Environments For Real-Time Measurement And Control: A Study Of HFJV In Anaesthesia

Thesis Submitted For The Degree Of
Doctor Of Philosophy

By

Salih Kabay BSc (Sussex)
Department of Engineering
University of Leicester

May 1990

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

I would like to dedicate this thesis to my parents
as a mark of my love and respect for them.

Environments For Real-Time Measurement And Control: A Study Of HFJV In Anaesthesia

by

Salih Kabay

Statement of Originality

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy in the Department of Engineering, University of Leicester, England. All work recorded in this thesis is original unless otherwise acknowledged in the text or by references. No part of this thesis has been submitted for another degree in this or any other University.

Salih Kabay
Leicester (May 1990).

Environments For Real-Time Measurement And Control: A Study Of HFJV In Anaesthesia

Salih Kabay

Thesis Submitted For The Degree Of
Doctor Of Philosophy

Abstract

This thesis is concerned with intelligent computer-based instrumentation which can be easily adapted for measurement, modelling and control in a range of application domains. The particular application area selected for study and used to illustrate the main features of the scheme was in anaesthesia for measurement and control of high-frequency jet ventilation (HFJV). The analytical methods and experimental procedures required for this area are also applicable to many other areas throughout engineering and biomedicine. A prototype general-purpose signal processing computer (SPC) encompassing many new design concepts was built to provide a flexible and user-friendly system for performing dedicated measurement and control tasks - as specified by the application program interface.

The other objective of this study was to develop a new measurement and control environment for investigating the underlying respiratory dynamics of patient airways during HFJV - to facilitate a study of the efficacy of this mode of ventilation. Drawing on past experience with the SPC, a new computer-system was designed which overcame the bandwidth limitations of the original SPC. Based around powerful, modern and cost-effective commercial system hardware it is shown that this second-generation SPC can perform real-time measurement, modelling and control in HFJV as required.

Modifications were carried out on an existing high-frequency jet ventilator to allow new modes of respiratory excitation. The signal processing system described together with these modifications to the jet ventilator coupled with the development of a new non-invasive fibre-optic chest-wall displacement transducer forms a complete environment which permits systematic identification of respiratory dynamics with extremely high precision in a fraction of the time taken by previous workers in this field. This is achieved with only minor changes to existing jet ventilation equipment and procedures. The system is intended to cope with the volume of information that needs to be considered during HFJV and the level of complexity that this method of ventilation entails.

The measurement environment has undergone clinical trials on a small population of patients. The study clearly validated the hypothesis that the respiratory airways exhibit characteristics similar to an acoustic resonant circuit over the range of frequencies covered by HFJV. Based on this study, a Lyapunov model-reference adaptive control (MRAC) system has been designed and simulated for performing automatic control of HFJV and tested for stability and convergence over a wide range of operating conditions.

The thesis concludes with a consideration of how the presented approach can be extended to take account of new hardware and software developments.

Acknowledgements

The author of any thesis is indebted to the kind and generous efforts of his family, friends and colleagues. Many people have contributed towards this thesis and I would like to thank them all for their help.

Every PhD student is spurred on by the imagination and support of his or her supervisor. Professor Barrie Jones who has supervised this study has been a constant source of inspiration, despite his pressing commitments as Head of Department, offering valuable and constructive suggestions at all times. His contribution has not been confined to academic matters and I would like to express my sincere thanks for all of his efforts.

Particular thanks also go to Dr Arvindra Sehmi for his enthusiasm and bullying to get this thesis completed, Dr John Fothergill for his interest and contribution to the Bioengineering research group, Professor Ian Postlethwaite for the computing facilities to implement the simulation studies and the tools for producing this thesis, Philip Brown for solving software problems on the SUN-workstations, Drs Mi-Ching Tsai and Weining Feng for helping with control-related problems.

Special thanks to Dr Peter Phillipson for our discussions on Lyapunov model reference adaptive control systems and for explaining the terminology and background of the subject, Naim Dahnoun for development of the fibre-optic displacement transducer, Dr Peter Mynott and Tony Corbett for providing me with essential funding via laboratory supervisions.

Thanks must also be extended to the University Department of Anaesthesia for all their help and support, especially the Head of Department, Professor Graham Smith. I would like to thank Drs Ted Lin, Mike Jones and Ian Wilson for carrying out the clinical studies. Also, special thanks to Steve Mottram for clarifying the clinical and physiological aspects of HFJV and for his generous assistance with the project.

A very special thank you is reserved for my parents for supporting me throughout my studies.Thank you also to my younger brothers Serdar and Hayri, and my sister Nilûfer for keeping me in touch with the family spirit.

I would also like to thank Andrea, Gareth, Tony and Edwina Shepherd for their considerable kindness and friendship over the past few years.

Finally, I would like to thank all my friends and colleagues at the University of Leicester for their help and support including - Vin, John, Ashok, Lin, Andy, Naim, Gareth, Dave, Sangeet, Wang, Zainol, George, Alison, Assad and all the members of the Engineering 5-aside football team.

Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveller, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;

Then took the other.....

-ROBERT FROST, "The Road Not Taken".

Contents

Terminology

The following is a list of abbreviations used in this thesis. Mathematical equations and terminology are fully described prior to their use.

AAF	Anti-Aliasing Filter
ABG	Average Blood Gases
ACSL	Automatic Continuous Simulation Language
ADC	Analogue to Digital Converter
AI	Artificial Intelligence
API	Application Program Interface
ARDS	Adult Respiratory Distress Syndrome
BB	Black-Board
BPM	Breaths per Minute
CMR	Common Mode Rejection
COAD	Chronic Obstructive Airways Disease
COPD	Chronic Obstructive Pulmonary Disease
CPR	Cardio-Pulmonary Resuscitation
DAC	Digital to Analogue Converter
DFT	Discrete Fourier Transform
DIF	Decimation-in-Frequency
DIT	Decimation-in-Time
DPM	Dual Plane Memory
DSP	Digital Signal Processor
ECG	Electrocardiogram
EMG	Electromyogram
EPROM	Eraseable-Programmable Read-Only Memory
ES	Expert-System
FFT	Fast Fourier Transform
FIFO	First-In First-Out
GSP	Graphics System Processor
HFJV	High Frequency Jet Ventilation
HFO	High Frequency Oscillation
HFPPV	High Frequency Positive Pressure Ventilation
HFV	High Frequency Ventilation
I:E	Inspiratory-to-Expiratory ratio
I/O	Input/Output
IPPV	Intermittent Positive Pressure Ventilation
ISR	Interrupt Service Routine
ITU	Intensive Therapy Unit
LED	Light-Emitting Diode

LHA	Leicestershire Health Authority
LIFO	Last-In First-Out
P+D	Proportional-plus-Derivative
PEEP	Positive End-Expiratory Pressure
PRBS	Pseudo-Random Binary Sequence
MDAC	Multiplying Digital to Analogue Converter
MIMO	Multiple-Input Multiple-Output
MISO	Multiple-Input Single-Output
MMI	Man-Machine Interface
MRAC	Model Reference Adaptive Control
RAM	Random Access Memory
SAC	Signal Acquisition Card
SAP	Signal Acquisition Processor
SAPMON	SAP Monitor Program
SHA	Sample and Hold Amplifier
SISO	Single-Input Single-Output
SNR	Signal-to-Noise Ratio
SPC	Signal Processing Computer

Chapter 1
Introduction

1.1 Motivation and Contributions

The research work described in this thesis is concerned with intelligent computer-based instrumentation which can be easily adapted for measurement, modelling and control in a range of application domains. The particular application area selected for study and used to illustrate the main features of the scheme was in anaesthesia for measurement and control of high-frequency jet ventilation (HFJV). The analytical methods and experimental procedures required for this area are also applicable to many other areas throughout engineering and biomedicine. A prototype general-purpose signal processing computer (SPC) encompassing many new design concepts was built to provide a flexible and user-friendly system for performing dedicated measurement and control tasks - as specified by the application program interface.

Despite its success over conventional modes of ventilation modes in many clinical applications, widespread acceptance of HFJV has been relatively poor. So the other objective of this study was to develop a new measurement and control environment for investigating the underlying respiratory dynamics of patient airways during HFJV - to facilitate a study of the efficacy of this mode of ventilation. Drawing on past experience with the SPC, a new computer-system was designed around a powerful, modern and cost-effective commercial system hardware that can perform real-time measurement, modelling and control tasks in HFJV as required.

Modifications were carried out on an existing high-frequency jet ventilator to allow new modes of respiratory excitation. The signal processing system described together with these modifications to the jet ventilator coupled with the development of a new non-invasive fibre-optic chest-wall displacement transducer forms a complete environment which permits systematic identification of respiratory dynamics with extremely high precision in a fraction of the time taken by previous workers in this field.

The measurement environment has undergone clinical trials on a small population of patients. The study clearly validated the hypothesis that the respiratory airways exhibit characteristics similar to an acoustic resonant circuit over the range of frequencies covered by HFJV. Based on this study, a Lyapunov model-reference adaptive control (MRAC) system has been designed and simulated for performing automatic control of HFJV and tested for stability and convergence over a wide range of operating conditions. The practical considerations of using an adaptive ventilator are discussed and new approaches are suggested for improving and extending the system to take account of new hardware and software developments.

1.2 Organisation

This thesis is organised into eight chapters and a brief summary of their contents are given below. The appendices contain several related publications and source code for programs.

Chapter 2: Methods for Studying and Modelling Dynamic Physiological Systems

This chapter introduces the fundamental concepts essential for investigating the behaviour of band-limited dynamical systems, in particular the behaviour of the respiratory system during high frequency ventilation. The merits of signal processing in the time- and frequency-domains are examined and the practical considerations necessary for representing signals in these domains. The philosophy and application of mathematical modelling is presented to assist in interpreting data collected from experiments. An alternative experimental approach to system identification is also presented which makes no assumptions about the structural configuration of the system. This approach is central to later chapters which attempt to determine the dynamic characteristics of the respiratory system in non-parametric form, using data collected from stimulus-response experiments.

Chapter 3: A Prototype General Purpose Signal Processing Computer

This chapter describes the specification, design and development of a prototype general purpose computer system known as the signal processing computer (SPC). The main design philosophy was to encapsulate flexibility, user-friendliness and low-level machine intelligence within a single instrument for data-acquisition, signal processing and control

applications. The SPC instrument-model is one of a generic signal processing system whose application-specific duties are set by the application program interface, for use in specific measurement domains.

Chapter 4: Mechanical and Physical Characteristics of the Respiratory System

This chapter is mainly a review of respiratory anatomy and physiology. Particular attention is given to the morphometry of the tracheobronchial tree because of its importance to respiratory function. Consideration is given to modelling the physiological aspects of HFJV and its effects on pulmonary mechanics and the mechanisms by which gas transport is maintained. The importance of pulmonary function testing based on respiratory impedance measurements is discussed and a new method of impedance measurement proposed.

Chapter 5: Instrumentation for Measurement and Control in High Frequency Jet Ventilation

The instrumentation described in this chapter is designed to provide real-time analysis of multi-channel respiratory data within the clinical environment. The system specifications enable respiratory dynamics testing of patients with only minimal changes to existing jet ventilation procedures. A framework for stimulus-response experiments has been designed to validate the hypothesis that acoustic resonance of the respiratory airways represents an optimal state for alveolar gas exchange. Real-time signal processing algorithms have been implemented around a user-friendly system to provide rapid, high-precision identification of patient respiratory dynamics.

Chapter 6: Clinical Patient Data

This chapter presents the results of clinical trials performed on real-patients using the instrumentation described in Chapter 5.

Chapter 7: Design and Simulation of a Second Order MRAC System for Artificial Ventilation

Since artificial ventilation deviates considerably from the normal physiological mechanism of respiration certain adverse effects can result when the method is used improperly. This chapter considers the physiological factors affecting gas exchange during artificial ventilation and puts forward criteria for selecting control variables for implementing automatically controlled ventilation. Also considered is a preliminary design of an automatically controlled ventilator device. A second-order model reference adaptive control (MRAC) scheme has been developed for this purpose, and a simulation study carried out to determine its dynamic performance under varying conditions.

Chapter 8: Discussion and Conclusions

This chapter contains concluding remarks and suggestions for future work.

Chapter 2
Methods for Studying and Modelling Dynamic Physiological Systems

2.1 Introduction

The dynamics of linear systems can be characterised by their frequency response, which is limited to a range of frequencies associated with the system's bandwidth. The system response rapidly attenuates for excitation frequencies outside of this bandwidth. The main objectives of this study are to investigate the characteristics of physiological systems, in particular the dynamic behaviour of the respiratory system during high frequency ventilation. This chapter aims to describe the concepts used to achieve an understanding of this behaviour.

The analytical methods used to identify the dynamics of a system are very important and must be selected according to certain guidelines. For example, the frequency response of a low noise, linear system with a single input stimulus, such as a step or ramp, can be easily calculated from deterministic transient analysis of the time-domain response. However, a similar approach would be less applicable to a physiological system which is relatively complex, noisy and often exhibiting non-linear and time-variant behaviour. The case for using randomly varying stimuli for characterising such systems is considered below.

The merits of signal processing in the time- and frequency-domains are examined, in particular the practical considerations necessary for representing signals in these domains. The philosophy and application of mathematical modelling is presented to assist in interpreting data collected from experiments. In this approach, a structural configuration for the system is assumed in the form of differential equations with unknown parameters.

The identification task is then a search in parameter space in an attempt to optimise some objective function. An alternative experimental approach to system identification makes no assumptions about the structural configuration of the system. This approach attempts to determine the dynamic characteristics of the system in non-parametric form, using data collected from stimulus-response experiments.

2.2 Discrete Signal Processing

In reality most signals vary continuously with time. To proceed with computer-based analysis of signals requires that the input waveform be sampled at regular intervals. This is done by passing the continuous signal through a sample and hold circuit (Sec. 3.7) which is used to lock on to the amplitude level corresponding to a given sampling instant. This level is then made available to the analogue-to-digital converter for digitisation. Digital signal processing algorithms can then be used to efficiently calculate the input spectrum, the output spectrum and the cross spectrum of data collected in this way.

The sampling interval Δt determines the maximum frequency f_{max} which can be analysed correctly, where $f_{max}= \frac{1}{2\Delta t}$ is known as the Nyquist frequency. Any frequency components in the applied waveform which are greater than f_{max} will produce *aliasing*, where frequency components above the Nyquist frequency are translated to a lower frequency component in the sampled waveform. This results in power at different frequencies being combined during analysis, leading to erroneous results (Bendat and Piersol, 1971). Thus low-pass analogue filters, also known as anti-aliasing filters (Sec. 3.8), must be used to sharply attenuate any frequency components greater than f_{max}. Inevitably, the gain characteristic of such filters exhibit a gradual roll-off for frequencies greater than the cut-off. Hence, some frequencies above f_{max} will feed through to produce aliasing. This effect can be further reduced by increasing the order of the filter, and hence the sharpness of the filter roll-off. However, care should be taken to compensate for the phase delay imposed on the signal as it passes through the filter. In particular, where several input channels are being filtered, the gain and phase characteristics of each anti-alias filter must be tightly matched across every channel (Sec. 3.8) to avoid introducing any bias errors.

