  ICRmask = 0;
2.0462                  addr.l -= (dword) DMAcount;
2.0463                  count -= DMAcount;
2.0464                  DMApageno = addr.w.hi;
2.0464                  DMAoffset = addr.w.lo;
2.0465                  if(InTPoll56(10000))
2.0465                  {
2.0466                          printf("%d ",row);
2.0467                          sound(100);
2.0467                          delay(1);
2.0468                          nosound();
2.0469                  }
2.0469      }
2.0470      InTDMA(icr,DMApageno,DMAoffset,count,ICRmask);
2.0470 }
2.0471
2.0472 //################################################
2.0472 void DMAfromDSP(long huge *Freq)
2.0473
2.0473 // NOTE :
2.0474 // 1.INTEL USES LITTLE ENDIAN ORDER. (ADDRESS OF WORD = LSB)
2.0475 //   DSP SENDS MSB FIRST.
2.0475 // 2.24 BIT TRANSFERS ARE DONE. AN ADDITIONAL BYTE IS INSERTED
2.0476 //   TO TRANSFORM THE VALUE INTO A LONG.
2.0476 //   INVESTIGATE MSB FOR SIGN :
2.0477 //     IF NEG INSERT $FF FOR 2's COMPLIMENT
2.0478 //     IF POS INSERT $00.
2.0478 // 3.TRANSFER PROCESS IS REVERSED !!!!!
2.0479 //   => COMPENSATED FOR ON DSP CARD
2.0480 // 4.READ ISR (TIMING OF DMA) TO TEST IF HF2 IS SET
2.0480 // 5.HF0 CAN NOT BE PLACED HERE AS THE DSP IS TO QUICK BETWEEN
2.0481 //   SUCCESSIVE DMA TRANSFERS. (TIME BETWEEN CLEAR AND NEXT SET)
2.0481 // 6.VALUE WRITTEN TO ICR DEPENDS ON THE DESIRED BYTES PER TRANSFERED WORD
2.0482 //   ICR = 0x60             ;SINGLE BYTE ONLY
2.0483 //   ICR = 0x40             ;INTEGER TRANSFER
2.0483 //   ICR = 0x20             ;24 BIT WORDS
2.0484
2.0484 {
2.0485      long huge *ptr = &Freq[NR-1];
2.0486      byte *b1 = (byte *)(&Freq[NR/4]),*b2 = (byte *)(&Freq[0]);
2.0486      int  i;
2.0487
2.0487      do HF2 = inportb(CardAddr+2);
2.0488      while((HF2&8) != 8);
2.0489
2.0489      DMAIn56(0x20,FP_SEG(ptr),(FP_OFF(ptr)+3),3*NR);
2.0490      DMA_Done(10000);
2.0491
2.0491      for(i=0;i<NR;i++)
2.0492      {
2.0492                  *b2++=*b1++;
2.0493                  *b2++=*b1++;
2.0494                  *b2++=*b1;
2.0494                  if((*b1&128)==128)
2.0495                          *b2=0xFF;
2.0495                  else
2.0496                          *b2=0;
```

```
2.0497                  b2++;b1++;
2.0497      }
2.0498      return;
2.0498 }
2.0499
2.0500 //################################################
2.0500 void C_DMAfromDSP(char *Freq)
2.0501
2.0501 // RECIEVE CHAR (8 BIT) VALUE FROM DSP.
2.0502 {
2.0503      char *ptr = &Freq[NR-1];
2.0503
2.0504      do HF2 = inportb(CardAddr+2);
2.0505      while((HF2&8) != 8);
2.0505
2.0506      DMAIn56(0x60,FP_SEG(ptr),FP_OFF(ptr),NR);
2.0506      DMA_Done(10000);
2.0507      return;
2.0508 }
2.0508
2.0509 //################################################
2.0509 void I_DMAfromDSP(int *Signal)
2.0510
2.0511 // RECIEVE INT (16 BIT) VALUE FROM DSP.
2.0511 {
2.0512      int *ptr = &Signal[NRaw-1];
2.0512
2.0513      do HF2 = inportb(CardAddr+2);
2.0514      while((HF2&8) != 8);
2.0514
2.0515      DMAIn56(0x40,FP_SEG(ptr),FP_OFF(ptr)+1,NRaw<<1);
2.0516      DMA_Done(10000);
2.0516      return;
2.0517 }
2.0517
2.0518 //################################################
2.0519 void DMAtoDSP(long Ch[])
2.0519
2.0520 // TRANSFER 24 BIT VALUES TO DSP. THE LONG ARRAY IS FIRST MADE
2.0520 // INTO A 3 BYTE ARRAY THROUGH REMOVING THE FOURTH BYTE. THE
2.0521 // MAXIMUM VALUE TRANSFERRED IS THEREFORE 2^23.
2.0522 {
2.0522      byte *b1 = (byte *)(&Ch[0]),*b2 = (byte *)(&Ch[0]);
2.0523      int i;
2.0523
2.0524      for(i = 0;i < NS; i++)
2.0525      {
2.0525                  *b1++ = *b2++;
2.0526                  *b1++ = *b2++;
2.0527                  *b1++ = *b2++;
2.0527                  b2++;
2.0528      }
2.0528      b1--;
2.0529
2.0530      DMAWrite56(CardAddr,FP_SEG(b1),FP_OFF(b1),(3*NS),DMA_Chan,3,DMAdown);
2.0530      DMAPoll56(CardAddr,1,1000);
2.0531      return;
2.0531 }
```

```
2.0532
2.0533 //################################################
2.0533 void To_DSP(void)
2.0534
2.0534 // TRANSFER TWO ARRAYS TO THE DSP.
2.0535 // NB! THE MAXIMUM VALUE ALLOWED IS 2^23. (SEE DMAtoDSP())
2.0536 //
2.0536 // EX : TO SYNTHESIZE TWO QUADRATURE SIGNALS
2.0537 //  Ch1 = Bfsin(Wft)+Brsin(Wrt)
2.0537 //  Ch2 = Bfsin(Wft-pi/2)+Brsin(Wrt+pi/2)
2.0538 {
2.0539     int i;
2.0539     long Ch1[NS],Ch2[NS];
2.0540
2.0541     for(i = 0;i < NS; i++)
2.0541     {
2.0542                 Ch1[i] = (long)(1024e2*cos(2*3.1416*4000*i/Fs))
2.0542                                 +(long)(512e2*cos(2*3.1416*1000*i/Fs));
2.0543
2.0544                 Ch2[i] = (long)(1024e2*cos(2*3.1416*4000*i/Fs-1.5708))
2.0544                                 +(long)(512e2*cos(2*3.1416*1000*i/Fs+1.5708));
2.0545     }
2.0545
2.0546     DMAtoDSP(Ch1);
2.0547     DMAtoDSP(Ch2);
2.0547     return;
2.0548 }
2.0548
2.0549 //################################################
2.0550 void Time(void)
2.0550
2.0551 // CALCULATE TIME DURATION OF OPERATIONS. USED IN DEBUUGING.
2.0552 // TO USE THIS OPTION INSERT ANOTHER OPTION IN THE MAIN MENU
2.0552 // WHICH POINTS TO THIS FUNCTION.
2.0553 {
2.0553     int i;
2.0554
2.0555     TimerStart();
2.0555     outportb(CardAddr+ICReg,inportb(CardAddr+ICReg)|8);
2.0556     for(i = 1;i <= 200; i++)
2.0556     {
2.0557                 I_DMAfromDSP(Xmem[row]);
2.0558                 I_DMAfromDSP(Ymem[row]);
2.0558                 C_DMAfromDSP(RFreq[row]);
2.0559                 C_DMAfromDSP(FFreq[row]);
2.0559                 Spectrogram(row);
2.0560                 (row == (MaxRow-1)) ? (row=0) : (row++);
2.0561     }
2.0561     outportb(CardAddr+ICReg,inportb(CardAddr+ICReg)&0xF7);
2.0562
2.0563     ElapsT = TimeElapsed();
2.0563     printf("%ld",ElapsT);
2.0564     getch();
2.0564     return;
2.0565 }
2.0566
2.0566 //################################################
2.0567 void MeanF(void)
```