If we assume the total number of data points is fixed, the sampling interval must then be chosen by a trade-off. If Δt is very large, the data will contain very little information about the high frequency dynamics. If Δt is very small, the disturbances may have a large influence and information on the low-frequency dynamics may be lost. The general criterion is to set the sampling interval to 10% of the settling time of a step response (Soderstrom and Stioca, 1989).

2.3 Band-Limited White Noise Stimulus

The input signal used for exciting a system has a significant influence on the results obtained. In some situations the choice of input is imposed by the type of identification method employed. For instance, transient analysis requires typically a step or an impulse as input. In other cases, the input may be chosen on the basis of the experimental conditions and the types of signal that the system can accommodate.

The classical method of frequency response testing of systems involves application of a number of sinusoidal stimuli, one after the other, and measuring the system's response over a range of frequencies. This method can be tedious, and cannot be applied to a system which adapts or shows rapid fatigue to the applied stimuli. An alternative and more efficient approach is to apply randomly varying stimuli because they contain a wide range of frequency components and are equivalent to applying a range of sinusoids simultaneously. Also, standard methods can be used to determine the frequency response and linearity of the system to each frequency component. However, a decision has to be made as to the type of random stimuli to use and how best to apply them.

White-noise excitation is particularly useful in system identification experiments where the duration of the experiment must be kept to a minimum. This type of input ensures that all modes of the system are being stimulated during identification, allowing for maximum information about the system to be gathered in the shortest possible time.

The power spectrum of white noise is flat, which means that it contains equal amounts of power over all equal frequency bands. In all practical applications the *white-noise* must be band-limited to cover the frequency range of interest, usually achieved by low-pass filtering at a cut-off set to the maximum frequency of interest.

The amplitude of the input stimulus is also significant in system identification. Safety precautions may impose constraints on how much variation in signal amplitude can be tolerated. Many systems exhibit nonlinear characteristics but can be piece-wise linearised over a range of amplitudes. In this case, the input amplitude will affect the system dynamics and the results obtained. Conversely, restricting the input amplitude to ensure linear behaviour may conflict with the requirement that a large input amplitude may be required to increase the accuracy of measurements.

2.4 Waveform Generation

Several methods exist for generating stimuli for frequency response testing of systems. The classic approach is to apply a unit impulse with a flat frequency spectrum. In practice, the impulse will have a finite width and is band-limited by low-pass filtering. The system under study is often stimulated with a number of such pulses applied at regular time

intervals, the output power due to each pulse contributing to an average estimate of the power spectrum. A flat frequency spectrum can also be realised by applying pulses at random time intervals, according to a Poisson point process (Bendat and Piersol, 1971).

A pseudo-random binary sequences (PRBS) provides another method of generating a signal with a band-limited white-noise spectrum. The PRBS can be easily produced using standard digital circuits which generates a randomly switching binary pulse train. A single pulse has a power spectrum which decreases with the square of the frequency. In the case of a PRBS, the randomly switching waveform does not remain in any state for very long, this tends to attenuate low frequency components to the degree required to achieve a flat spectrum.

The availability of specialised digital signal processors means that computers can now be used to simultaneously stimulate and measure the evoked responses from a system. Template waveforms can be pre-generated and stored in data files for replay via a digital-to-analogue converter (DAC). This provides the flexibility to playback data at faster or slower speeds to obtain waveforms with different frequency characteristics. If the waveform to be generated is symmetrical, substantial memory reduction can be achieved by repeatedly cycling through a single period of the waveform. Digitally generated waveforms provide for much greater accuracy, stability and flexibility than analogue methods. Also, the input to the system can be adjusted on the basis of some control law, allowing the computer to behave like an intelligent feedback controller (Chapter 7).

2.5 Time-Domain Analysis

Time-domain analysis makes use of processing methods that operate directly on the raw signal. This is in contrast to frequency-domain methods (Sec. 2.6) which involve some form of spectral representation. Typical time-domain representations of signals include average zero-crossing rate (Fusefeld, 1978), turning-points (Lago and Jones, 1983), energy (Hayward, 1978), and autocorrelation. These methods have the advantage that the signal processing is relatively simple to implement and the results provide a useful basis for estimating important features of the signal.

For example, a zero-crossing is said to occur if successive samples have different algebraic signs and the rate of zero-crossings provides a crude measure of the frequency content of a signal. Since high frequencies imply high zero-crossing rates, and low frequencies imply low zero-crossing rates, there is a strong correlation between zero-crossing rate and energy distribution with frequency. The application of this type of analysis can be widely found in electromyography.

More complex methods of time-domain analysis can also be used. For example, Sehmi (1988) developed a knowledge-based approach to perform event analysis of significant waveforms from an ensemble of time-locked sensory and cognitive evoked-responses. More recently, Loudon et al. (1990) have incorporated classification theory into a similar scheme, applied to the decomposition of myoelectric motor unit action potentials. These schemes are computation intensive and are limited to computer systems which can efficiently support high-level languages such as C and Prolog.

There are a number of practical considerations in implementing a representation based on time-domain methods of analysis. Although the basic algorithm for computing, say, a zero crossing requires only a comparison of signs of pairs of successive samples, special care must be taken in the sampling process. Clearly, the zero-crossing is affected by many factors including the dc offset in the analogue-to-digital converter (ADC), 50 Hz mains noise in the signal, and any noise that may be present in the digitising system. Therefore, the analogue processing stages preceding digitisation play a critical role in minimising these effects.

The signal processing computer (SPC) described in Chapter 3 tackles the practical issues of implementing a general-purpose signal processing system. The SPC uses automatic gain control circuitry to calibrate input signals to minimise the effects of quantisation errors (Sec. 3.7), auto-tracking anti-alias filters whose cut-off frequency is set according to the sampling frequency to reject high-frequency artefact (Sec. 3.8), and coherent time-domain averaging for enhancing the signal-to-noise ratio of sampled signals (Sec. 3.13). The coherent averaging routine is another example of a non-trivial time-domain method which involves complex triggering and event-timing algorithms.

2.6 Frequency-Domain Analysis

The frequency-domain representation of a signal can be obtained by applying Fourier analysis (Kabay, 1990). The basic property of the Fourier transform is its ability to decompose a waveform into its constituent frequency components. For instance, a sum of sine waves overlapping in time, transforms into a weighted sum of impulses which, by definition, are non-overlapping.

With the developments in an efficient class of algorithms known as the fast Fourier transform (FFT) by Cooley and Tukey (1965), and the availability of digital signal processor (DSP) devices, has meant that many workers can now perform frequency-domain analysis of a signal in real-time on a relatively inexpensive computer system. In fact, the speed advantage of FFT-based spectral analysis is such that, in many cases, it is more efficient to implement a time-domain calculation by transforming the analysis into the frequency-domain, and inverse transforming the result back into the time-domain.

2.6.1 The Fast Fourier Transform

The FFT is based on the concept of sub-dividing a large computational problem into a large number of sub-problems which can be solved more easily (Danielson and Lanczos, 1942). The FFT results in a dramatic computational saving over direct calculation of the discrete Fourier transform (DFT) for increasing sample lengths. The radix-2 in-place FFT algorithms are the most popular since they make optimal usage of memory and are relatively easy to implement. Two types of in-place algorithm exist:

- the *decimation in time* (DIT) method where the transforms of shorter sequences, each composed of every *rth* sample, are computed and then combined into one big transform, and
- the *decimation in frequency* (DIF) method where short pieces of the sequence are combined in *r*-ways to form *r* short sequences, whose separate transforms taken together constitute the complete transform.

In the DIT algorithm, the first stage of the DFT is expressed as the sum of two half-length DFTs composed of even and odd samples of the original sequence. This method produces output (frequency) samples that are in correct sequence. The DIF algorithm calculates two half-length DFTs composed of the first and second halves of the original sequence. In this case, the output is composed of the even and odd samples respectively. Thus, DIT refers to grouping the input sequence into even and odd samples, whereas DIF refers to grouping the output sequence into even and odd samples.

The DFT calculation makes repeated use of a fundamental calculation known as the *butterfly*. The butterfly execution speed can be used to benchmark the performance of the FFT algorithm. For in-place calculation of the FFT, the data sequence must be scrambled to correct for the repeated movement of numbered members of the input data to the end of the sequence. In the DIT algorithm the input is bit-reversed and the output is in correct order. The reverse is true for the DIF algorithm.

The main considerations in implementing Fourier analysis of signals using the FFT are described elsewhere (Kabay, 1990). Chapter 5 considers the time-critical aspects of FFT-based spectral analysis (see also Kabay and Jones, 1990).

2.6.2 Power Spectrum

The power spectrum of a signal of infinite duration is defined as the Fourier transform of its autocorrelation function (Oppenheim and Schafer, 1989). We cannot estimate the power spectrum of a signal by directly applying the Fourier transform since the Fourier integral is non-convergent. This problem does not arise in practice due to the limited time and resources available for experimentation. Hence, the power spectrum of a signal, with finite record-length, can be estimated directly from its Fourier transform:

$$\Phi_{xx}(\omega) = X(\omega)X^*(\omega) = |X(\omega)|^2 \qquad (2.1)$$

where $X(\omega)$ is the Fourier transform of $x(t)$ and $X^*(\omega)$ is the complex conjugate of $X(\omega)$.

As seen from equation (2.1), the power spectrum of a signal, calculated from the squared magnitude of its complex Fourier transform, provides a measure of the power distribution of the signal with frequency. It does not carry any phase information about the harmonic components of the signal. Hence, for large data sequences, the accuracy of this approach can be improved by dividing the signal into shorter segments and averaging the associated estimates.

2.6.3 Transfer Function Measurements

This section will derive a method of computing the transfer function of single-input single-output (SISO) linear system, where both the magnitude and phase spectra can be easily obtained over a wide range of frequencies. This method has the advantage of not making any assumptions on the character of the input signal, i.e. the analysis is valid for all stimuli.

The Fourier transform S_x of a time-varying signal $x(t)$ can be expressed as a complex number of the form:

$$S_x = X_r + jX_i \qquad (2.2)$$

where $j = \sqrt{-1}$, X_r and X_i are the real and imaginary parts of S_x at frequency $\omega = 2\pi f$, and contain magnitude and phase information of this frequency component of $x(t)$.

Consider a single-input $x(t)$, single-output $y(t)$ linear system $h(t)$, the transfer function $H(j\omega)$ is:

$$H(j\omega) = \frac{S_y}{S_x} \qquad (2.3)$$

If $x(t)$ is random white noise, $H(j\omega)$ can be found at all frequencies in $x(t)$. However, due to synchronisation problems and statistical considerations related to the randomness of the input $x(t)$, the computation of $H(j\omega)$ by this method is not practical (Richards, 1967). These problems can be overcome by using the concept of power spectra.

The input power spectrum is:

$$G_{xx} = S_x S_x^* = X_r^2 + X_i^2 \qquad (2.4)$$

where S_x^* is the complex conjugate of S_x. The term G_{xx} no longer contains phase information. Similarly,

$$G_{yy} = S_y S_y^* = Y_r^2 + Y_i^2 \tag{2.5}$$

The cross-power spectrum is defined as:

$$G_{yx} = S_y S_x^* = Z_r + jZ_i \tag{2.6}$$

where $Z_r = (X_r Y_r + X_i Y_i)$ and $Z_i = (X_r Y_i - X_i Y_r)$ which preserves the phase relationship between input and output and can be used to compute the transfer function from:

$$H(j\omega) = \frac{G_{yx}}{G_{xx}} \tag{2.7}$$

and modulus,
$$|H(j\omega)| = \frac{|G_{yx}|}{G_{xx}} = \frac{\sqrt{Z_r^2 + Z_i^2}}{X_r^2 + X_i^2} \tag{2.8}$$

and phase,
$$\angle H(j\omega) = \tan^{-1}\left(\frac{Z_i}{Z_r}\right) \tag{2.9}$$

Since the cross-spectrum is a complex series with real and imaginary components, it contains both gain and phase information about the system. The system transfer function can then be estimated by dividing the cross-spectrum by the input spectrum.

Together with the transfer function, we can also calculate the coherence as a function of frequency. The coherence is a normalised measure which varies between zero (if the output is not linearly correlated to the input) and unity (if there is a perfect linear correlation between input and output). Non-linearities, or the presence of components in the output unrelated to the input, will reduce the coherence. The coherence function γ^2 is given by:

$$\gamma^2(j\omega) = \frac{|G_{yx}|^2}{G_{xx} G_{yy}} \qquad 0 \le \gamma^2 \le 1 \tag{2.10}$$

The coherence γ^2 is a real term which, for any frequency ω gives the fraction of power at the response that is due to the input. Whenever γ^2 is less than unity, either the system is non-linear or the signals do not have a causal relationship, or both. The deviation of γ^2

from unity is a quantitative measure of these conditions. Since physiological systems are both nonlinear and have high noise content, the coherence function should provide a valuable measure of confidence in spectral estimates.

This method of transfer function analysis overcomes many problems, especially the synchronisation problems associated with signal averaging. The time-domain signal averaging scheme mentioned in Sec. 2.5 makes use of complex triggering algorithms to synchronise channels before performing coherent averaging. Using cross-spectral methods, the amplitude- and phase-spectra are automatically correlated. Hence, averaging may be performed by simply summing the individual spectra without worrying about synchronisation.

2.6.4 Filtering of Signals

Signal averaging is the most common type of filtering to be applied during measurements. In particular, the large variability and noise content of physiological signals often necessitates signal averaging before meaningful results can be obtained. Signal averaging is most effective where the noise component is additive and has zero mean; the signal-to-noise ratio will improve with $\sqrt{N}$ where N is the number of averages.

Filtering can also be used for removal of baseline drift or slowly-varying nonstationarities, which is common in physiological measurements. Typically, this may arise when a strong low-frequency periodic source (e.g. the heart) contributes components to response recording (e.g. respiratory pressure and flow measurements). This type of noise can be removed either by using trend removal based on a least-squares-error polynomial method (Marmarelis and Marmarelis, 1978) or by using high-pass filtering.

2.6.5 Windowing

The major limitation of FFT-based spectral analysis is the associated spectral sampling. Since the spectrum is computed for a discrete set of sample frequencies, important features in the spectrum may not be evident in the sampled spectrum. This error can be easily reduced by improving the spectral resolution δf, which is inversely proportional to the FFT sample length. However, the data windowing which must be applied to the signal in advance of FFT-based analysis, means that the spectral resolution is limited by the window length. A more persistent source of error caused by windowing a signal is the sharp discontinuities that are produced in the time-domain. This results in the convolution of the true spectrum with a sinc function with characteristic side-lobes in the frequency domain (Oppenheim and Schafer, p702, 1989). The side-lobes are responsible for the distortion which tends to smear energy across spectral components. Although the accuracy of measurements can be improved by selecting appropriately tapering window functions, the frequency resolution will always be compromised.