```
2.0567 // IN CALCULATING THE MEAN FREQUENCY WE SEARCH FOR THAT FREQUENCY SUCH
2.0568 // THAT THE ENERY CONTENT TO THE LEFT IS EQUAL TO THE ENERGY CONTENT TO
2.0569 // THE RIGHT.
2.0569 {
2.0570 #define D DisW
2.0570
2.0571     int  i,j,xoff,yoff,MaxFreq=5000;
2.0572     float FBin = MaxFreq/NR,EF,ER,Esum;
2.0572     char  *TF,*TR;
2.0573     int  yval[MaxRow];
2.0574
2.0574
2.0575     if(Bvel_flag != 1) return;
2.0575
2.0576     for(row = 0;row < MaxRow; row++)
2.0577     {
2.0577                EF = 0;
2.0578                ER = 0;
2.0578                TF = FFreq[row];
2.0579                TR = RFreq[row];
2.0580                for(j = 0;j < NR; j++)
2.0580                {
2.0581                          EF += TF[j];
2.0581                          ER += TR[j];
2.0582                }
2.0583                Esum = 0;
2.0583                j = 0;
2.0584                do
2.0584                {
2.0585                          Esum += TF[j];
2.0586                          j++;
2.0586                }while(Esum < (EF/2));
2.0587                BVel[row] = (int)((EF*FBin*j)/(EF+ER));
2.0588
2.0588                Esum = 0;
2.0589                j = 0;
2.0589                do
2.0590                {
2.0591                          Esum += TR[NR-1-j];
2.0591                          j++;        // NEG FREQ ARRAY REVERSED!!
2.0592                }while(Esum < (ER/2));
2.0592                BVel[row] -= (int)((ER*FBin*j)/(EF+ER));
2.0593     }
2.0594
2.0594     yoff = (int)((DisW[0].Act[3]+DisW[0].Act[1])/2);
2.0595     xoff = DisW[0].Act[0];
2.0595
2.0596     for (i=0;i<MaxRow;i++)
2.0597               yval[i]=yoff-(int)((((float)(BVel[i]))/MaxFreq)*(DisW[0].Act[3]-yoff));
2.0597
2.0598     setcolor(15);
2.0599     moveto(xoff,yval[0]);
2.0599     xoff += 2;
2.0600     for (i = 1;i < MaxRow; i++)
2.0600     {
2.0601               lineto(xoff,yval[i]);
2.0602               xoff += 2;
2.0602     }
```

```
2.0603
2.0603      while(!kbhit()){}
2.0604
2.0605      // REDRAW THE SPECTROGRAM TO OVERWRITE THE VELOCITY DISPLAY.
2.0605      GrWin(D[0].Act[0],D[0].Act[1],D[0].Act[2],D[0].Act[3],D[0].fcol,D[0].bcol,1);
2.0606      Spec_Prep(0);
2.0606      DisW[0].MisC[0]=DisW[0].Act[0];
2.0607      for(row=0;row<MaxRow;row++)
2.0608                  Spectrogram(row);
2.0608      row = 0;
2.0609
2.0610 #undef D DisW
2.0610 return;
2.0611 }
2.0611
2.0612 //################################################
2.0613 void WaveS(void)
2.0613
2.0614 // SAVE THE VELOCITY WAVEFORM AFTER CALCULATION. A .VEL
2.0614 // FILE EXTENSION IS USED.
2.0615 {
2.0616      int i;
2.0616      FILE *stream;
2.0617      char Path_Name[80] = "",path[50] = "",fext[4] = "";
2.0617
2.0618      sprintf(Path_Name,"");
2.0619      sprintf(path,"c:\\bc\\files\\patdata\\");
2.0619      sprintf(fext,".vel");
2.0620      strcat(Path_Name,path);
2.0621      strcat(Path_Name,FileName);
2.0621      strcat(Path_Name,fext);
2.0622
2.0622      if((stream=fopen(Path_Name,"wt"))==NULL)
2.0623      {
2.0624                  sound(100);
2.0624                  delay(500);
2.0625                  nosound();
2.0625                  return;
2.0626      }
2.0627
2.0627      for(i=0;i<MaxRow;i++)
2.0628                  fprintf(stream,"%d\n",BVel[i]);
2.0628
2.0629
2.0630      fclose(stream);
2.0630      SOUND();
2.0631      return;
2.0631 }
2.0632
2.0633 //################################################
2.0633 int gprintf(int *xloc,int *yloc,char *fmt, ... )
2.0634
2.0635 // OUTPUT OF TEXT IN GRAPHICS MODE.
2.0635 {
2.0636      va_list  argptr;
2.0636      char str[140];
2.0637      int cnt;
2.0638
```

```
2.0638      va_start(argptr,fmt);
2.0639      cnt = vsprintf(str,fmt,argptr);
2.0639      outtextxy(*xloc,*yloc,str);
2.0640      *yloc += textheight("H")+2;
2.0641      va_end(argptr);
2.0641
2.0642 return(cnt);
2.0642 }
2.0643
2.0644 //##############################################
2.0644 int Boundary_Select(int start)
2.0645
2.0646 // SELECT A VELOCITY SEGMENT FOR ANALYSIS.
2.0646 // EXAMPLE OF USAGE :
2.0647 //    srow=Boundary_Select(10);
2.0647 //    frow=Boundary_Select(srow+10);
2.0648
2.0649 {
2.0649      int i,x = DisW[0].MisC[0],xoff = start<<1,Rnum = start;
2.0650      int y1 = DisW[0].Act[1],y2 = DisW[0].Act[3];
2.0650      char ch;
2.0651
2.0652      do
2.0652      {
2.0653                for(i = 0;i < NR<<1; i += 2)
2.0653                          putpixel(xoff+x+1,y1+i,15);putpixel(xoff+x+1,y2-i,15);
2.0654                ch = getch();
2.0655
2.0655                switch(ch)
2.0656                {
2.0657                          case 0x4B:{
2.0657                                              for(i = 0;i < NR<<1;i += 2)
2.0658                                                        putpixel(xoff+x+1,y1+i,0);putpixel(xoff+x+1,y2-i,0);
2.0658                                              Rnum--;
2.0659                                              xoff -= 2;
2.0660                                    };break;
2.0660
2.0661                          case 0x4D:{
2.0661                                              for(i = 0;i < NR<<1;i += 2)
2.0662                                                        putpixel(xoff+x+1,y1+i,0);putpixel(xoff+x+1,y2-i,0);
2.0663                                              Rnum++;
2.0663                                              xoff += 2;
2.0664                                    };break;
2.0664                }
2.0665      }while (ch != 0xD);
2.0666      return(Rnum);
2.0666 }
2.0667
2.0668 //##############################################
2.0668 void LookW(void)
2.0669
2.0669 // INSPECT THE CAPTURED TIME WAVEFORM. ONE USE IS TO ENSURE
2.0670 // THAT THE AFFORE MENTIONED WAS CAPTURED WITHOUT CLIPPING.
2.0671 // THE OPTION EXIST TO STEP THROUGH (PAGE UP/DOWN) THE ARRAYS
2.0671 // AND SO VIEW ALL EXISTING DATA.
2.0672 {
2.0672 #define D DisW
2.0673      int  test = 1,r = row,i;
```

```
2.0674    char kbchar;
2.0674
2.0675    {
2.0676              if ((D[i].Mapptr = GrSave(D[i].L,D[i].T,D[i].R,D[i].B)) != NULL)
2.0677              {
2.0677                        GrWin(D[i].L,D[i].T,D[i].R,D[i].B,D[i].fcol,D[i].bcol,1);
2.0678                        XY_Prep(i);
2.0678              }else return;
2.0679    }
2.0680
2.0680    I_signal_view(Xmem[r],3,NRaw);
2.0681    I_signal_view(Ymem[r],4,NRaw);
2.0682    C_signal_view(FFreq[r],5,NR);
2.0682    C_signal_view(RFreq[r],6,NR);
2.0683
2.0683    while(!kbhit()){}
2.0684    while(test)
2.0685    {
2.0685
2.0686              if (kbhit())
2.0686              {
2.0687                        kbchar=getch();
2.0688                        switch(kbchar)
2.0688                        {
2.0689
2.0689                         case 0x49 :{
2.0690                                              (r==(MaxRow-1))?(r=0):(r++);
2.0691                                              I_signal_view(Xmem[r],3,NRaw);
2.0691                                              I_signal_view(Ymem[r],4,NRaw);
2.0692                                              C_signal_view(FFreq[r],5,NR);
2.0693                                              C_signal_view(RFreq[r],6,NR);
2.0693                                    }break;
2.0694
2.0694                         case 0x51 :{
2.0695                                              (r==0)?(r=MaxRow-1):(r--);
2.0696                                              I_signal_view(Xmem[r],3,NRaw);
2.0696                                              I_signal_view(Ymem[r],4,NRaw);
2.0697                                              C_signal_view(FFreq[r],5,NR);
2.0697                                              C_signal_view(RFreq[r],6,NR);
2.0698                                    }break;
2.0699
2.0699                      case 0x1B : test=0;
2.0700                    break;
2.0700                              }
2.0701              }
2.0702
2.0702    }
2.0703
2.0704    for(i = 3;i <= 6; i++)
2.0704              GrRestore(D[i].L,D[i].T,D[i].Mapptr);
2.0705
2.0705 #undef D DisW
2.0706 return;
2.0707 }
2.0707
2.0708 //##############################################
2.0708 void Go(void)
2.0709
```

```
2.0710 // READ DATA AS CAPTURED BY DSP56001. FOR DEBUGGING PURPOSES
2.0710 // EXAMPLES ARE SHOWN OF WHERE TO PLACE RELEVENT FUNCTIONS.
2.0711 // THE DMA-DISPLAY SEQUENCE IS REPEATED UNTIL INTERUPTED BY
2.0711 // THE USER BY EITHER THE ESC (EXIT) OR SPACEBAR (INSPECTION)
2.0712 // KEYSTROKES. NOTE THE PLACEMENT OF THE HOST2DSP FLAG WHICH
2.0713 // IS USED TO COORDINATE THE DMA TRANSFERS.
2.0713 {
2.0714     char choice;
2.0714     int  test = 1,i;
2.0715
2.0716     //ValWindow();
2.0716     //To_DSP();
2.0717
2.0718     TimerStart();
2.0718
2.0719     // SET HF0 IN ICR
2.0719     outportb(CardAddr+ICReg,inportb(CardAddr+ICReg)|8);
2.0720     while(test)
2.0721     {
2.0721                 I_DMAfromDSP(Xmem[row]);
2.0722                 I_DMAfromDSP(Ymem[row]);
2.0722                 C_DMAfromDSP(RFreq[row]);
2.0723                 C_DMAfromDSP(FFreq[row]);
2.0724                 Spectrogram(row);
2.0724
2.0725                 // I_signal_view(Xmem[row],1,NRaw);  // NOTE TIMING OF DSP INT !!
2.0725                 // I_signal_view(Ymem[row],2,NRaw);
2.0726
2.0727                 if (kbhit())
2.0727                 {
2.0728                         // RESET HF0 IN ICR
2.0729                         outportb(CardAddr+ICReg,inportb(CardAddr+ICReg)&0xF7);
2.0729                         choice=toupper(getch());
2.0730
2.0730                         switch(choice)
2.0731                         {
2.0732                                 case 0x20 : ReDraw();
2.0732                                         Bvel_flag = 1;
2.0733                                         return;
2.0733                                 case 0x1B : test = 0;
2.0734                         break;
2.0735                             }
2.0735                 }
2.0736                 row = ++row % MaxRow;
2.0736     }
2.0737     return;
2.0738 }
2.0738 //###############################################
```

```
3.0000 //*************************************************
3.0001 //              ACQUISITION AND ANALYSIS OF
3.0002 //              LOWER LIMB FLOW WAVEFORMS
3.0003 //*************************************************
3.0003
3.0004 // Copyright(C) L.SMITH
3.0005 // Department of Biomedical Engineering
3.0005 // University of Cape Town
3.0006 // Observatory
3.0006 // Cape Town
3.0007 // New South Africa
3.0008 // (021) 471250 x 235
3.0008
3.0009 #define DWNum 7                 // NUMBER OF DISPLAYWINDOWS
3.0010 #define MNum 4                  // NUMBER OF SUBMENUS CALLED FROM MAIN MENU
3.0010
3.0011 #define DMA_Chan 1
3.0011 #define Fs 40e3                 // SAMPLING FREQUENCY
3.0012 #define NR 64                   // NUMBER OF VALUES RECEIVED FROM DSP56001
3.0013 #define NRaw 128                // NUMBER OF RAW DATA VALUES RECEIVED FROM DSP56001
3.0013 #define NS 256                  // NUMBER SEND TO DSP
3.0014 #define Val_Win 200             // NUMBER OF SPECTRAL LINES PER DISPLAY WINDOW
3.0014 #define VLine_DWn 400           // NUMBER OF VERTICAL LINES PER DISPLAY WINDOW
3.0015
3.0016 #define ColLevels 16            // SELECT AS A EVEN NUMBER !!
3.0016 #define MaxColInt 63            // MAXIMIMUM COLOUR INTENSITY
3.0017 #define MinColInt 23            // MINIMUM COLOUR INTENSITY
3.0017 #define PalOff   16             // START OF GRAY SCALES
3.0018
3.0019 #include <dos.h>
3.0019 #include <ctype.h>
3.0020 #include <stdio.h>
3.0021 #include <stdlib.h>
3.0021 #include <conio.h>
3.0022 #include <graphics.h>
3.0022 #include <math.h>
3.0023 #include <mem.h>
3.0024 #include <stdarg.h>
3.0024 #include <alloc.h>
3.0025 #include <string.h>
3.0025
3.0026 //***************** STRUCTURES *****************
3.0027 struct MenuWindow{
3.0027     char      *Menulist;
3.0028     int       L,T,R,B;
3.0028     int       fcol,bcol;                // FORE- AND BACKGROUND
3.0029     int       type_op;                  // HOW OPENED
3.0030     char      *KeyList;                 // HOTKEYS
3.0030     void      (*ProcPointer[15])();     // POINTERS TO FUNCTIONS CALLED
3.0031     int       ArgList[15];
3.0032     char      far *Mapptr;              // BACKGROUND SAVE
3.0032 };
3.0033
3.0033 struct DisplayWindow{
3.0034     int       status;       // 0:CLOSED,1:ACTIVE ,2:DE-ACTIVATED
3.0035     char      title[50];
3.0035     char      xlabel[50];
3.0036     char      ylabel[50];
```

```
3.0036    int       fcol,bcol,Icol,Dcol;          // ..,PAINTCOLOR, LETTERCOLOR
3.0037    int       L,T,R,B;
3.0038    int       Act[4];
3.0038    int       MisC[4];
3.0039    char      far *Mapptr;
3.0039 };
3.0040
3.0041 //************* PROTOTYPE DECLARATIONS **********
3.0041 void GrRestore(int l,int t,char far *Bitmapptr);
3.0042 char far *GrSave(int l,int t,int r,int b);
3.0043 void InitPalette(void);
3.0043 void To_DSP();
3.0044 void ValWindow(void);
3.0044 void ScaleVal(void);
3.0045 void ReadFile(void);
3.0046 void Get_FileN(int type_op);
3.0046 void WriteFile(void);
3.0047 void Spectrogram(int row);
3.0047 void DMAtoDSP(long Ch[]);
3.0048 void I_DMAfromDSP(int *Signal);
3.0049 void C_DMAfromDSP(char *Freq);
3.0049 void DMAfromDSP(long huge *Freq);
3.0050 void DMAIn56(byte icr,word segmentpart,word offsetpart,word count);
3.0050 void InTDMA(byte icr,word DMApageno,word DMAoffset,word count,byte ICRmask);
3.0051 int InTPoll56(word Timeout);
3.0052 void DMA_Done(word Timeout);
3.0052 void C_signal_view(char *signal,int n,int nr);
3.0053 void I_signal_view(int *signal,int n,int nr);
3.0053 void signal_view(long huge *signal,int n,int nr);
3.0054 int  Boundary_Select(int start);
3.0055 void ReDraw(void);
3.0055 void WaveS(void);
3.0056 int  gprintf(int *xloc,int *yloc,char *fmt,...);
3.0057 void Go(void);
3.0057 void MeanF(void);
3.0058 void LookW(void);
3.0058 void GrWin(int l,int t,int r,int b,int fcol,int bcol,int type_op);
3.0059 void Init_Graph(void);
3.0060 void Test(void);
3.0060 void Listprint(int l,int t,int fcol,char *list,char *PossKey);
3.0061 void XY_Prep(int n);
3.0061 void Spec_Prep(int n);
3.0062 void DefaultWin(void);
3.0063 void Menu_Man(int Num);
3.0063 unsigned long int TimeElapsed(void);
3.0064 void TimerStart(void);
3.0064 void Time(void);
3.0065 void BackWin(int col);
3.0066 void Quit(void);
3.0066
3.0067 //************************************************
```

MATLAB SOURCE CODE LISTING (4.0000-4.0807)

```
4.0000 %/* ************************************************
4.0001 %/* *
4.0002 %/* *                    CW DOPPLER ULTRASOUND
4.0003 %/* *
4.0004 %/* ************************************************
4.0005
4.0006 % Waveform analysis (See respective m files for help regarding parameters)
4.0007 % NB !! 386 MATLAB required.
4.0008 %
4.0009 % Usage :
4.0010 % **        1 : load c:\bc\files\patdata\name.raw
4.0011 % **        2 : [X,Y] = lrd(name);
4.0012 %           3 : [F,R] = arspec(X,Y,15,1);
4.0013 %           4 : B = bvel(F,R,1);
4.0014 %           5 : Bfil = vfil(B,10,5,0);
4.0015 %           6 : Bseg = ss3(Bfil);
4.0016 %                   (or ..ss,..ss1,..ss1)
4.0017 %           7 : global Data
4.0018 %^          8 : var = disfit(Bseg);
4.0019 %           9 : disnel(Bseg,var,'name');
4.0020 %           10 : If so desired save parameters using "save"
4.0021 %
4.0022 % **       If so desired the periodogram data may be analyzed as an
4.0023 %          alternative. Replace 1-3 with :
4.0024 %          1 : load c:\bc\files\patdata\name.spc
4.0025 %          2 : [F,R] = lsd(name);
4.0026 %^         For Laplacian analysis use
4.0027 %                   par = lapfit(Bseg);
4.0028 %                   To calculate and view the result use
4.0029 %                   par = dislap(Bseg);
4.0030
4.0031 %************************************************
4.0032 % Calculate biased autocorrelation of a given function
4.0033 % R = acorr(fun,k,norm)
4.0034 %  k = number of correlation points to do
4.0035 %  if norm = 1 R is normalized so that R(1) = 1;
4.0036
4.0037 function [R] = acorr(fun,k,norm)
4.0038 N = max(size(fun));
4.0039 m = 1;
4.0040 for i = 1:k
4.0041     R(m) = (sum(fun(1:N-m+1).*fun(m:N)))/N;
4.0042     m = m+1;
4.0043 end
4.0044 if norm == 1
4.0045     R = R./max(R);
4.0046 end
4.0047
4.0048 %************************************************
4.0049 %  Perform AR spectral estimation on given raw data.
4.0050 %  [F,R] = arspec(xmem,ymem,nk,gr)
4.0051 %          xmem = raw data sampled into X memory
4.0052 %          ymem = raw data sampled into Y memory
4.0053 %          nk   = AR filter order
4.0054 %          gr   = 0 no graph
4.0055 %               = 1 display graph
4.0056 %  The following is done :
4.0057 %          1.        Frequency domain processing
```