2.7 Mathematical Modelling

The previous sections have introduced the practical and theoretical basis for capturing and processing real data. The main objectives of these measurements and analysis is to gain a deeper understanding about the system under investigation. The methodology adopted in this thesis uses nonparametric modelling of observations based on time- and frequency-domain signal processing algorithms. Also, mathematical modelling using physical laws to construct linear and nonlinear differential equations of the system is used for simulating the performance of the system under controlled conditions.

Mathematical modelling based solely on physical insight is likely to be incomplete due to the complexity of real systems. Some of the disadvantages of this approach are:

- the model is only valid under certain operating conditions;
- the model gives little physical insight, since the parameters have no direct physical meaning and should only be used as tools that describe the system's overall behaviour;
- the mathematical models are relatively easy to construct and use.

2.7.1 Data and System Models

In general, models can be classified as being either a *data model* or *system model.* A data model is phenomenologically nonspecific providing a purely conceptual tool whose main function is to present experimental results in a way that makes comparisons possible. The model is arbitrary since it has no structural implications and makes no functional statements regarding the system from which data are derived. Often, workers introduce a simple physical representation of the model, lumping spatial and temporal characteristics with basic mechanisms.

A system model is physically or structurally specific, attempting to approximate some physical reality, incorporating the laws of physics, with the aim of comparing simulated with experimental results. Despite the oversimplification involved in such an approach, deviations of the model from reality may be examined and the assumptions that are incorporated in the model tested for their influence on the simulated results. Care must be taken in assessing the adequacy of the model and its assumptions need constant critical evaluation. The temptation to assume that the model simulates the physical system when predicted results agree with the real situation must be resisted. It should be borne in mind that there is always the possibility that other models would yield an equally strong correlation.

2.7.2 Classification of Models

Mathematical models of dynamic systems can be classified in various ways which describe how the effect of an input signal will influence system behaviour at subsequent times. The input-output structure of a model is either single-input/single-output (SISO), multiple-input/single-output (MISO) or multiple-input/multiple-output (MIMO). The simplest and most commonly used model is the SISO where a description is given of the influence of one input on one output. The MISO and MIMO models give a more realistic representation of physiological systems, however they are more difficult to analyse.

A model is described as being linear if the output depends linearly on the input stimulus; otherwise it is nonlinear. Where a system has parameters that change with time, auto-tracking or real-time identification for estimating the model parameters may be necessary (Chapters 5 and 7).

To accurately model a real system, with a distributed parameter structure, would require an infinite number of equations or the use of partial differential equations. The most common approach, however, is to use lumped parameter models based on a finite number of differential equations to achieve an approximate model of the real system. The majority of models are deterministic, where the response of the model can be exactly calculated given the input stimulus. Stochastic theory can be used to incorporate uncertainty into the model to take into account the effects of unknown disturbances which make an exact solution impossible.

2.8 System Identification

System identification is an experimental approach, where experiments are performed on the system and a model fitted to the recorded data through the numerical exercise of parameter estimation. This approach suffers from some problems:

- an appropriate model structure must be found, which can be a non-trivial task especially if the system dynamics are nonlinear;
- if the observed data is contaminated by noise, then the model complexity will be erroneously increased to account for the noise;
- where a time-invariant model is being *fitted* to a system with time-varying parameters the models may not converge;
- some signals or state variables, important in defining the model, are unobservable.

In general, an identification experiment is performed by exciting the system with an appropriate input signal (Secs. 2.3-2.4) and observing its stimulus/response over a finite time interval. Many workers then attempt to fit a parametric model of the process to the recorded input and output sequences. Once an appropriate model has been selected, some

statistical method is used to estimate the unknown parameters of the model. An iterative cycle of events then follow, where a tentative structure is chosen and the corresponding parameters estimated. The model obtained is then tested to see if it is representative of the system. If this fails, a more complicated model structure must be considered, its parameters estimated, the new model validated, and so on.

Chapters 5, 6 and 7 lay the foundations for a recursive or on-line identifications scheme. This approach is well suited to the use of adaptive control in regulating the ventilatory requirements of a patient based on the most recent estimate of the patient model parameters. The instrumentation of Chapter 5 can be used to perform this recursive estimation scheme in real-time. The time-varying characteristics of a *real* patient can then be incorporated into a control scheme (Chapter 7) that monitors the current status of a patient model.

2.9 References

Bendat, J.S. and Piersol, A.G. (1971): "*Random data: analysis and measurement procedures*", Wiley, New York.

Cooley, J.W. and Tukey, J.W. (1965): "An algorithm for the machine computation of complex Fourier series", *Math. Comput.*, **19**: 297-301.

Danielson, G.C. and Lanczos, C. (1942): "Some improvements in practical Fourier analysis, and their application to x-ray scattering from liquids", *J. Franklin Inst.*, **233**: 365-380.

Fusefeld, R.D. (1978): "Instrument for quantitative analysis of the elctromyogram", *Med. & Biol. Eng. & Comput.*, **16**, 290-295.

Goodwin, G.C. and Payne, R.L. (1977): "*Dynamic system identification: experiment design and data analysis*", Academic Press, New York.

Hayward, M. (1977): "Automatic analysis of electromyograms in healthy subjects of different ages", *J. Neurol. Sci.*, **33**, 397-413.

Kabay, S. (1990): "The FFT and other methods of discrete Fourier analysis", In: *Digital Signal Processing: Principles, devices and applications*, N.B. Jones and J.D. McKWatson (Eds.), IEE Control Engineering Series **42**, 65-77.

Kabay, S. (1990): "A case study of a DSP chip for on-line monitoring and control in anaesthesia", In: *Digital Signal Processing: Principles, devices and applications*, N.B. Jones and J.D. McKWatson (Eds.), IEE Control Engineering Series **42**, 313-327.

Lago, P.J.A. and Jones, N.B. (1983): "Turning points spectral analysis of the interference myoelectric activity", *Med. & Biol. Eng. & Comput.*, **21**, 333-342.

Ljung, L. (1987): "*System identification: theory for the user*", Prentice Hall, New Jersey.

Loudon, G.H., Sehmi, A.S. and Jones, N.B. (1990): "Intelligent classification in EMG decomposition", *Proc. Comadem '90*, (submitted for publication).

Marmarelis, P.Z. and Marmarelis, V.Z. (1978): "*Analysis of physiological systems: the white noise approach*", Plenum Press, New York.

Richards, P.I. (1967): "Computing reliable power spectra", *IEEE Spectrum*, **4**, 83-90.

Sehmi, A.S. (1988): "*New environments for neurophysiological investigations*", PhD Thesis, University of Leicester.

Soderstrom, T. and Stoica, P. (1989): "*System identification*", Prentice Hall, New Jersey.

Chapter 3
A Prototype General Purpose Signal Processing Computer

3.1 Introduction

This chapter describes the design and implementation of a prototype general purpose computer system known as the *signal processing computer* (SPC). The main design philosophy was to encapsulate flexibility, user-friendliness and low-level machine intelligence within a single instrument for data-acquisition, signal processing and control applications. The SPC specification was formulated on the basis that it would be used in a time-critical environment, such as a hospital intensive therapy unit or operating theatre, where the user is unable to devote much attention to instrument details. The SPC architecture can be user-configured to support many different applications. An application program is created by interfacing with the generic SPC kernel to specify the required system configuration. Based on this approach, the *instrument-model* is one of a generic signal processing system whose application-specific duties are set by the application program interface.

The last few decades has seen considerable effort been devoted to the design of low-cost, high-performance microprocessor devices. The widespread availability of such technology has meant that a previously unattainable level of complexity and control could now be incorporated into instrumentation. The greater flexibility and extendibility offered by microprocessor-based systems has been utilised to produce a new class of *intelligent* instrumentation. Features such as greater fault-tolerance, diagnostic self-testing, inter-communication with peripheral devices, etc., are now standard specifications.

More powerful and costly instruments offer *building-blocks* instead of *solutions*, where the user is responsible for combining these blocks according to a specific measurement problem. Whilst this offers considerable flexibility, it also assumes the user to have sufficient expertise to use the building blocks properly. This approach would be most valuable in a research environment where users have knowledge about the implementation technique of the instrument and also of the application domain. However, it would be tempting disaster to use this concept in, say, a clinical environment where most users are non-professionals with minimal expertise in this domain.

The SPC instrument design is based on a generic system which is extendible, via the application program interface (API), for use in specific measurement domains. The power and flexibility of this system architecture enables researchers and non-experts alike to manipulate the instrument to its full potential; the researcher has the option to develop a customised API to suit his particular requirements, whilst the novice user can be guided through an application by a user-friendly interface. The man-machine interface supports learning with context-sensitive help for the novice user, without being verbose with the more proficient user. The following sections describe the SPC in detail.

3.2 SPC Instrument Development

A formal specification for the SPC instrument was initially published by Hailstone et al. (1986, Appendix 1). The philosophical objectives of the SPC have already been discussed in the previous section. The SPC concept originated at Sussex University[1] and has gradually evolved through the intellectual input of many people. This section gives a brief summary of the contribution of other workers and identifies the area of development contributed by the author. The list of contributors are given in the paper included in Appendix 1.

The SPC generic kernel came about from the efforts of Hailstone and Sehmi. Kabay made significant modifications to the SPC kernel to support the real-time extensions that were incorporated into the system. The first application to be implemented on the SPC was a simulation study of electromyographic spectral-analysis based solely around the SPC kernel (Sehmi, 1988). Parekh (1986) assisted in this study with the development of the turning points algorithm (Lago and Jones, 1983) and in generating simulated electromyographic data. The graphics libraries were jointly designed and written by Hailstone, Sehmi and Kabay. Whilst the dual plane memory subsystem was the conception and development of McKWatson. The project was supervised by Jones.

1. Division of Biomedical Engineering, University of Sussex, Falmer, Brighton, East Sussex.

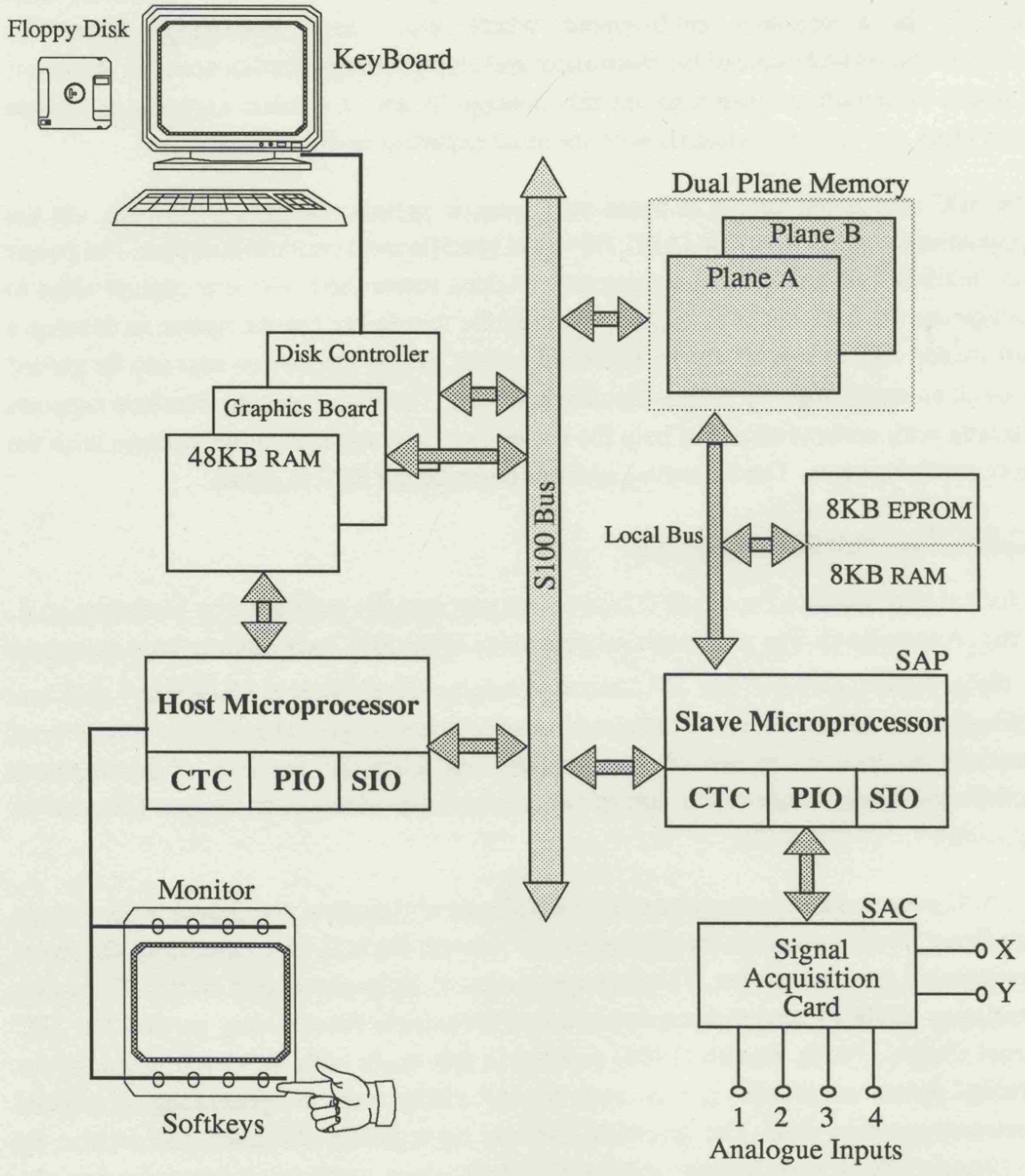


Fig. 3.1 The signal processing computer (SPC) architecture. The schematic shows the SPC system layout and the communication paths that exist between modules.

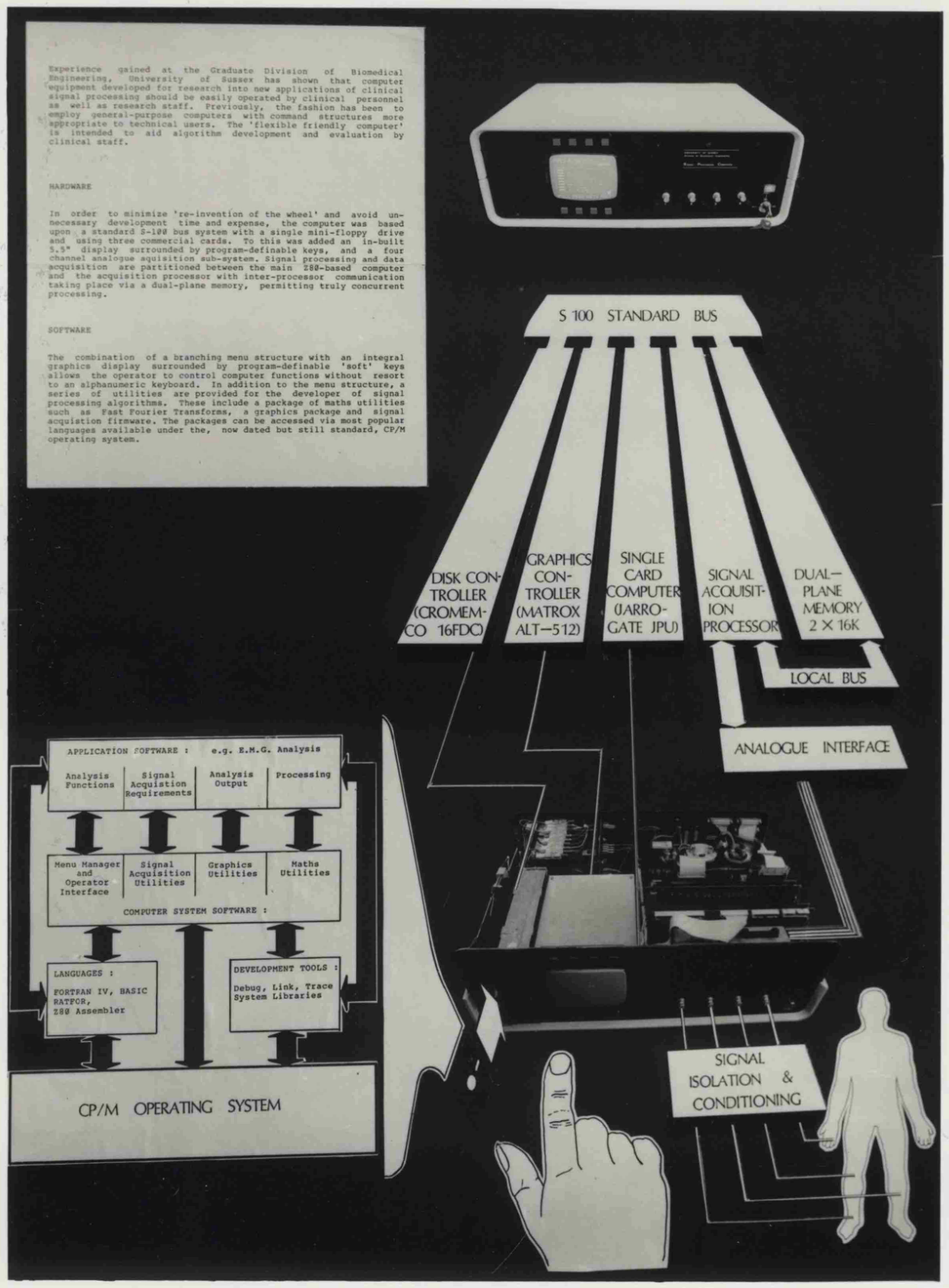


Plate 3.1 The Signal Processing Computer system overview.

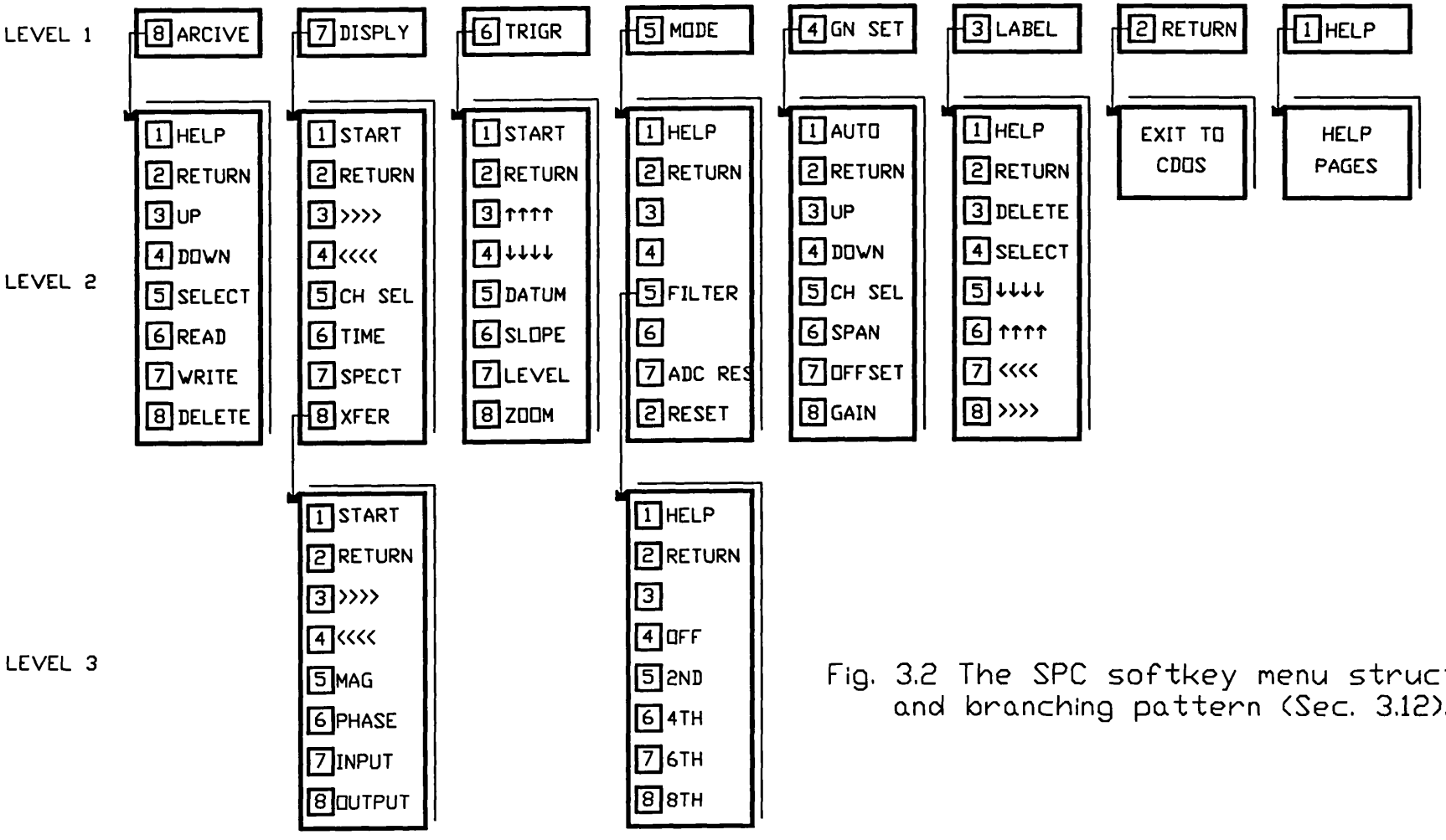


Fig. 3.2 The SPC softkey menu structure and branching pattern (Sec. 3.12).

The main contribution of the author has been in the development of the SPC signal acquisition processor (SAP), signal acquisition card (SAC), programmable anti-aliasing filters and final system integration at hardware and software levels. A non-trivial application program interface was written for real-time data-acquisition and signal processing.

3.3 Signal Processing Computer Hardware

The SPC system architecture (Fig. 3.1) is split between a front-end *signal acquisition processor* (SAP) and the *host* system processor. The SAP is dedicated to handling tasks such as analogue signal conditioning and data-acquisition. The host is responsible for signal processing, handling the user-interface, control and management of peripheral devices and interfacing to the SAP. Both processors are standard 8-bit devices (Z80B) and operate asynchronously. However, the host may synchronise and seize control of the SAP at any time by issuing control commands across the S100 bus. High-speed data transfer between processors is achieved via the *dual-plane memory* (DPM) system. A system overview is given in Plate 3.1.

Application programs are downloaded from a floppy disk for execution in the host local memory space. The SPC user-interface is implemented on a high resolution video monitor. Eight programmable softkeys are provided for user-selection of SPC functions. The user is presented with a menu of the form shown in Fig. 3.2, where the softkeys are labelled according to their function. The host-resident software kernel, responsible for supervising the SPC, monitors the softkeys for a user selection. Once a key has been pressed, an event-handler is activated to perform the selected operation. Some key functions may be set generate a hierarchy of menu options.

The SAP is a slave processor, whose main task is to relieve the host of duties associated with data-acquisition. Software stored in EPROM is used to initialise and define the functionality of the SAP. The host specifies completely the actions it requires the SAP to perform. The dual-processor architecture of the SPC has scope for limited use as a simple parallel processor, where a complex computational task is split between the processors to enhance throughput. However, the advantages of such a scheme should be balanced against the management overhead associated with parallel processing.

Fig. 3.3 shows the main elements of the signal conditioning and data-acquisition system which is resident on the *signal acquisition card* (SAC). Signal degradation due to interference from electro-magnetic and radio-frequency sources is minimised by housing the SAC in a shielded case. The SAC is an intelligent interface card whose parameters are directly programmable via the SAP. Typically, this includes setting the anti-alias filter characteristics, input-channel gain/offset levels, sampling rate and input channel selection.

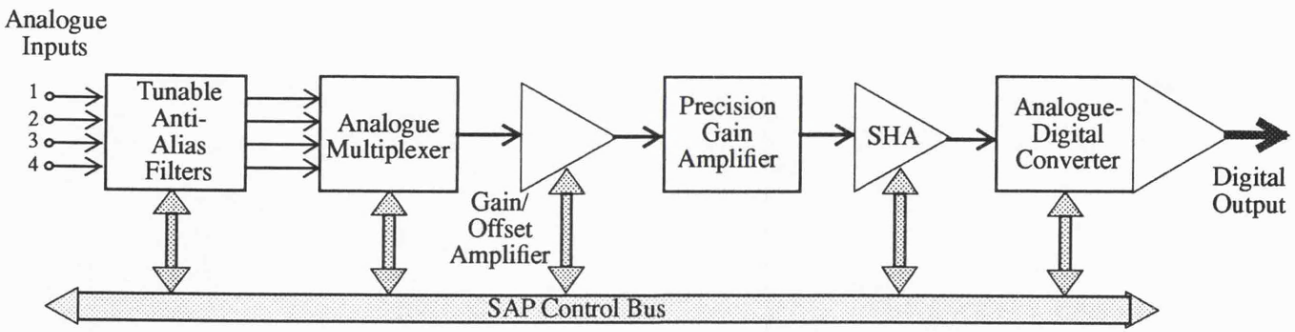


Fig. 3.3 The signal acquisition card (SAC).

Each input channel is anti-alias filtered prior to multiplexing. As each channel is sampled, the SAP will download the pre-calibrated gain/offset settings. A high-precision gain stage boosts the signal amplitude to optimise the signal-to-noise of measurements. The sample-and-hold amplifier (SHA) is used to *freeze* the signal during the conversion cycle of the ADC.

Traditional data-acquisition systems rely on the user to ensure that input signals have voltage levels which optimise the signal-to-noise ratio of measurements. Also, the pre-filtering and gain/offset amplifiers are external devices which must be provided by the user. This often leads to an undesirable clutter of instruments which only serves to complicate the measurement task. The SAC overcomes these difficulties by embedding into the SPC digitally programmable filtering and gain control of input channels.

All digitised data is stored in the dual-plane memory (DPM) which comprises two equal memory planes of 16 KB each. At any instant, one DPM plane will be addressable by the host whilst the other memory plane is addressable by the SAP. The memory planes can be switched between processors by using the memory-swapping technique described in Sec. 3.7.2. A maximum data transfer rate of 3.2 MB/s is achievable using this technique.

The remainder of this chapter describes the individual hardware and software modules that make up the SPC.

3.4 The Host Processor

The host processor card, based on a Z80B (@6 MHz) CPU with 64 KB of dynamic RAM, slots directly into the SPC back-plane and supports a parallel and serial I/O, 3 programmable timers and a PROM resident monitor program. Applications software and data can be downloaded into the host via the 5.25" floppy disk drive. The host can be interrogated at the disk operating system level by connecting a terminal to the serial port of the floppy disk controller.

Software development was performed on a microprocessor system with specifications similar to the SPC. The development environment included a wordprocessor, macro assembler, rational Fortran preprocessor/compiler, runtime language-dependent libraries, user-defined library routines, relocatable linker and assembly-level debugger. This should be compared with the sophisticated software development tools currently available on IBM personal computers. The enhanced efficiency, performance and functionality attainable using IBM-PC based programming environments is demonstrated in Chapter 5.

3.5 The Graphics Display System

The SPC user-interface is a graphics-based menu system. The functionality and effectiveness of the instrument is determined by the speed and resolution of the display system. The main restriction on the choice of graphics display was the requirement that the SPC be a portable self-contained instrument. The graphics system (Fig. 3.1) is based on a 7" JVC monitor and an I/O mapped Matrox ALT-512 graphics controller. The graphics card contains 16 KB of video RAM and is resident on the S100 bus. The maximum resolution supports double-buffered display of up to 256x256x1 pixels.

Host Activity	*Host Processor Interface Control Signals*				*Active Port*
	pDBIN	*$\overline{pWR}$*	*HSTD*	*HSTC*	
Read					
Data	On	Off	On	Off	Data Out
Control	On	Off	Off	On	Control Out
Write					
Data	Off	On	On	Off	Data In
Control	Off	On	Off	On	Control In

a) Host processor interface logic.

SAP Activity	*SAP Interface Control Signals*				*Active Port*
	$\overline{RD}$	*$\overline{WRI}$*	*$\overline{DSTRB}$*	*$\overline{CSTRB}$*	
Read					
Data	On	Off	On	Off	Data Out
Control	On	Off	Off	On	Control Out
Write					
Data	Off	On	On	Off	Data In
Control	Off	On	Off	On	Control In

b) SAP interface logic.

Table 3.1 Truth table of logic signals for Host/SAP input/output operations (see Fig. 3.6).

Using the suite of low-level graphics primitives developed by Hailstone et. al. (1985), it is possible to draw high-level graphics such as grids, axes, labels, lines, frames, markers and waveforms. The graphics utilities was written in Z80-assembly language to maximise the display bandwidth and is available as a library module. High-level access (e.g. via Fortran) to the graphics library is easily achieved since the assembler routines high-level parameter passing conventions.

3.6 Signal Acquisition Processor

The signal acquisition processor (SAP) is a custom-built S100 board, based on a Z80A (@4 MHz) CPU, with timer, parallel/serial I/O, 24 KB static RAM and 8KB EPROM. The SAP plays a central role in the SPC architecture since it interfaces with many modules, including the host, dual-plane memory and signal acquisition card. Fig. 3.4 is a schematic diagram of the SAP, where the S-100 bus interface signals are shown to the left and local bus interfaces to the DPM and SAC are shown to the right of the diagram. The main components of the SAP are described below.

3.6.1 Host-to-SAP Communications

The host can control and monitor the activity of the SAP by issuing I/O instructions via the SAP-resident control and data ports. Switches SW_1-SW_4 in Fig. 3.5 are used to map the SAP in host I/O address space. The SAP I/O ports are decoded by strobing SAPIO from the host addresses A_4-A_7 and the logical state of SW_1-SW_4.

The SAP was designed to support two-modes of communication with the host:

- register-level bi-directional data transfer between processors (Fig. 3.6); this enables the host to control SAP operation and also monitor SAP generated status information.
- high-bandwidth, high-volume block data transfer between processors via the DPM (Sec. 3.7).

Register-level communications is controlled by a latched control/data port (Fig. 3.6) which provides a bidirectional interface between the host and SAP. For example, the host can transmit/receive single byte packets of control/data to/from the SAP. Fig. 3.5 shows the how the SAP control/data port base address is mapped onto the host processor.