```
4.0058 %              2.        Autocorrelation (Rx)
4.0059 %              3.        Extrapolation of Rx
4.0060 %              4.        Spectral calculation
4.0061
4.0062 function [F,R] = arspec(xmem,ymem,nk,gr)
4.0063 [N,M] = size(xmem);
4.0064 fs = 40e3; t = 0:N-1; fc = 5000; N_FFT = 512;
4.0065 csin = sin(2*pi*t*fc/fs)';
4.0066 ccos = cos(2*pi*t*fc/fs)';
4.0067
4.0068 for I = 1:M
4.0069     clc
4.0070     I
4.0071     xc = csin.*xmem(:,I);
4.0072     yc = ccos.*ymem(:,I);
4.0073     Ssum = xc+yc;
4.0074     Rx = acorr(Ssum,nk+1,0);
4.0075     [k,a,error,alpha] = durbin(Rx,nk);
4.0076
4.0077     m = 0;
4.0078     for n = nk+2:N_FFT/2
4.0079                Rx(n) = -sum(a(2:nk+1).*Rx(m+nk+1:-1:2+m));
4.0080                m = m+1;
4.0081     end
4.0082     Rx = [Rx(N_FFT/2:-1:2) Rx];
4.0083     Fsum = abs(fft(Rx,N_FFT))';
4.0084     R(:,I) = Fsum(((fc/fs)*N_FFT):-1:1);
4.0085     F(:,I) = Fsum(((fc/fs)*N_FFT)+1:((fc/fs)*2*N_FFT));
4.0086 end
4.0087 if (gr == 1)
4.0088     f3d(F,R,max(size(F)));
4.0089 end
4.0090
4.0091 %*************************************************
4.0092 % Calculate mean blood velocity given forward and reverse flow.
4.0093 % B = bvel(F,R,gr)
4.0094 %  F  = Forward Frequencies
4.0095 %  R  = Reverse Frequencies
4.0096 %  gr = 0 no graph
4.0097 %     = 1 display graph
4.0098
4.0099 function [B] = bvel(F,R,gr)
4.0100 clg;
4.0101 FBin = 5000/64;
4.0102 freqs = (FBin:FBin:5000)';
4.0103 for i = 1:max(size(F))
4.0104     f = sum((freqs.*F(:,i)));
4.0105     r = sum((freqs.*R(:,i)));
4.0106     B(i) = f/(sum(F(:,i))+sum(R(:,i)))-r/(sum(F(:,i))+sum(R(:,i)));
4.0107 end
4.0108 if (gr == 1)
4.0109     plot(B),grid,title('Mean Frequency'),xlabel('Frames (200 = 3sec)'),...
4.0110     ylabel('Frequency (Hz)');
4.0111 end
4.0112
4.0113 %*************************************************
4.0114 %  Fitting a function to a set of data using the Nelder-Mead
4.0115 %  simplex algorithm for minimizing a function of several
```

```
4.0116 %  variables.
4.0117 %  NB! FIRST USE : global Data
4.0118 %  var = a1,a2,a3,b1,b2 (SEE DIFFERENCE EQU)
4.0119 %  var = disfit(bvel)
4.0120
4.0121 function [var] = disfit(bvel)
4.0122 Data = [];
4.0123 Data(1,:) = 1:max(size(bvel));          % Time increments
4.0124 Data(2,:) = bvel;
4.0125 Data = Data';
4.0126 % var = a1,a2,a3,b1,b2 (SEE DIFFERENCE EQU)
4.0127 % var = [-2.8 2.7 -0.9 0.2 -0.2]';
4.0128 var = [lap2dis(bvel)]';
4.0129 var = fmins('disfun',var,1e-5);
4.0130
4.0131 %*************************************************
4.0132 %  DISFUN is used by DISFIT.  Disfun(var) returns the error
4.0133 %  between the data and the values computed by the current
4.0134 %  function of var.  DISFIT assumes the following function :
4.0135 %
4.0136 %  y(n) = -a1*y(n-1) - a2*y(n-2) - a3*y(n-3) +
4.0137 %         b1*u(n-1) + b2*u(n-2)
4.0138 %
4.0139 %  where u(n) = unit impulse function
4.0140 %  i.e.  u(1) = 1
4.0141 %            = 0 n>1
4.0142
4.0143 function f = disfun(var)
4.0144 a1 = var(1);a2 = var(2);a3 = var(3);b1 = var(4);b2 = var(5);
4.0145 t = Data(:,1);
4.0146 y = Data(:,2);
4.0147
4.0148 yp(1) = 0;
4.0149 yp(2) = b1;
4.0150 yp(3) = -a1*yp(2)+b2;
4.0151 for n = 4:max(size(y))
4.0152       yp(n) = -a1*yp(n-1)-a2*yp(n-2)-a3*yp(n-3);
4.0153 end
4.0154 yp = yp(:);
4.0155 f = sum((yp-y).^2);
4.0156
4.0157 % Statements to plot progress of fitting:
4.0158 %clg;
4.0159 %plot(t,y,t,yp)
4.0160 %xt = max(t)/4.5;
4.0161 %yt = max(y)/1.5;
4.0162 %text(xt,1.1*yt,['variable = ' num2str(var(1)) ' ' num2str(var(2)) ...
4.0163 %' ' num2str(var(3)) ' ' num2str(var(4)) ' ' num2str(var(5))]),
4.0164 %text(xt,1.0*yt,['err norm = ' num2str(f)])
4.0165
4.0166 %*************************************************
4.0167 % Curve fitting procedure to calculate model parameters of velocity
4.0168 % segment. Laplacian fitting done in the frequency domain. A display
4.0169 % of the parameters calculated is given.
4.0170 % par = dislap(segment);
4.0171
4.0172 function [par] = dislap(bseg)
4.0173
```

```
4.0174 % points = no of FFT points, fft2 is the two dim. (real+imag) fft
4.0175 echo off
4.0176 clg;
4.0177 points = 512;
4.0178 fs = 66;
4.0179 Bpoints = max(size(bseg));
4.0180 T = (1:Bpoints)/fs;
4.0181 A = 0;B = 0;G = 0;LTD = 0;
4.0182
4.0183 FREQ = fft2(bseg,1,points)/fs;
4.0184 MAG = (sqrt(real(FREQ).*real(FREQ)+imag(FREQ).*imag(FREQ)));
4.0185 MAG = MAG/MAG(1);
4.0186
4.0187 % Note that the the index of the FREQ and MAG arrays must be multiplied by;
4.0188 % 2*pi * (index=fft bin no)*fs/points to get radian freq;
4.0189 % No is the maximum  fft bin no of interest, this corresponds to
4.0190 % a frequency of 2*pi*No*fs/points;
4.0191 No = 50;
4.0192 w1 = 0:1:No-1;
4.0193 % Note that the actual frequency (radians) is 2*pi*(fft bin no)*fs/points;
4.0194 w = 2*pi*w1*fs/points;
4.0195 UNIT = ones(1:No);
4.0196
4.0197 % Perform weighted least squares fit in the frequency domain
4.0198 % No of iterations
4.0199 for ITERATION = 0:5
4.0200     if ITERATION == 0
4.0201                 Ek = ones(1:No);
4.0202     else
4.0203                 for j = 1:No
4.0204                         i = 2*pi*j*fs/points;
4.0205                         W3 = [i^3 i^2 i 1];
4.0206                         Ek(j) = 1/(P*W3')^2;
4.0207                 end;
4.0208     end;
4.0209
4.0210     L0 = UNIT*Ek';
4.0211     L2 = (Ek.*w)*w';
4.0212
4.0213     T1 = (w.*Ek)*(imag(FREQ(1:No)))';
4.0214     T3 = (w.*w.*w.*Ek)*(imag(FREQ(1:No)))';
4.0215
4.0216     S0 = (UNIT.*Ek)*(real(FREQ(1:No)))';
4.0217     S2 = (w.*w.*Ek)*(real(FREQ(1:No)))';
4.0218     S4 = (w.*w.*w.*w.*Ek)*(real(FREQ(1:No)))';
4.0219
4.0220     U2 = (w.*w.*Ek)*(real(FREQ(1:No)).*real(FREQ(1:No))+imag(FREQ(1:No)).*imag(FREQ(1:No)))';
4.0221     U4 = (w.*w.*w.*w.*Ek)*(real(FREQ(1:No)).*real(FREQ(1:No))+imag(FREQ(1:No)).*imag(FREQ(1:No)))';
4.0222     U6 = (w.*w.*w.*w.*w.*w.*Ek)*(real(FREQ(1:No)).*real(FREQ(1:No))+imag(FREQ(1:No)).*imag(FREQ(1:No)))';
4.0223
4.0224     M = [L0 0 T1 S2 -T3;0 L2 -S2 T3 S4; T1 -S2 U2 0 -U4;S2 T3 0 U4 0;T3 -S4 U4 0 -U6];
4.0225     C = [S0 T1 0 U2 0]';
4.0226     SOL = inv(M)*C;
4.0227
4.0228     P = [SOL(5) SOL(4) SOL(3) 1];
4.0229     poles = roots(P);
4.0230     Z = [SOL(2) SOL(1)];
4.0231     zeros = roots(Z);
```