Table 3.1 shows the truth table of signals defined in Fig. 3.6 for all possible host/SAP port operations. Processor activity is given in the first column, followed by the status of intermediate control signals. The end column shows the active port (Fig. 3.6) and the current direction of data flow. This scheme avoids bus contention since only one port can be active at any instant. For example, when the host is reading the *data out* port (Fig. 3.6), the output buffers (IC7) will be enabled to give it the necessary privilege. However, the

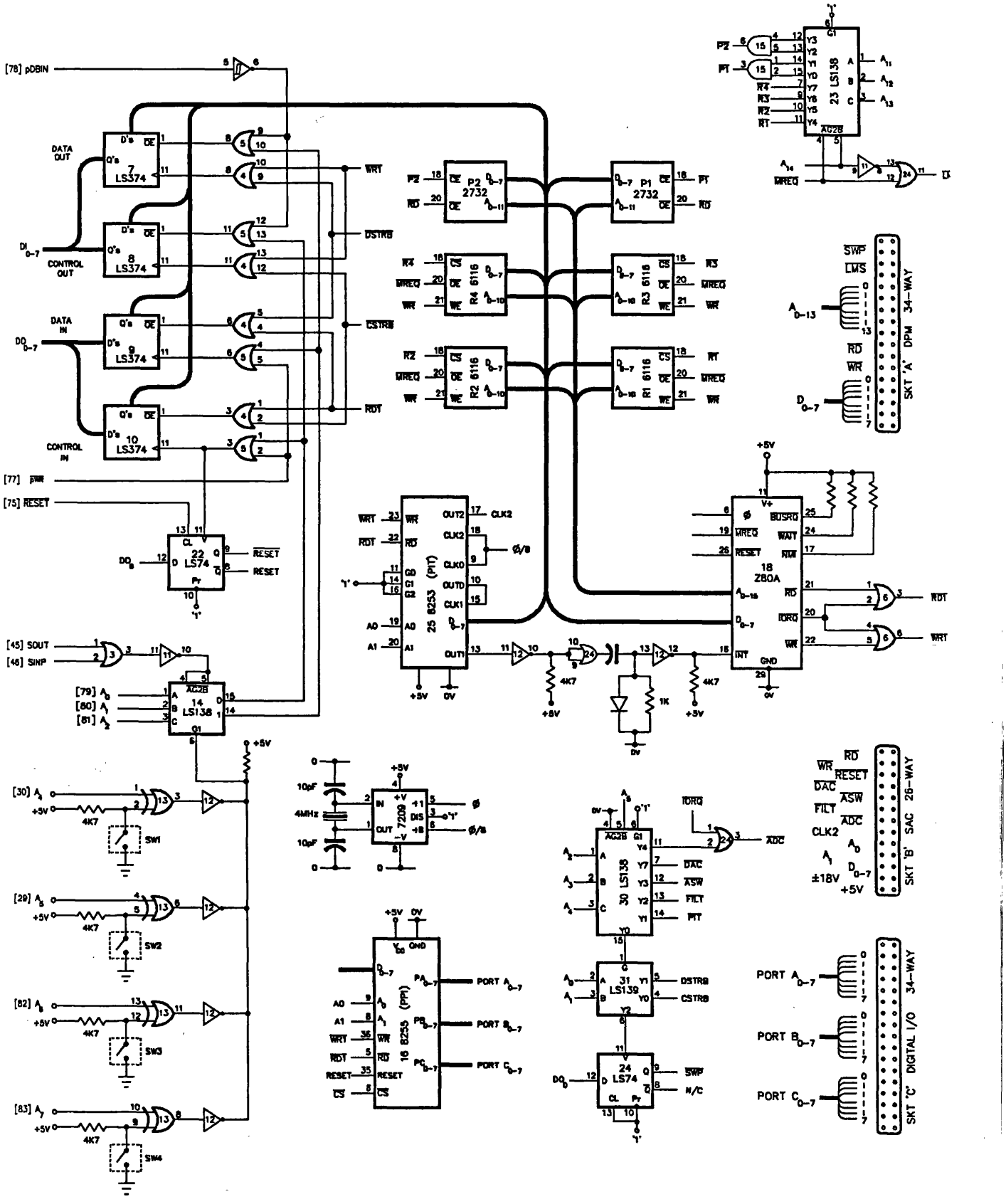
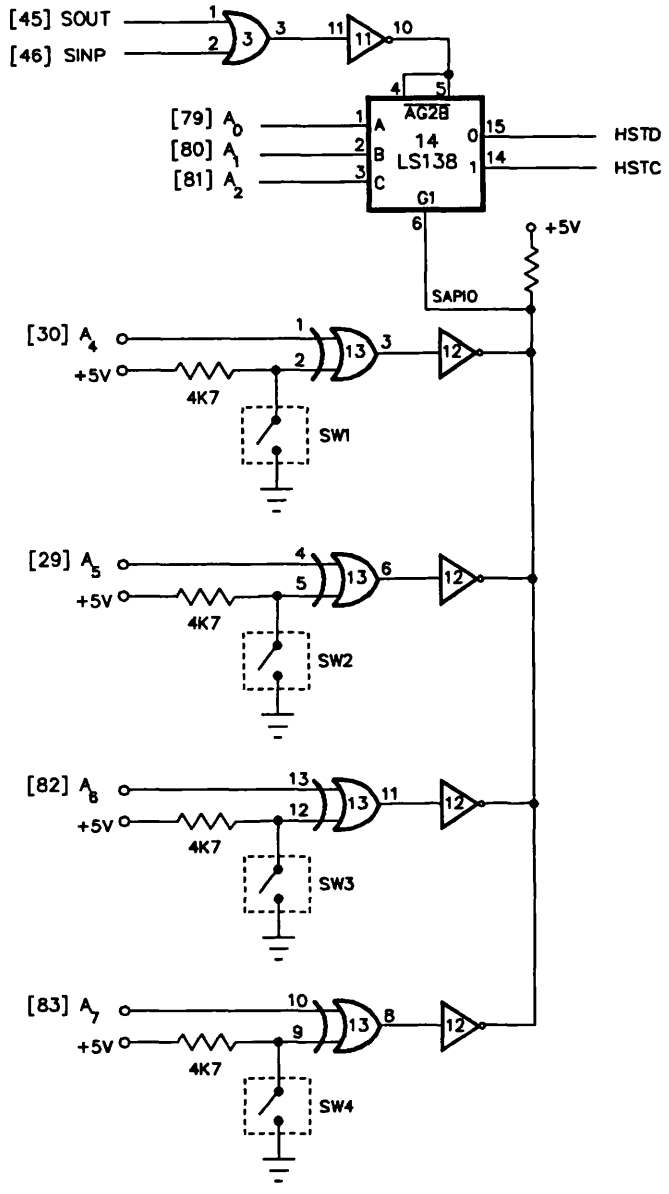


Fig. 3.4 The signal acquisition processor (SAP).



SW1	SW2	SW3	SW4
On	On	Off	Off

Base address = 1100 XXXX (0CX Hex)

Fig. 3.5 Switch bank for selecting the SAP base-address in Host I/O space.

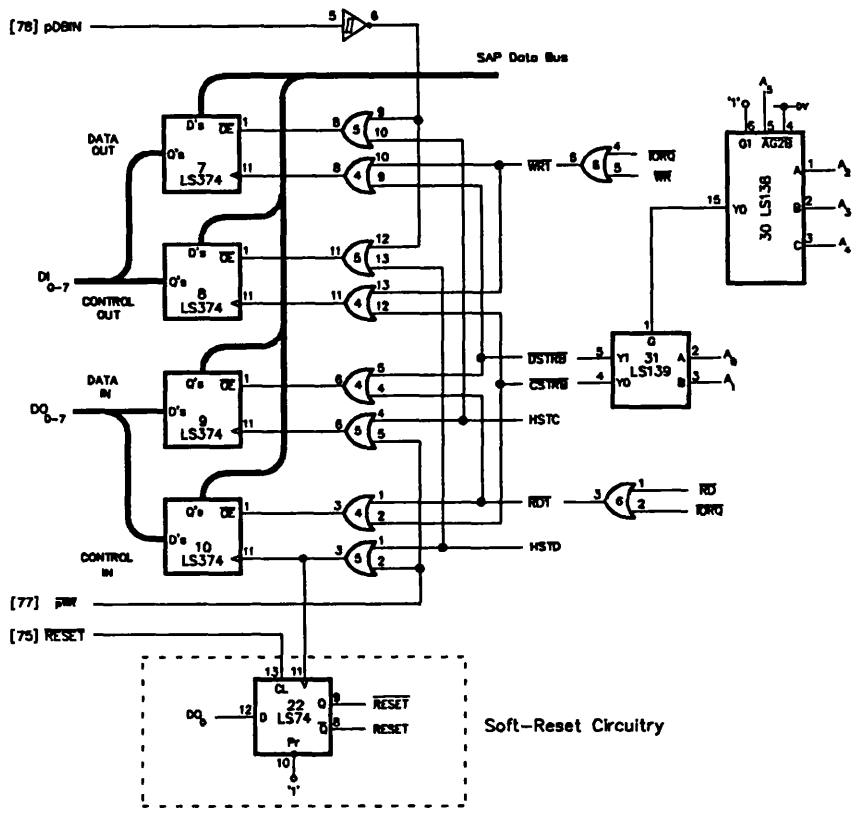


Fig. 3.6 Host-to-SAP bidirectional interface circuitry.

output buffers of *control out* (IC8) are disabled at this instant to prevent bus contention. Sec. 3.11.2 describes the handshake protocol used to transfer information between the two processors.

3.6.2 Software Reset Circuit

The SAP is an asynchronous microprocessor system whose functionality is defined by its EPROM monitor. Following a cold SPC start-up, the SAP monitor program (Sec. 3.11) automatically executes from address 0H to initialise all SAP variables with default values (e.g. sampling frequency, anti-alias filter characteristics, number of channels to sample, data-acquisition buffer size, gain/offset amplifier settings, etc.) to ensure that the system is fully functional.

The reset circuit shown in Fig. 3.6 was designed so that the user could soft-reset the SAP via the host. Hence, the user can reinitialise the SAP to its default parameters without having to exit the main application program. The soft-reset is issued by a single output instruction from the host. The CLK signal to IC22P11 is taken from IC10 (Fig. 3.6), since this goes active low whenever the host writes to the SAP control input port. The SAP *RESET* line mirrors the status of data line DO0.

3.7 Dual Plane Memory System

Real-time data-acquisition requires careful consideration of data input/output and system memory usage. The SPC is based on a dual-processor architecture where data-acquisition is controlled by the SAP and signal processing performed by the host. The dual plane memory (DPM) sub-system was designed for high-speed data transfer between SAP and host. Using the DPM, the SAP can perform continuous data-acquisition at a maximum rate of 20 KWords per second and transfer up to 16 KB of sample data to the host, without losing any samples.

The DPM circuit (Fig. 3.7) is symmetrically divided into two memory planes. Each memory plane comprises 16 KB of RAM, configured as eight 2Kx8 bit static RAM devices, which are tri-state buffered to both host and SAP address/data lines. A processor is granted access to a memory plane by activating the tri-state buffers that link its address/ data lines to the corresponding memory plane.

3.7.1 DPM Memory Mapping

Each processor has exclusive access to a single DPM plane at any given instant. Each DPM plane is dynamically mapped as a contiguous memory block in the linear address space of its parent processor. Fig. 3.8 shows the circuitry used to determine the base

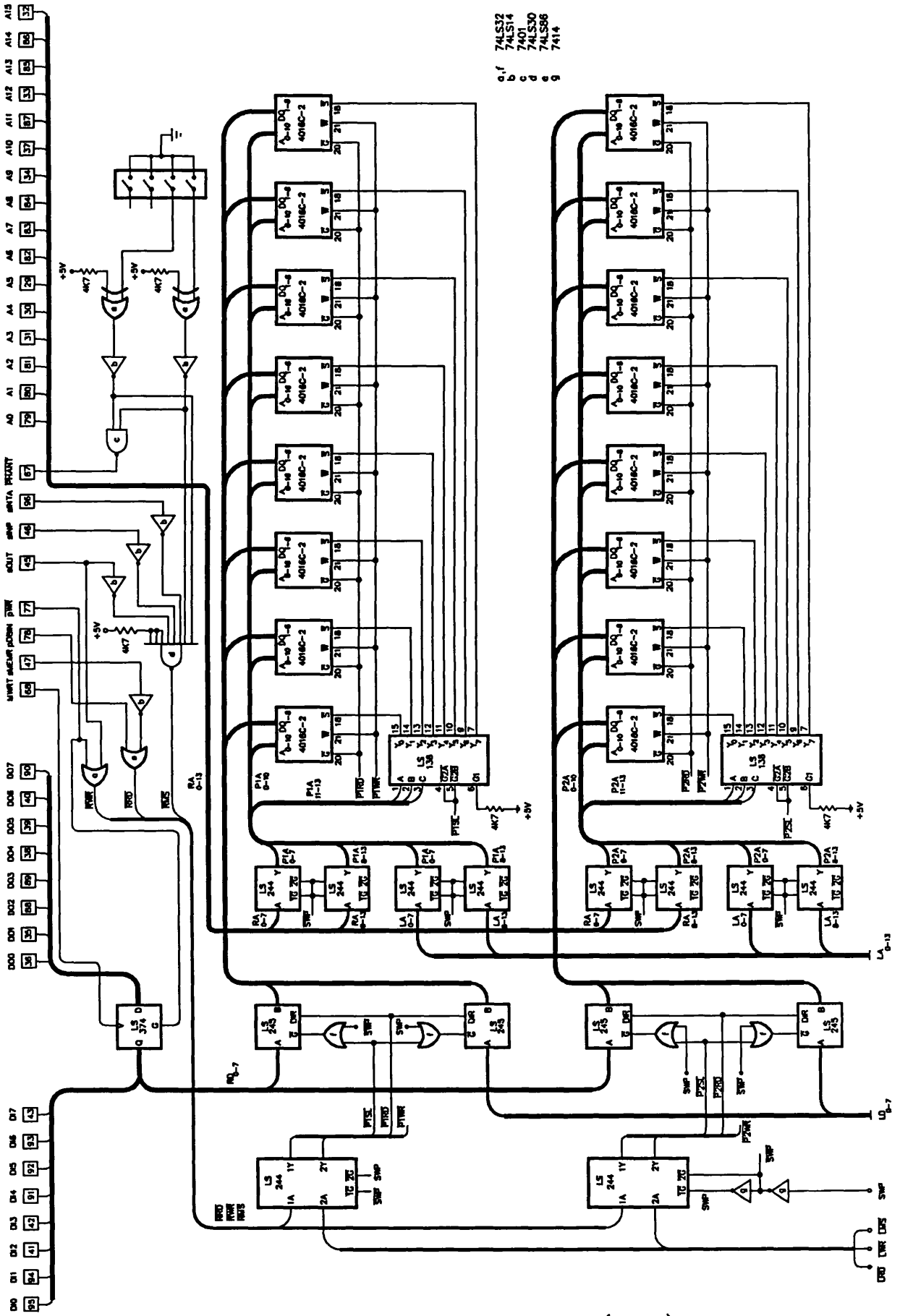
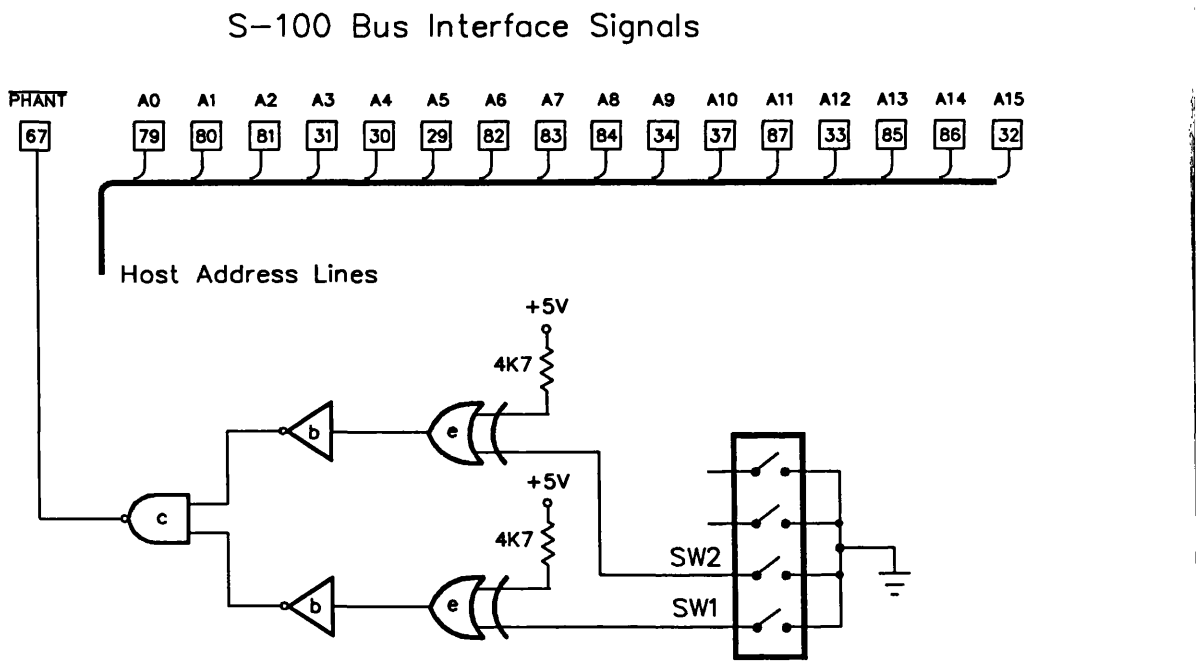


Fig. 3.7 The dual plane memory (DPM) subsystem.



SW1	SW2	A14	A15	DPM Base Address
On	On	'0'	'0'	0000 H
On	Off	'1'	'0'	4000 H
Off	On	'0'	'1'	8000 H
Off	Off	'1'	'1'	C000 H

Fig. 3.8 DPM base-address mapping in host address space.

address of the DPM in host memory space. The inset table shows the switch and address line states necessary to map the DPM onto a given 16 KB host memory boundary. On the SAP, the DPM mapping is hardwired to reside directly above the local RAM.

Since the host processor contains the full complement of 64 KB of RAM, the S-100 *phantom* line was used to bank-switch the DPM into host memory space by disabling a 16 KB block of host-resident RAM.

The DPM memory-mapping was selected to be between 8000H-BFFFH on the host and between 4000H-7FFFH on the SAP. The DPM area is a volatile medium since it can be swapped between processors at any time. Hence, to avoid inadvertent system failure, the programmer must ensure that any critical program/data specific to a given processor lies clear of the DPM buffer.

3.7.2 Memory Plane Swapping

Traditional data transfer techniques using, say, direct memory access (DMA) can achieve data rates of up to 0.5 MB/s. The DPM uses standard logic devices to achieve a maximum data transfer rate of up to 3.2 MB/sec. Such transfer rates are made possible by a scheme known as memory-swapping. Traditional methods operate by physically transferring data between a source and destination block of memory. Memory-swapping redirects the address/data/control lines of a processor to the block of memory containing the required data. Fig. 3.9 shows the method used to implement DPM swapping.

The SAP accesses the DPM memory via:

- control lines (*LMS,LRD,LWR*) which are used to strobe the selected memory block for reading and writing;
- address lines LA_0-LA_{13} can address up to 16 KB of DPM memory;
- data lines LD_0-LD_7 to read or write data into memory.

Fig. 3.9 shows the high degree of symmetry that exists between DPM memory planes. The control/address/data lines of each processor are buffered to each individual DPM plane. For example, the SAP has access to DPM plane '1' (or plane '2') via buffers D1/ SA1/SD1 (or D2/SA2/SD2). The SAP generated *SWP* signal determines the access rights of a given processor to a given DPM memory plane.

The DPM provides for high speed data transfer between processors. Since no physical transfer of data actually takes place, the time taken to implement memory-swapping is limited by the software used to alter the state of SWP. This achieves approximately a six-fold increase in bandwidth over traditional DMA techniques.

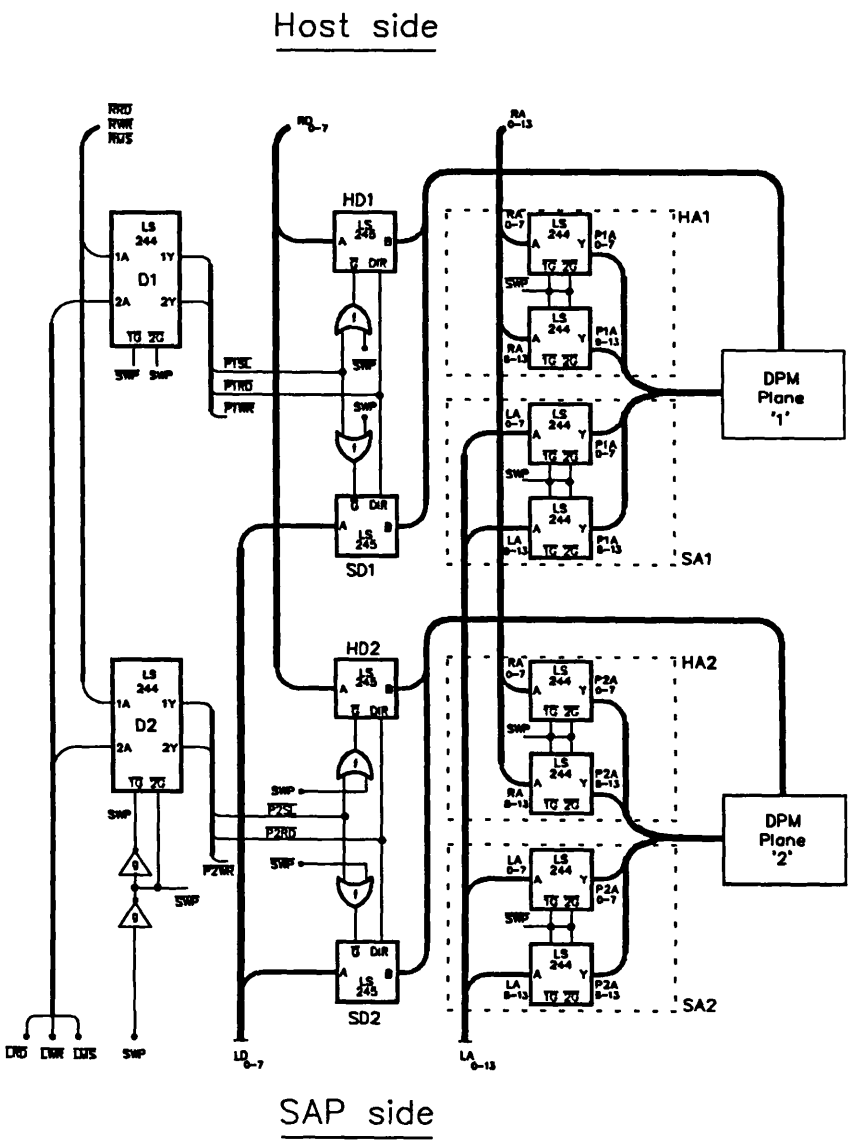


Fig. 3.9 DPM plane swapping circuitry.

3.8 Signal Acquisition Card

The signal acquisition card (SAC) is the computer-interfaced signal conditioning and data-acquisition system of the SPC (Fig. 3.3). The SAC has four anti-alias filtered analogue input channels which are multiplexed to a single analogue-to-digital converter (ADC). The SAC can achieve a maximum sampling rate of 3 KHz per channel- with all channels active. This sampling rate fulfils the requirements of many biomedical and industrial measurements.

The signal-conditioning amplifiers presented many design challenges due to the limited *a-priori* knowledge of the application domain. Consideration was given to a wide variety of parameters that determine the quality of an amplifier, including the sensitivity, common-mode rejection (CMR), noise, and static protection. Ideally, the SAC should have floating inputs which are isolated from chassis ground and all digital communications should be via opto-isolated devices. However, the added cost and complexity was not considered viable for a prototype system.

The ADC resolution can be switched between 8- or 12-bit modes under SAP control. Switching from high to low ADC resolution allows for higher sampling rates and halves the memory requirements for a given measurement. The 12-bit resolution of the ADC provides an accuracy of up to 1mV. However, to ensure that 12-bit accuracy is maintained, input amplifier noise and all external switching noises must be kept below 1 mV over a wide bandwidth.

The SAP has direct control over the SAC and can program it to perform low-level signal processing of input channels. In particular, the SAC can be programmed to perform auto-calibration of input channels by applying reference voltages at the SAC inputs. In this way, the SAP can build up a table of reference voltages versus measured voltages, which can then be used to formulate a calibration equation for the input system. A digital correction factor can then be applied to all measurements to compensate for the imperfections of the SAC.

Microprocessor-based signal conditioning overcomes the problems associated with gain control using potentiometer devices. The SAC uses digitally programmable preamplifiers and attenuators which have a dynamic range of 72 dB and a possible 4096 unique gain settings. The SAP is used to program the input gain and offset amplifiers of the SAC by writing to special register locations via a local control bus.

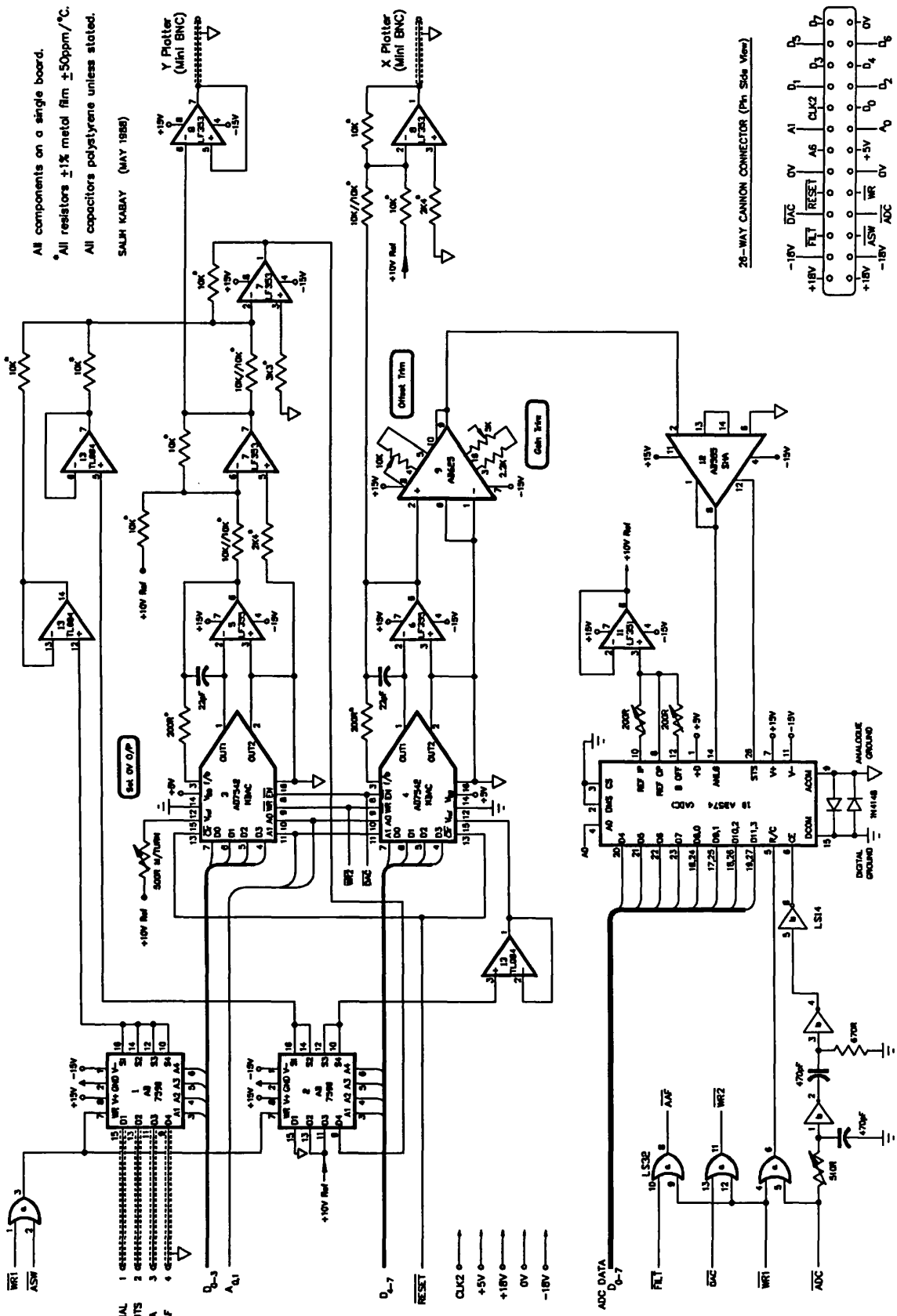


Fig. 3.10 The signal acquisition card (SAC).

3.8.1 Automatic Signal Conditioning

The SAC can be digitally programmed under the direct control of the SAP. The main programmable devices in the SAC (Fig. 3.3) include:

- the anti-aliasing filters whose cut-off frequency and attenuation characteristics can be set to suit the experimental conditions (Sec. 3.8.3);
- the analogue switches (AD7590) used for multiplexing the selected analogue input channels; the SAP uses data lines D_0-D_3 (Fig. 3.10) to set the multiplexer (IC1, AD7590) sampling sequence;
- gain and offset adjustment of input signals.

The programmable gain/offset amplifiers of the SAC are shown in schematic form in Fig. 3.11. During normal data-acquisition, the SAC mode control switch is set to apply zero volts to summer S1. The output from S2 represents the original input signal plus a programmable DC offset generated by the offset MDAC. This signal is then switched through to the reference input of the gain control MDAC. Coupled with instrumentation amplifier, the final MDAC stage operates as a programmable amplifier.