```
4.0232
4.0233    Error = 0;
4.0234    for j = 0:No-1
4.0235            i = j*2*pi*fs/points;
4.0236            W1 = [i 1];
4.0237            W3 = [i^3 i^2 i 1];
4.0238            Error = Error+(MAG(j+1)-(Z*W1')/(P*W3'))^2;
4.0239    end;
4.0240    Error = sqrt(Error)/No;
4.0241
4.0242    Best = imag(poles(1));
4.0243    if Best == 0,
4.0244            Gest = -poles(1);
4.0245            Aest = -real(poles(2));
4.0246            Best = imag(poles(2));
4.0247    else
4.0248            Aest = -real(poles(1));
4.0249            Gest = -poles(3);
4.0250    end;
4.0251
4.0252 end;
4.0253
4.0254 H = (Gest*Aest*Aest+Gest*Best*Best)/((Gest-Aest)*(Gest-Aest)+Best*Best);
4.0255 FLOWest = H.*( exp(-Gest.*T) + exp(-Aest.*T).*(((Gest-Aest)/Best).*sin(Best.*T)-cos(Best.*T) ) );
4.0256 FREQest = fft2(FLOWest,1,points)/fs;
4.0257 MAGest = (sqrt(real(FREQest).*real(FREQest)+imag(FREQest).*imag(FREQest)));
4.0258
4.0259 OUT = [Aest;Best;Gest];
4.0260 LTDest = cos(atan(Best/Aest));
4.0261 LTEest = sqrt(Aest^2+Best^2);
4.0262 LTRest = Gest;
4.0263 OUT2 = [LTDest;LTEest;LTRest];
4.0264 MSE = sum((FLOWest/max(FLOWest)-bseg/max(bseg)).^2);
4.0265 par = OUT2;
4.0266
4.0267 subplot(221);
4.0268 plot(T,bseg/max(bseg),T,FLOWest/max(FLOWest)),grid;
4.0269 title('Impulse response');
4.0270 subplot(222);
4.0271 plot(w,MAG(1:No)/max(MAG),w,MAGest(1:No)/max(MAGest)),grid;
4.0272 title('Frequency transform');
4.0273
4.0274 subplot(224),title('S-Domain'),s_zpplot([[0 0 0]' poles],'c5'),grid;
4.0275
4.0276 xe = 0.1;xr = 0.2;xd = 0.3;xp = 0.4;
4.0277 text(xe,0.45,'LTE','sc');
4.0278 text(xr,0.45,'LTD','sc');
4.0279 text(xd,0.45,'LTR','sc');
4.0280 text(xp,0.45,['PI ' '(' num2str(abs((max(bseg)-min(bseg))/mean(bseg))) ')'],'sc');
4.0281
4.0282 text(0.0,0.415,'Laplace','sc');
4.0283 text(xe,0.415,num2str(LTEest),'sc');
4.0284 text(xr,0.415,num2str(LTDest),'sc');
4.0285 text(xd,0.415,num2str(LTRest),'sc');
4.0286 text(xp,0.415,num2str(abs((max(FLOWest)-min(FLOWest))/mean(FLOWest))),'sc');
4.0287
4.0288 text(0.0,0.375,'S-Dom','sc');
4.0289 text(xe,0.375,'Zeros','sc');
```

```
4.0290 text(xe,0.34,num_str(zeros(1)),'sc');
4.0291
4.0292 text(xd,0.375,'Poles','sc');
4.0293 text(xd,0.34,num_str(poles(1)),'sc');
4.0294 text(xd,0.315,num_str(poles(2)),'sc');
4.0295 text(xd,0.29,num_str(poles(3)),'sc');
4.0296
4.0297 text(0.0,0.25,'MSE','sc');
4.0298 text(xe,0.25,num_str(MSE),'sc');
4.0299
4.0300 %echo on
4.0301 % The waveform parameter estimates are:
4.0302 % A B G  (Real,imaginary for complex poles, and real pole position resp.)
4.0303 %OUT
4.0304 % LTD,LTE,LTR are
4.0305 %OUT2
4.0306 %Error
4.0307 %echo off
4.0308
4.0309 %*************************************************
4.0310 % Display variables analized.
4.0311 % varc(Bseg,var,string)
4.0312 %  var = a1,a2,a3,b1,b2 (SEE DIFFERENCE EQU)
4.0313
4.0314 function disnel(s,var,str)
4.0315 clg;fs = 66;
4.0316 u = [1 zeros(1,max(size(s)-1))];                    % system input
4.0317
4.0318 % var = a1,a2,a3,b1,b2 (SEE DIFFERENCE EQU)
4.0319 a = [1 var(1:3)'];
4.0320 b = [0 var(4:5)'];
4.0321 T = filter(b,a,u);
4.0322
4.0323 [Zz,Pz,k] = tf2zp(b,a);
4.0324 Ps = fs.*log(Pz);
4.0325 Zs = fs.*log(Zz);
4.0326 Ps = Ps(:);                    % ensure column vector
4.0327 if imag(Ps(1)) == -imag(Ps(2))
4.0328     sys_2 = poly(Ps(1:2));
4.0329     LTR = -real(Ps(3));
4.0330 else
4.0331     if imag(Ps(2)) == -imag(Ps(3))
4.0332                 sys_2 = poly(Ps(2:3));
4.0333                 LTR = -real(Ps(1));
4.0334     else
4.0335                 str = 'error';
4.0336     end
4.0337 end
4.0338 LTE = sqrt(sys_2(3));
4.0339 LTD = sys_2(2)/(2*LTE);
4.0340 if LTD > 1
4.0341     LTD = 1;
4.0342 end
4.0343 PI = abs((max(T)-min(T))/mean(T));
4.0344 MSE = disfun(var);
4.0345
4.0346 subplot(221),title('Impulse Response'),plot([s' T']),grid;
4.0347 subplot(222),title('Z-Domain'),z_zpplot([0.*Zz Pz],'c5'),grid;
```