The SAC mode control switch can also be configured for performing calibration tests. In this mode, the SAP disables all inputs to the multiplexer and applies an internally generated reference voltage of 0 V (or 10 V) to summer S1 and 10 V to the reference input of the gain MDAC. This allows the SAP to perform internal test calibrations on all components of the SAC independently of any external devices.

3.8.2 Analogue-to-Digital Converter

A sample and hold amplifier (SHA) is an integral component of all data-acquisition systems. This requirement is due to the finite conversion time of ADCs. In Fig. 3.10, the *STS* output of the ADC is used to trigger the SHA to switch from *tracking* the signal to *hold* mode, where the SHA output is held at a constant voltage level corresponding to the sampling instant. This leaves the ADC with plenty of time to perform the conversion.

The ADC can be programmed to operate in either 12 or 8-bit resolution. This feature allows the operator to trade-off memory requirements with ADC resolution. A lower ADC resolution also results in a 30% increase in system bandwidth.

3.8.3 Anti-Aliasing Filters

Inevitably, most signal processing systems will have to operate on signals contaminated with noise. Anti-alias filtering is often necessary prior to digitisation to prevent *aliasing* (Sec. 2.2) and to reduce the high-frequency noise content of the input signal.

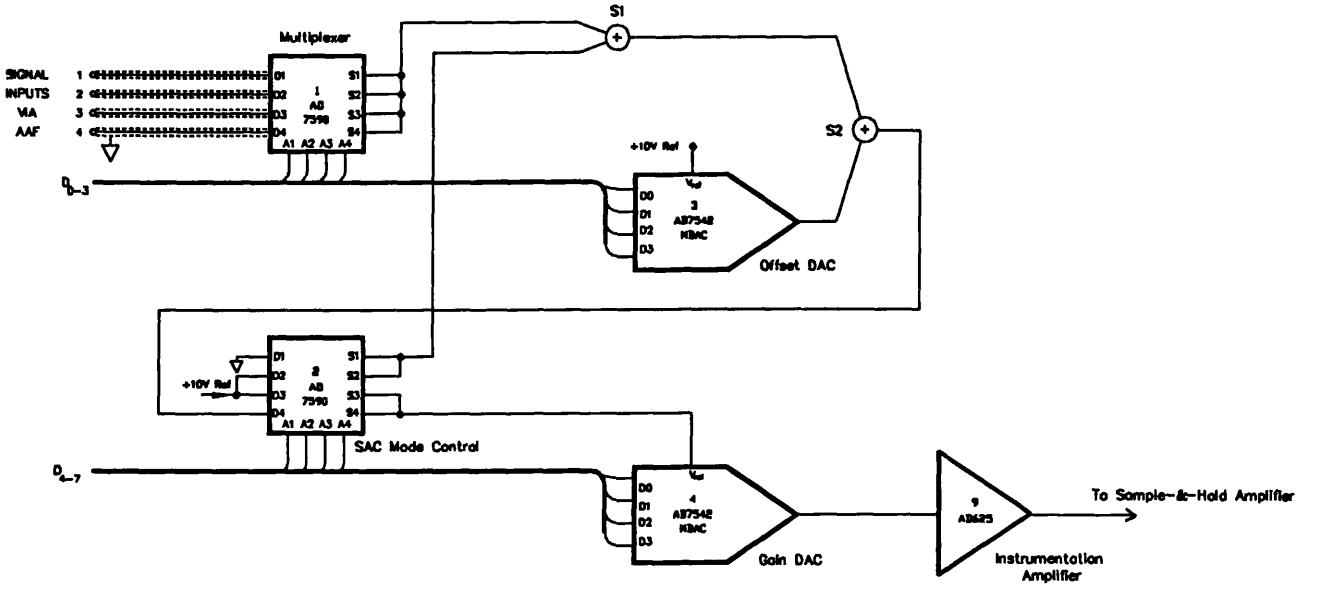


Fig. 3.11 A simplified schematic of the programmable gain/offset circuit of the SAC given in Fig. 3.10.

The SAC has four channels of anti-alias filtering with programmable cut-off frequency and attenuation characteristics. The filters were designed around a digitally programmable set of switched-capacitor filter devices (Fig. 3.12) whose cut-off frequency can be precisely controlled using the input clock. Only two external resistors are required to set the filter passband gain and Q-value. Matched filter characteristics across input channels are easily attainable with this approach.

Similar performance using traditional filter circuits could only be accomplished by careful selection of components and painstaking adjustments to filter time-constants. Also, any change in filter characteristics would require a new filter circuit and component values.

3.9 Signal Processing Computer Software

The SPC was designed to provide a system solution to some of the general problems of measurement recording, data-acquisition and signal processing. The software written to drive the SPC can be separated into three independent modules:

- the host-resident generic kernel that provides a library of primitive interface routines to the SPC hardware (Sec. 3.10). The kernel defines the windowing and menu system, graphics user interface, the SAP communications protocol, etc.
- the SAP-resident monitor program which initialises the SPC front-end processor for the task of data-acquisition and communications with the host (Sec. 3.11).
- the application program interface used to configure the SPC generic kernel to perform in a dedicated application domain (Sec. 3.12.1).

The remainder of this chapter is devoted to a description of these software modules[1].

3.10 The Host Generic Kernel

The host-resident generic kernel encapsulates flexibility and low-level machine intelligence to support data-acquisition, signal processing and control for user-definable application domains. The generic kernel provides:

- a definition of the underlying system data structures;
- supervision of softkey selections and task allocation;
- menu-management, to control menu propagation and backtracking;
- graphical user-interface management;
- communications and control between the host and SAP.

1. Source-code listings can be obtained from Prof. N.B. Jones, Engineering Department, University of Leicester, Leicester LE1 7RH.

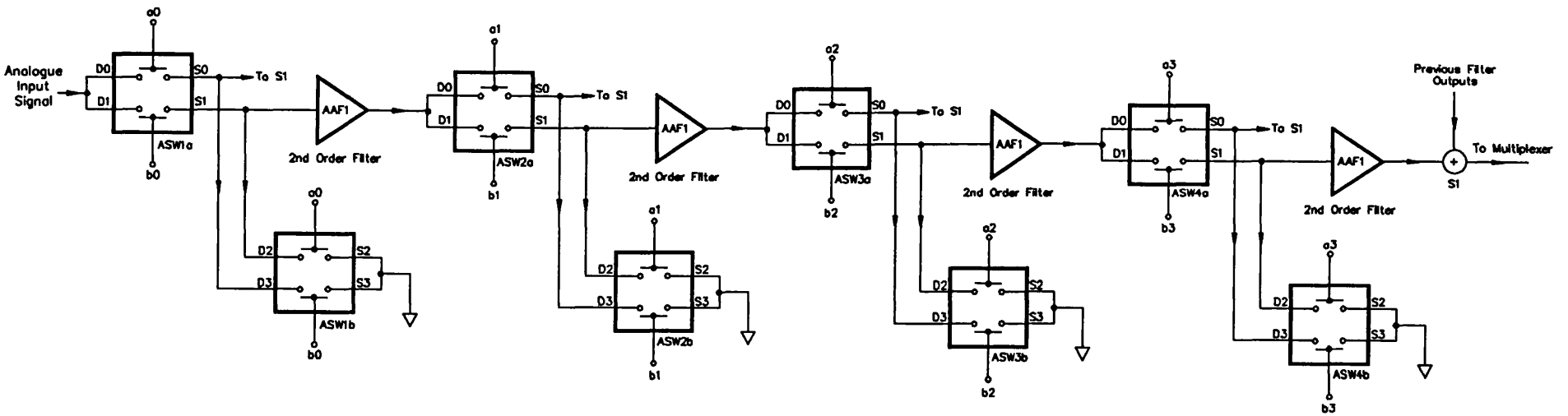


Fig. 3.12 A single channel digitally programmable switched-capacitor anti-aliasing filter. The filter characteristics are controlled by diverting the input signal (via the analogue input switches ASW1-4) through the required number of 2nd order filter stages.

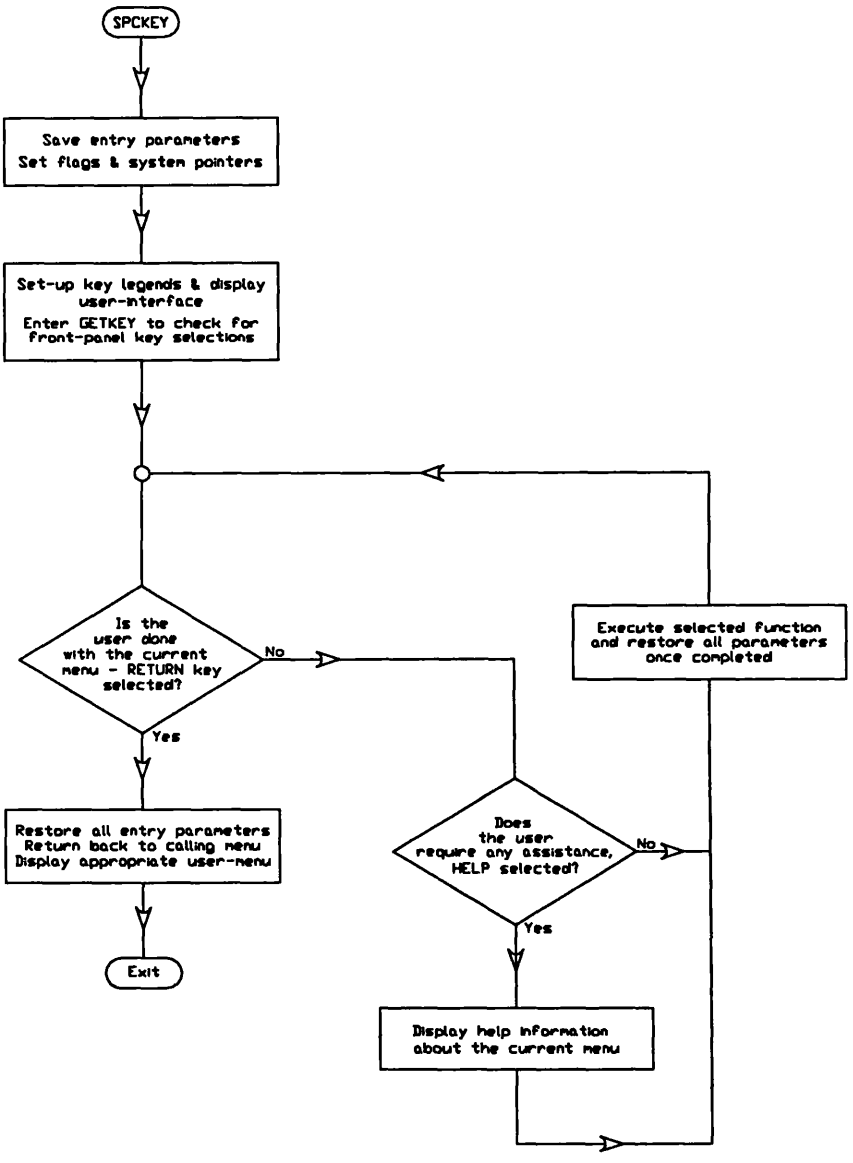


Fig. 3.13 The SPCKEY macro which is used to generate the SPC menus.

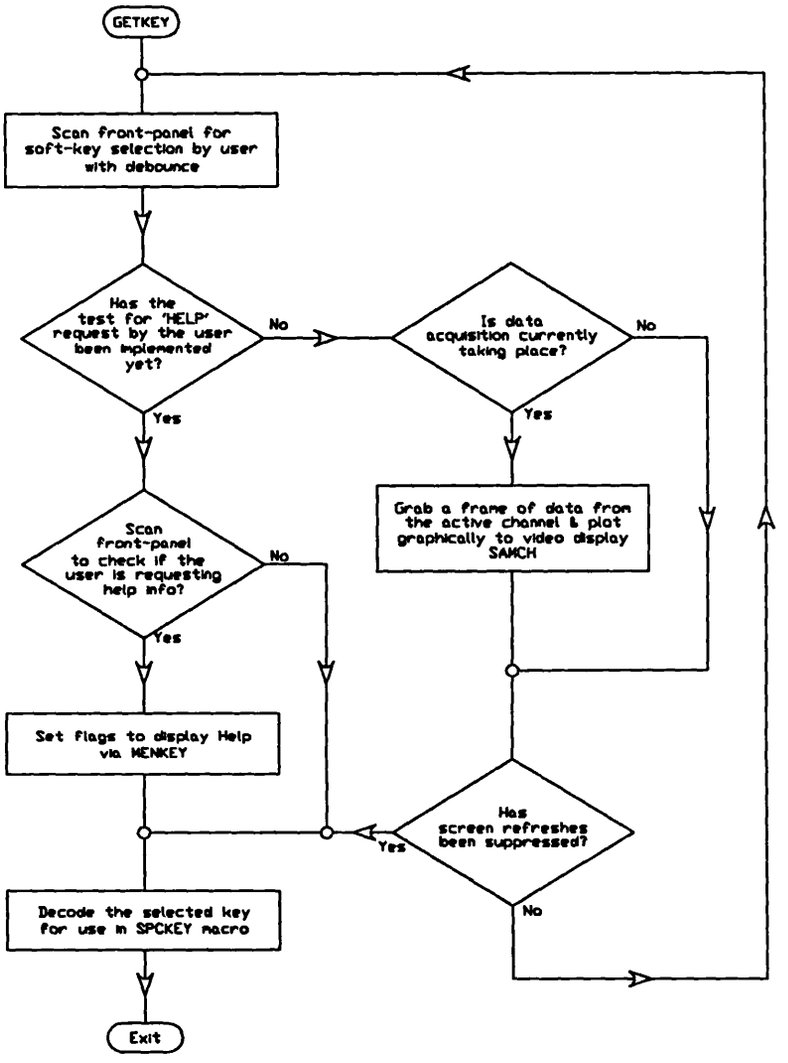


Fig. 3.14 The GETKEY function is used to monitor the status of the softkeys.

The SPC user-interface is based around eight programmable softkeys which are used to select system-wide operations; they are not application specific. The softkeys are defined by the application program interface (API) to support the required measurement and control tasks. The softkey legends should be representative of the functions they perform. A typical key legend (Fig. 3.2) will have a complex branching menu-structure.

The following sections will describe the above operations in greater detail.

3.10.1 Softkey Decoding

The SPCKEY macro (Fig. 3.13) is used to coordinate the flow of data within the application program. The softkeys are polled according to the GETKEY algorithm specified in Sec. 3.10.2. A user-request is activated when the operator selects a softkey. Once a selection is made, program flow branches to the relevant subroutine to implement the assigned function. For example, selecting the ARCHIVE softkey (Fig. 3.2) generates a new menu containing options relevant to data storage/retrieval.

A recursive calling sequence to SPCKEY can be used to generate multi-level menu pages (Fig. 3.13). Backtracking to the calling menu is controlled by the contents of the stack. Since all softkey functions are implemented as subroutines, the return path is automatically saved onto the stack. On returning from a particular function, program flow is restored into the most recent invocation of the SPCKEY macro.

The SPCKEY method of softkey decoding presents a consistent data structure for menus and also an automatic method of backtracking that relieves the host of complex resource management software.