```
4.0348 subplot(224),title('S-Domain'),s_zpplot([0.*Zs Ps],'c5'),grid;
4.0349
4.0350 xe = 0.1;xr = 0.2;xd = 0.3;xp = 0.4;
4.0351 text(xe,0.45,'LTE','sc');
4.0352 text(xr,0.45,'LTD','sc');
4.0353 text(xd,0.45,'LTR','sc');
4.0354 text(xp,0.45,['PI ' '(' ...
4.0355     num2str(abs((max(s)-min(s))/mean(s))) ')'],'sc');
4.0356
4.0357 text(0.0,0.415,'Nelder','sc');
4.0358 text(xe,0.415,num2str(LTE),'sc');
4.0359 text(xr,0.415,num2str(LTD),'sc');
4.0360 text(xd,0.415,num2str(LTR),'sc');
4.0361 text(xp,0.415,num2str(PI),'sc');
4.0362
4.0363 text(0.0,0.375,'Z-Dom','sc');
4.0364 text(xe,0.375,'Zeros','sc');
4.0365 text(xe,0.34,num_str(Zz(1)),'sc');
4.0366 text(xe,0.315,num_str(Zz(2)),'sc');
4.0367 text(xe,0.29,num_str(Zz(3)),'sc');
4.0368
4.0369 text(xd,0.375,'Poles','sc');
4.0370 text(xd,0.34,num_str(Pz(1)),'sc');
4.0371 text(xd,0.315,num_str(Pz(2)),'sc');
4.0372 text(xd,0.29,num_str(Pz(3)),'sc');
4.0373
4.0374 text(0.0,0.25,'S-Dom','sc');
4.0375 text(xe,0.25,'Zeros','sc');
4.0376 text(xe,0.215,num_str(Zs(1)),'sc');
4.0377 text(xe,0.19,num_str(Zs(2)),'sc');
4.0378 text(xe,0.165,num_str(Zs(3)),'sc');
4.0379
4.0380 text(xd,0.25,'Poles','sc');
4.0381 text(xd,0.215,num_str(Ps(1)),'sc');
4.0382 text(xd,0.19,num_str(Ps(2)),'sc');
4.0383 text(xd,0.165,num_str(Ps(3)),'sc');
4.0384
4.0385 text(0.0,0.125,'MSE','sc');
4.0386 text(xe,0.125,num_str(MSE),'sc');
4.0387
4.0388 text(0.0,0.085,'I.D.','sc');
4.0389 text(xe,0.085,str,'sc');
4.0390
4.0391 %*************************************************
4.0392 function [k,a,error,alpha] = durbin(r,nk)
4.0393 %DURBIN    Durbin's method for time-domain IIR filter design.
4.0394 %  [k,a,error,alpha] = durbin(r,nk).
4.0395 %  Finds a filter with denominator order nk, (zeros nk at
4.0396 %  origin of z-plane), given the autocorrelation row vector r.
4.0397 %  Size(r) = nk+1. The LPC filter coefficients are returned
4.0398 %  in row vector a (length nk+1), ordered in descending
4.0399 %  powers of Z, with reflection coefficients in row vector k
4.0400 %  (length nk). See references for error and alpha.
4.0401 %
4.0402 %                  1
4.0403 %  H(z)= ------------------------------
4.0404 %                        -1    -2    -3
4.0405 %       1 + a1*z + a2*z  + a3*z  ....
```

```
4.0406 %
4.0407 %  L. Smith 22-2-92        (c) Copyright 1992.
4.0408 %
4.0409 %  References:
4.0410 %  [1] Papamichalis, Panos E., Practical Approaches to
4.0411 %  Speech Coding, Englewood Cliffs, NJ: Prentice - Hall, 1987.
4.0412 %  [2] Rabiner, L.R. and R.W. Schafer, Digital Processing
4.0413 %  of Speech Signals, Englewood Cliffs, NJ: Prentice - Hall, 1978.
4.0414 %  [3] Makhoul, John, "Linear Prediction: A Tutorial Review,
4.0415 %  " Proceedings of the IEEE, Volume 63 (April,1975): pp. 561 - 580.
4.0416
4.0417 % DURBIN
4.0418      a(1) = 1.0;
4.0419      anew(1) = 1.0;
4.0420      k(1) = -r(2)/r(1);
4.0421      alpha = r(1)*(1.0-(k(1)^2));
4.0422      a(2) = k(1);
4.0423      for i = 2:nk
4.0424                  error = 0.0;
4.0425                  for j = 1:i
4.0426                              error = error+(a(j)*r(i-j+2));
4.0427                  end
4.0428                  k(i) = -(error/alpha);
4.0429                  alpha = alpha*(1.0-(k(i)^2));
4.0430                  a(i+1) = k(i);
4.0431                  for j = 2:i
4.0432                              anew(j) = a(j)+(k(i)*a(i-j+2));
4.0433                  end
4.0434                  for j = 2:i
4.0435                              a(j) = anew(j);
4.0436                  end
4.0437      end
4.0438
4.0439 % END DURBIN
4.0440
4.0441 %*************************************************
4.0442 % Mesh plot of forward and reverse frequency points.
4.0443 % f3d(F,R,n)
4.0444 %  n = number of points (200 = full 3sec)
4.0445
4.0446 function f3d(F,R,n)
4.0447 clg;
4.0448 s = [5 5 5];
4.0449 m = [-37.5 80];
4.0450 RF = [F(64:-1:1,1:n);R(:,1:n)];
4.0451 mesh(RF,m,s),title('FORWARD AND REVERSE FREQUENCIES');
4.0452
4.0453 %*************************************************
4.0454 % Converts Laplacian variables to discrete equivalents
4.0455 % to aid in the discrete curve fitting method employed.
4.0456 % var = lap2dis(bseg)
4.0457 %  var = a1,a2,a3,b1,b2 (SEE DIFFERENCE EQU)
4.0458
4.0459 function var = lap2dis(s)
4.0460 clg;fs = 66;i = sqrt(-1);
4.0461 u = [1 zeros(1,max(size(s)-1))];                % system input
4.0462 res = lapfit(s);
4.0463 clg;
```

```
4.0464 LTD = res(1); LTE = res(2); LTR = res(3);
4.0465 Ps(1) = -LTR;
4.0466 Ps(2) = -LTE*LTD - i*LTE*sin(acos(LTD));
4.0467 Ps(3) = real(Ps(2)) - i*imag(Ps(2));
4.0468 Pz = exp(Ps/fs);
4.0469 Zz = [0 0 0];
4.0470 [num,den] = zp2tf(Zz',Pz',1);
4.0471 T = filter(num,den,u);
4.0472 gain = max(s)/max(T);
4.0473 num = num*gain;
4.0474 var = [den(2:4) num(1) num(2)];
4.0475 var(:)';
4.0476
4.0477 %T = filter(num,den,u);
4.0478 %subplot(221),title('Impulse response'),plot([s' T']),grid;
4.0479 %subplot(224),title('S-Domain'),s_zpplot([[0 0 0]' Ps'],'c5'),grid;
4.0480 %subplot(222),title('Z-Domain'),z_zpplot([[0 0 0]' Pz'],'c5'),grid;
4.0481
4.0482 %*************************************************
4.0483 % Curve fitting procedure to calculate model parameters of velocity
4.0484 % segment. Laplacian fitting done in the frequency domain.
4.0485 % result = lapfit(segment to analyze);
4.0486 %  result = [LTD LTE LTR]';
4.0487 function [result] = lapfit(bseg)
4.0488
4.0489 % points = no of FFT points, fft2 is the two dim. (real+imag) fft
4.0490 echo off
4.0491 clg;
4.0492 points = 512;
4.0493 fs = 66;
4.0494 Bpoints = max(size(bseg));
4.0495 T = (1:Bpoints)/fs;
4.0496 A = 0;B = 0;G = 0;LTD = 0;
4.0497
4.0498 FREQ = fft2(bseg,1,points)/fs;
4.0499 MAG = (sqrt(real(FREQ).*real(FREQ)+imag(FREQ).*imag(FREQ)));
4.0500 MAG = MAG/MAG(1);
4.0501
4.0502 % Note that the the index of the FREQ and MAG arrays must be multiplied by;
4.0503 % 2*pi * (index=fft bin no)*fs/points to get radian freq;
4.0504 % No is the maximum  fft bin no of interest, this corresponds to
4.0505 % a frequency of 2*pi*No*fs/points;
4.0506 No = 50;
4.0507 w1 = 0:1:No-1;
4.0508 % Note that the actual frequency (radians) is 2*pi*(fft bin no)*fs/points;
4.0509 w = 2*pi*w1*fs/points;
4.0510 UNIT = ones(1:No);
4.0511
4.0512 % Perform weighted least squares fit in the frequency domain
4.0513 % No of iterations
4.0514 for ITERATION = 0:5
4.0515     if ITERATION == 0
4.0516                 Ek = ones(1:No);
4.0517     else
4.0518                 for j = 1:No
4.0519                         i = 2*pi*j*fs/points;
4.0520                         W3 = [i^3 i^2 i 1];
4.0521                         Ek(j) = 1/(P*W3')^2;
```

```
4.0522                  end;
4.0523      end;
4.0524
4.0525      L0 = UNIT*Ek';
4.0526      L2 = (Ek.*w)*w';
4.0527
4.0528      T1 = (w.*Ek)*(imag(FREQ(1:No)))';
4.0529      T3 = (w.*w.*w.*Ek)*(imag(FREQ(1:No)))';
4.0530
4.0531      S0 = (UNIT.*Ek)*(real(FREQ(1:No)))';
4.0532      S2 = (w.*w.*Ek)*(real(FREQ(1:No)))';
4.0533      S4 = (w.*w.*w.*w.*Ek)*(real(FREQ(1:No)))';
4.0534
4.0535      U2 = (w.*w.*Ek)*(real(FREQ(1:No)).*real(FREQ(1:No))+imag(FREQ(1:No)).*imag(FREQ(1:No)))';
4.0536      U4 = (w.*w.*w.*w.*Ek)*(real(FREQ(1:No)).*real(FREQ(1:No))+imag(FREQ(1:No)).*imag(FREQ(1:No)))';
4.0537      U6 = (w.*w.*w.*w.*w.*w.*Ek)*(real(FREQ(1:No)).*real(FREQ(1:No))+imag(FREQ(1:No)).*imag(FREQ(1:No)))';
4.0538
4.0539      M = [L0 0 T1 S2 -T3;0 L2 -S2 T3 S4; T1 -S2 U2 0 -U4;S2 T3 0 U4 0;T3 -S4 U4 0 -U6];
4.0540      C = [S0 T1 0 U2 0]';
4.0541      SOL = inv(M)*C;
4.0542
4.0543      P = [SOL(5) SOL(4) SOL(3) 1];
4.0544      poles = roots(P);
4.0545      Z = [SOL(2) SOL(1)];
4.0546      zeros = roots(Z);
4.0547
4.0548      Error = 0;
4.0549      for j = 0:No-1
4.0550                  i = j*2*pi*fs/points;
4.0551                  W1 = [i 1];
4.0552                  W3 = [i^3 i^2 i 1];
4.0553                  Error = Error+(MAG(j+1)-(Z*W1')/(P*W3'))^2;
4.0554      end;
4.0555      Error = sqrt(Error)/No;
4.0556
4.0557      Best = imag(poles(1));
4.0558      if Best == 0,
4.0559                  Gest = -poles(1);
4.0560                  Aest = -real(poles(2));
4.0561                  Best = imag(poles(2));
4.0562      else
4.0563                  Aest = -real(poles(1));
4.0564                  Gest = -poles(3);
4.0565      end;
4.0566
4.0567 end;
4.0568
4.0569 H = (Gest*Aest*Aest+Gest*Best*Best)/((Gest-Aest)*(Gest-Aest)+Best*Best);
4.0570 FLOWest = H.*( exp(-Gest.*T) + exp(-Aest.*T).*(((Gest-Aest)/Best).*sin(Best.*T)-cos(Best.*T) ) );
4.0571 FREQest = fft2(FLOWest,1,points)/fs;
4.0572 MAGest = (sqrt(real(FREQest).*real(FREQest)+imag(FREQest).*imag(FREQest)));
4.0573
4.0574 OUT = [Aest;Best;Gest];
4.0575 LTDest = cos(atan(Best/Aest));
4.0576 LTEest = sqrt(Aest^2+Best^2);
4.0577 LTRest = Gest;
4.0578 OUT2 = [LTDest;LTEest;LTRest];
4.0579 result = OUT2;
```

```
4.0580
4.0581 %echo on
4.0582 % The waveform parameter estimates are:
4.0583 % A B G  (Real,imaginary for complex poles, and real pole position resp.)
4.0584 %OUT
4.0585 % LTD,LTE,LTR are
4.0586 %OUT2
4.0587 %Error
4.0588 %echo off
4.0589
4.0590 %*************************************************
4.0591 % Separate "C" file into Xmem and Ymem for MATLAB (386 MATLAB NEEDED)
4.0592 % [Xmem,Ymem] = lrd(name)
4.0593
4.0594 function [Xmem,Ymem] = lrd(name)
4.0595 Xmem = name(1:200,:)';
4.0596 Ymem = name(201:400,:)';
4.0597 clear name;
4.0598
4.0599 %*************************************************
4.0600 % Separate "C" file into Ffreq and Rfreq for MATLAB (386 MATLAB NEEDED)
4.0601 % [FFreq,RFreq] = lsd(name)
4.0602
4.0603 function [FFreq,RFreq] = lsd(name)
4.0604 FFreq = name(1:200,:)';
4.0605 RFreq = name(201:400,64:-1:1)';
4.0606 clear name;
4.0607
4.0608 %*************************************************
4.0609 % Prints complex number to string.
4.0610 %  str = num_str(num)
4.0611
4.0612 function str = num_str(num)
4.0613
4.0614 str = num2str(real(num));
4.0615 if(imag(num) ~= 0)
4.0616     str_i = num2str(abs(imag(num)));
4.0617     if(sign(imag(num)) == 1)
4.0618                 str = [str '+j' str_i];
4.0619     else
4.0620                 str = [str '-j' str_i];
4.0621     end
4.0622 end
4.0623
4.0624 %*************************************************
4.0625 % To determine transfer function of an IIR filter given the impulse
4.0626 % response of the system. Prony's curve fitting is done.
4.0627 %  pro(Bseg,nb,na)
4.0628 %  Bseg = filtered velocity segment to analyze
4.0629 %  nb = numerator order
4.0630 %  na = denominator order
4.0631 %  NB!! Bseg(1) changed due to prony DIV
4.0632
4.0633 function pro(s,nb,na)
4.0634 hold off;clg;fs = 66;
4.0635 s = s + (max(s)/1000);
4.0636 u = [1 zeros(1,max(size(s)-1))];                    % system input
4.0637
```

```
4.0638 % PRONY
4.0639 % NUMBER OF POLES AND ZEROS AS GIVEN (NB! na>nb)
4.0640 xi = 1:(max(size(s)));
4.0641 [b,a] = prony(s,nb,na);
4.0642 [Zz,Pz,k] = tf2zp(b,a);
4.0643 Ps = fs.*log(Pz);
4.0644 Zs = fs.*log(Zz);
4.0645 T = filter(b,a,u);
4.0646
4.0647 if na == 3
4.0648     % PRONY
4.0649     Ps = Ps(:);              % ensure column vector
4.0650     if imag(Ps(1)) == -imag(Ps(2))
4.0651                 sys_2 = poly(Ps(1:2));
4.0652                 LTR = -real(Ps(3));
4.0653     else
4.0654                 if imag(Ps(2)) == -imag(Ps(3))
4.0655                         sys_2 = poly(Ps(2:3));
4.0656                         LTR = -real(Ps(1));
4.0657                 else
4.0658                         return;                                  % ERROR
4.0659                 end
4.0660     end
4.0661     LTE = sqrt(sys_2(3));
4.0662     LTD = sys_2(2)/(2*LTE);
4.0663     PI = (max(T)-min(T))/mean(T);
4.0664 end
4.0665
4.0666 subplot(221),title('Impulse Response'),plot([s' T']),grid;
4.0667 subplot(222),title('Z-Domain'),z_zpplot([0.*Zz Pz],'c5'),grid;
4.0668 subplot(224),title('S-Domain'),s_zpplot([0.*Zs Ps],'c5'),grid;
4.0669
4.0670 if na==3
4.0671     xe=0.1;xr=0.2;xd=0.3;xp=0.4;
4.0672     text(xe,0.45,'LTE','sc');
4.0673     text(xr,0.45,'LTD','sc');
4.0674     text(xd,0.45,'LTR','sc');
4.0675     text(xp,0.45,['PI ' '(' num2str((max(s)-min(s))/mean(s)) ')'],'sc');
4.0676
4.0677     text(0.0,0.415,'Prony','sc');
4.0678     text(xe,0.415,num2str(LTE),'sc');
4.0679     text(xr,0.415,num2str(LTD),'sc');
4.0680     text(xd,0.415,num2str(LTR),'sc');
4.0681     text(xp,0.415,num2str(PI),'sc');
4.0682
4.0683 end
4.0684
4.0685 %*************************************************
4.0686 % Select waveform segment for analysis by disfit() and normalize
4.0687 % the selected segment.
4.0688 %  S = ss3(B)
4.0689 %  B = blood velocity
4.0690
4.0691 function [S] = ss3(B)
4.0692 clg;
4.0693 B = B(:)';
4.0694 subplot(211),plot(B,'r'),title('BLOOD VELOCITY WAVEFORM'),grid,hold on
4.0695 [x,y] = ginput(1);
```

```
4.0696 xb = floor(x);
4.0697 subplot(211),plot(xb,B(xb),'xw');
4.0698 [x,y] = ginput(1);
4.0699 xe = floor(x);
4.0700 subplot(211),plot(xe,B(xe),'xw');
4.0701 x = sort([xb xe]);
4.0702 xb = x(1);xe = x(2);
4.0703 Sp = abs(xe-xb);
4.0704 S = B(xb:xb+Sp);
4.0705 S = S-S(1);
4.0706 S(max(size(S))) = 0;   % ss3 difference
4.0707 S = S./max(S);
4.0708 hold off
4.0709 subplot(212),title('SELECTED SEGMENT (NORMALIZED)'),plot(S,'g'),grid
4.0710
4.0711 %*************************************************
4.0712 function s_zpplot(zepo,col)
4.0713 %  SZPPLOT  Plots zeros and poles in S-Domain
4.0714
4.0715 [nzp,nc]=size(zepo);
4.0716 if fix(nc/2)~=nc/2
4.0717   disp('Error: ZEPO should have an even number of columns')
4.0718 return,end
4.0719
4.0720 w = 0;
4.0721 mm = max(abs([zepo(:)])); m=fix(mm+1); r=m*27/20;
4.0722 axis([-r r -m m]);
4.0723 plot(w)
4.0724 hold
4.0725 for k = 1:2:nc
4.0726   iz = find(abs(zepo(:,k))>0);
4.0727    if length(iz) > 0
4.0728         plot(real(zepo(iz,k)),imag(zepo(iz,k)),['o' col])
4.0729    end
4.0730   ip = find(abs(zepo(:,k+1))>0);
4.0731   if length(ip) > 0
4.0732        plot(real(zepo(ip,k+1)),imag(zepo(ip,k+1)),['x' col])
4.0733   end
4.0734 end
4.0735
4.0736 hold
4.0737 axis('normal')
4.0738
4.0739 %*************************************************
4.0740 % Lowpass filter the selected velocity segent using a Yulewalk IIR
4.0741 % filter. NOTE : FiltFilt is used to minimise phase distortion.
4.0742 % v = vfil(s,fcut,N,gr)
4.0743 %  s    = segment to filter
4.0744 %  fcut = cutoff frequency
4.0745 %  N    = filter order
4.0746 %  gr   = 0 no graph
4.0747 %       = 1 display graph
4.0748
4.0749
4.0750 function [v] = vfil(s,fcut,N,gr)
4.0751 fs = 66;frdiv = 10;n = 256;
4.0752 fc = round(fcut/(fs/(2*frdiv)));          %num points where gain will be unity
4.0753 f = [0:1/frdiv:1];
```

```
4.0754 H = [ones(1,fc) zeros(1,frdiv-fc+1)];
4.0755 fhz = f*fs/2;
4.0756 [Bh,Ah] = yulewalk(N,f,H);
4.0757 hh = freqz(Bh,Ah,n);      % compute complex frequency response
4.0758 hy = abs(hh);             % compute magnitude
4.0759 v = filtfilt(Bh,Ah,s);
4.0760
4.0761 ff = fs/(2*n)*(0:n-1);
4.0762 xi = 1:max(size(s));
4.0763 if (gr == 1)
4.0764     clg;
4.0765     subplot(211),plot(xi,s,'r',xi,v,'w'),title('Filter Output'),...
4.0766     ylabel('Frequency (Hz)');
4.0767     subplot(212),plot(fhz,H,'r',ff,hy,'w'),title('Frequency Response'), ..
4.0768     xlabel('Frequency (Hz)'),ylabel('Magnitude');
4.0769 end
4.0770
4.0771 %*************************************************
4.0772 function z_zpplot(zepo,col)
4.0773 %ZPPLOT  Plots zeros and poles in Z-Domain.
4.0774 %
4.0775 %  zpplot(ZEPO)   zpplot([Z,P]);
4.0776 %
4.0777 %  The zeros and poles, specified by ZEPO (See ZP for the format) are
4.0778 %  plotted, with 'o' denoting zeros and 'x' poles. Poles and zeros
4.0779 %  associated with each double column of ZEPO are plotted sequentially.
4.0780
4.0781 [nzp,nc] = size(zepo);
4.0782 if fix(nc/2) ~= nc/2
4.0783   disp('Error: ZEPO should have an even number of columns')
4.0784 return,end
4.0785
4.0786 %
4.0787 om = 2.1*pi*[1:100]/100;
4.0788 w = exp(om*sqrt(-1));
4.0789 mm = max(abs([zepo(:)])); m = fix(mm+1); r = m*27/20;
4.0790 axis([-r r -m m]);
4.0791 plot(w,'w')
4.0792 hold
4.0793 for k = 1:2:nc
4.0794   iz = find(abs(zepo(:,k))>0);
4.0795    if length(iz) > 0
4.0796        plot(real(zepo(iz,k)),imag(zepo(iz,k)),['o' col])
4.0797    end
4.0798   ip = find(abs(zepo(:,k+1))>0);
4.0799   if length(ip) > 0
4.0800       plot(real(zepo(ip,k+1)),imag(zepo(ip,k+1)),['x' col])
4.0801   end
4.0802 end
4.0803
4.0804 hold
4.0805 axis('normal')
4.0806
4.0807 %*************************************************
```

REFERENCES

Alessio, T.D., Di Giuliomaria, C., Sacco, R. and Cavallaro, A.(1983) On the Use of autoregressive time series in the modelling of Doppler signals for diagnostic purposes. Clin. Phys. Physiol. Meas., Vol. 4, No. 4, 395-401.

Atkinson, P., Woodcock, J.P., (1982) Doppler ultrasound and its use in clinical measurement. (Medical Physics Series. Academic Press). London.

Brown,J.M., Nahorski, Z.T.,Woodcock, J.P. and Morris, S.J. (1978) Transfer-function modelling of arteries. Med. & Biol. Eng. & Comput., Vol. 16, 161-164.

Bunn, A.E. and Guelke, R.W.(1983) Mechanics of the cochlea model. Acustica, 53, Vol. 5, 238-249.

Campbell, W.B., Skidmore, R. and Baird, R.N.(1984) Variability and reproducibility of arterial doppler waveforms. Ultrasound in Med. & Biol., Vol. 10, No. 5, 601-606.

Capper, W.L., Amoore, J.N., Clifford, P.C., Immelman, E.J., and Harries-Jones, E.P.(1986) A system for the rapid analysis of the femoral waveform at the bedside. Ultrasound in Med. & Biol., Vol.12, No. 1, 31-38.

Capper, W.L., Bunn, A.E. and Guelke, R.W.(1991) The estimation of tube wall compliance using acoustic input impedance. IEEE Trans. on Biomed. Eng. 38(6), 544-550.

Capper, W.L., Smith, L., Immelmann, E.J.(1994). A system for the acquisition and processing of arterial blood flow velocity waveforms. (In press).

Evans, D.H. and Schlindwein, F.S.(1989) A real-time autoregressive spectrum analyser for Doppler ultrasound signals. Ultrasound in Med. & Biol., Vol. 15, No. 3, 263-272.

Evans, D.H. and Schlindwein, F.S.(1990) Selection of the order of autoregressive models for spectral analysis of Doppler ultrasound signals. Ultrasound in Med. & Biol., Vol. 16, No. 1, 81-91.

Fitzgerald, D.E., Gosling, R.G., Woodcock, J.P.(1971) Grading dynamic capability of arterial collateral circulation. Lancet. 1, 66.

Garbini, J.L., Forster, F.K. and Jorgensen, J.E.(1982) Measurement of fluid turbulence based on pulsed ultrasound technique. Part I, II. J Fluid. Mech., 118, 445-470, 471-505.

Giordano, A.A., Frank, M.H.(1985) Least square estimation with application to digital signal processing. Wiley, New York, 40-44, 250-254.

Gosling, R.G. and King, D.H.(1974) Continuous wave ultrasound as an alternative and complement to X-rays in vascular examination. In CARDIOVASCULAR APPLICATIONS OF ULTRASOUND (Edited by R.E. Reneman), Chap. 22. North-Holland, Amsterdam.

Guelke, R.W. and Bunn, A.E.(1984) Resonance in theories of hearing. The J. of Laryngology and Otology. 98, 1177-1183.

Johnston, K.W., Kassam, M., and Cobbold, R.S.C.(1983) Relationship between doppler pulsatility index and direct femoral pressure measurement in the diagnosis of aorto-iliac occlusive disease. Ultrasound in Med. & Biol., 9, 271-281.

Johnston, K.W., Kassam, M., Koers, J., Cobbold, R.S.C. and MacHattie, D.(1984) Comparative study of four methods for quantifying doppler ultrasound waveforms from the femoral artery. Ultrasound in Med. & Biol., 10, 1-12.

Kaluzynski, K.(1989) Selection of a spectral estimation method for the assessment of velocity distribution based on the spectral distribution of ultrasonic signals. Med. & Biol. Eng. & Comput., 27, 463-469.

Kaluzynski, K.(1989b) Order selection in Doppler blood flow signal spectral analysis using autoregressive modelling. Med. & Biol. Eng. & Comput., 27, 89-92.

Law, Y.F., Graham, J.C., Cotton, L.T. and Roberts, V.C. (1984) Validity of the transfer function model of the human arterial system of the lower limb in man. Med. & Biol. Eng. & Comput., 22, 537-542.

Meara, L.A. (1984) Pole-zero extraction by non-linear regression of discrete-time arterial blood-flow waveforms. Med. & Biol. Eng. & Comput., 22, 281-284.

Mendenhall, M., Scheaffer, R.L., (1986) p.621-p.627. Mathematical Statistics with Applications. Third Edition (Duxbury Press - Boston).

Skidmore, R. (1979) The use of the transcutaneous ultrasonic flow meter in the dynamic analysis of blood flow. Ph.D. Thesis, University of Bristol.

Skidmore, R. and Woodcock, J.P. (1980) Physiological interpretation of Doppler-shift waveforms-I. Ultrasound in Med. & Biol., 6, 7-10.

Skidmore, R, Baker, J.D. and Cole, S.E.A.(1989) Laplace transform analysis of femoral artery Doppler signals : The state of the art. Ultrasound in Med. & Biol., Vol. 15,No. 1, 13-20.

Wijn, P.F.F., van der Sar, P., Gootzen, T.H.J.M., Tilmans, M.H.J., Skotnicki, S.H. (1987) Value of the spectral broadening index in continuous wave Doppler measurements. Med. & Biol. Eng. & Comput., 22, 281-284.