3.10.2 Softkey Polling

The softkeys are monitored using the software polling technique implemented within GETKEY (Fig. 3.14). Software polling is generally less efficient than interrupt-driven event-handling and will result in wasted CPU time. However, GETKEY incorporates greater functionality into the polling-loop to improve CPU efficiency by executing background tasks in between softkey selections.

The SPC user-interface supports learning with context-sensitive help for the novice user, without being verbose with the more proficient user. The concept of context-sensitive help is now an established part of user-interface design. However, the method by which help is activated still requires some effort on the part of the user. The SPC generates context-sensitive help based on the duration of a key press. The activation of help information is controlled by GETKEY, which runs a background task to check the time for

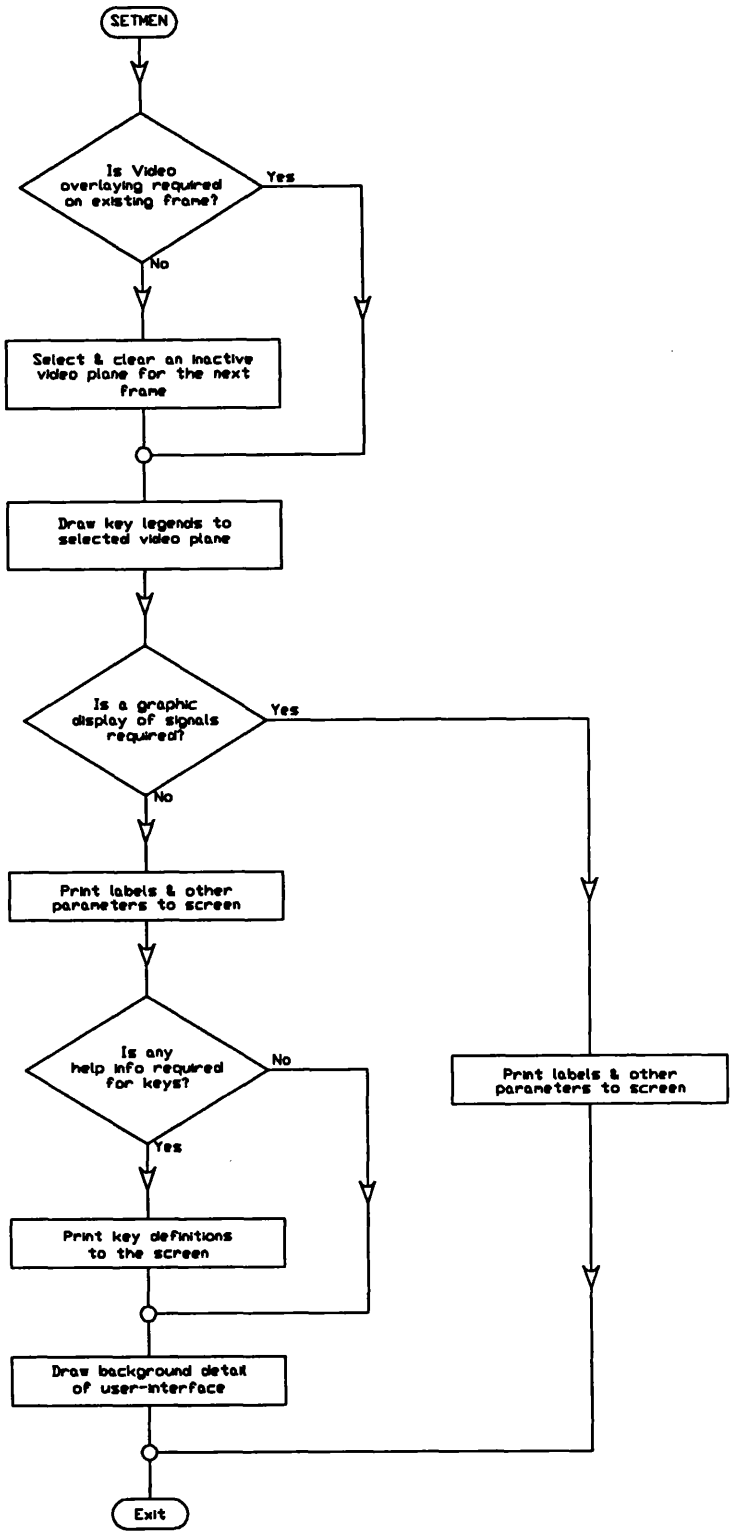


Fig. 3.15 The SETMEN function used to drive the host graphic user-interface.

which a softkey remains active. If this time exceeds a tenth of a second, GETKEY will enable the help flag HFLAG which informs SETMEN (Fig. 3.15) to pop-up the relevant help page on the selected softkey.

GETKEY can also spawn off background processes to handle data-acquisition for menus (e.g. GnSET and TRIGGER) that require user decisions based on data acquired from the front-end. For instance, when the TRIGGER menu is active, the SPC must continuously capture and display frames of trigger data. The user can then halt data-acquisition to proceed with the definition of a trigger template (Sec. 3.12.2A.).

3.10.3 Graphic Menu Interface

Screen layout is an important part of every user-interface. The SPC user-interface is structured within the host generic kernel in an effort to standardise the screen layout (Plate 3.2) across all applications. Whilst, the API can define the softkey legends, the number of menus and the functions associated with softkeys - it cannot alter the format of the user-interface. For example, the host kernel will pick-up the softkey legends and place them in the relevant softkey locations.

All softkey definitions and menu information is handled by SETMEN (Fig. 3.15). The menu system can support many different display formats by setting the appropriate control flags. A softkey selection is highlighted, by inverting the video mode of the softkey legend, to acknowledge the selection.

3.10.4 Host-SAP Flexible Control Interface

This section defines the software component of the host-to-SAP interface circuitry described in Sec. 3.6.1. The software interface is based on a simple communications protocol with only three message types: command, status set and status read (Sec. 3.10.5). The control-, data-, and status-port structures are described below.

A. Control Port

All communications from host-to-SAP must be via the control port. This is an 8-bit register reserved for informing the SAP of the required operation. The control register has the form shown below (Fig. 3.16):

B_7	B_6	B_5	B_4	B_3	R/$\overline{S}$	$\overline{RDY}$	$\overline{SAPRST}$

Fig. 3.16 The host-to-SAP control port.

- $\overline{SAPRST}$ is used to force a soft reset of the SAP (Sec. 3.6.2);
- $\overline{RDY}$ indicates that the host is ready to receive data from the SAP;

- $R/\overline{S}$ is a flag that sets the SAP to perform a read or set operation;
- B_3-B_7 are used to address the functions listed in Sec. 3.10.5.

B. Data Port

The host data port is used for transferring new data-acquisition parameters to the SAP, see Category 2 functions listed in Sec. 3.10.5. The host uses the SAP data port to read existing SAP parameter values see Category 3 functions listed in Sec. 3.10.5.

C. Status Port

The current activity of the SAP is always available to the host via the SAP status port. The structure of this port has the form shown below (Fig. 3.17):

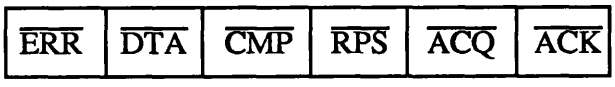


Fig. 3.17 The SAP status port.

- $\overline{ACK}$ the SAP is listening to the host;
- $\overline{ACQ}$ the SAP is currently busy;
- $\overline{RPS}$ the SAP requests a plane swap;
- $\overline{CMP}$ shows the current memory plane being addressed by the SAP;
- $\overline{DTA}$ the SAP is sending data to the host;
- $\overline{ERR}$ an error condition has been trapped.

The communications protocol is handled by software drivers resident on each processor. The host protocol handler (HST) is outlined in the flow diagram shown in Fig. 3.18. HST is responsible for bidirectional communications with the SAP protocol handler SAPPRO (Sec. 3.11.2).

The host-to-SAP handshake protocol is controlled by the status signals $\overline{ACK}$ and $\overline{ACQ}$. For example, before the host can execute any of the functions listed in Sec. 3.10.5, it must wait until the SAP deactivates $\overline{ACQ}$. This condition ensures that the SAP is ready to receive communications from the host (e.g. $\overline{ACK}$ becomes active). During data-acquisition, the $\overline{RPS}$ flag informs the host that the current DPM buffer has been filled and requests permission for a DPM plane swap. The $\overline{CMP}$ bit keeps track of the DPM plane currently *phantomed* into the host address space. In the event of a hardware failure, the SAP can inform the host of the error by activating $\overline{ERR}$ and loading the error code into the SAP data port. This system provides the host with a comprehensive and effective method of controlling SAP activity.

Function Name	*Control Code (Hex)*	*Read Description*
SMPS	09	Data/Program Address
SSTRT	11	Data/Program Length
SSTOP	19	Sampling Interval
SXFR	21	Frame Number
SXEC	29	SAC Input Channels
SPLOT	31	Analogue Gain Control
SDAC	39	Analogue Offset Control
RESET	FE	Soft Reboot of SAP

Table 3.2 Command message functions.

Function Name	*Control Code (Hex)*	*Set Description*
AAFSLP	01	AAF Rolloff Selecter
SAFLO/HI	71/B1	AAF Cut-Off Frequency
SDPALO/HI	41/49	Data/Program Address
SDPLLO/HI	51/59	Data/Program Length
SSILO/HI	61/69	Sample Interval
SCHSPC	79	SAC Input Channels
SGANLO/HI	81/89	Analogue Gain Control
SOFFLO/HI	91/99	Analogue Offset Control
SDPMLO/HI	A1/A9	DPM Buffer Length
SPLTIM	B1	X-Y Plotter Time-Base
SWDLEN	B9	ADC Wordlength
SFUN1-8	C1-F9	User-Definable Functions

Table 3.3 SAP set message functions.

Function Name	*Control Code (Hex)*	*Read Description*
RDPALO/HI	45/4D	Data/Program Address
RDPLLO/HI	55/5D	Data/Program Length
RSILO/HI	65/6D	Sample Interval
RFRNO	75	Frame Number
RCHSPC	7D	SAC Input Channels
RGANLO/HI	85/8D	Analogue Gain Control
ROFFLO/HI	95/9D	Analogue Offset Control
RDPMLO/HI	A5/AD	DPM Buffer Length
RPLTIM	B5	X-Y Plotter Time-Base
RWDLEN	BD	ADC Wordlength
RFUN1-8	C5-FD	User-Definable Functions

Table 3.4 SAP status message functions.

3.10.5 SAP Software Functions

This section tabulates the suite of SAP resident routines which enable the host to control the SPC front-end.

Category 1: (Command Messages)
Command messages are used for standard SAP operations which do not require any I/O through the data port (Table 3.2).

Category 2: (SAP Set Messages)
This category of SAP functions are used to update front-end parameters and require data input from the host (Table 3.3).

Category 3: (SAP Status Messages)
This set of functions allow the host to interrogate important SAP memory addresses and require data output from the SAP (Table 3.4).

3.11 Signal Acquisition Processor Monitor Program

This section aims to give an overview of SAPMON, the monitor program used to control the SAP (Sec. 3.6). The tables of Sec. 3.10.5 summarise the suite of functions supported by SAPMON. In general, SAPMON forms the low-level host interface to the SPC front-end. The main functions of SAPMON can be summarised:

- initialisation of SAP and SAC on reset;
- software controlled signal conditioning;
- management of the data-acquisition process;
- handling bi-directional communications with the host.

The remainder of this section will describe these in greater detail.

3.11.1 SAP Initialisation

The bootstrap procedure SAPRST initialises the SAP and sets the SAC parameters to default values. This routine performs diagnostic testing of the DPM and other hardware. If a fault is detected, an error message will be sent to the host so that corrective action may be taken. The SPC configuration following boot-up is as follows:

- Active Inputs: Channel '1'
- Bandwidth: 900 Hz
- ADC Resolution: 12-Bits
- Amplifier Gain/Offset: 36 dB
- Anti-Aliasing Filter: 2nd Order Butterworth (-3 dB @900 Hz)
- Sample Buffer Size: 16 KB
- DPM Base Address: 8000 H

Once these parameters are set, the SAP enters the protocol handler SAPPRO ready for commands from the host.

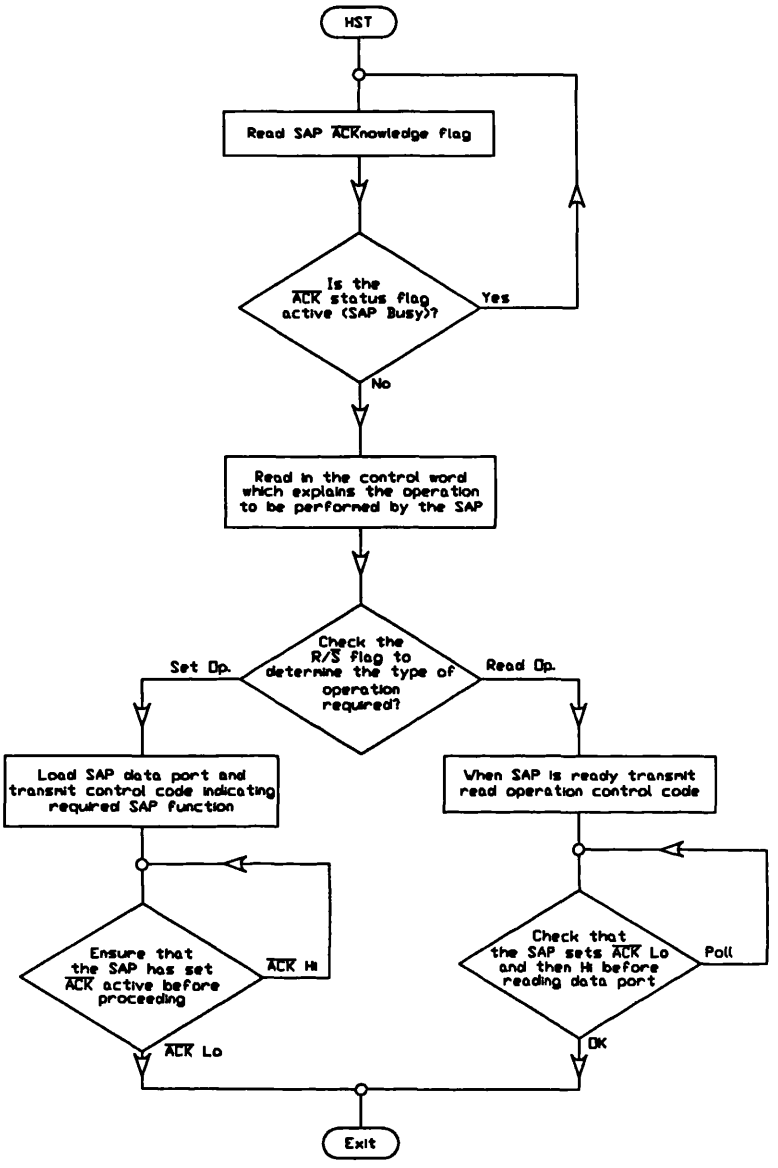


Fig. 3.18 The SPC host protocol handler (HST).

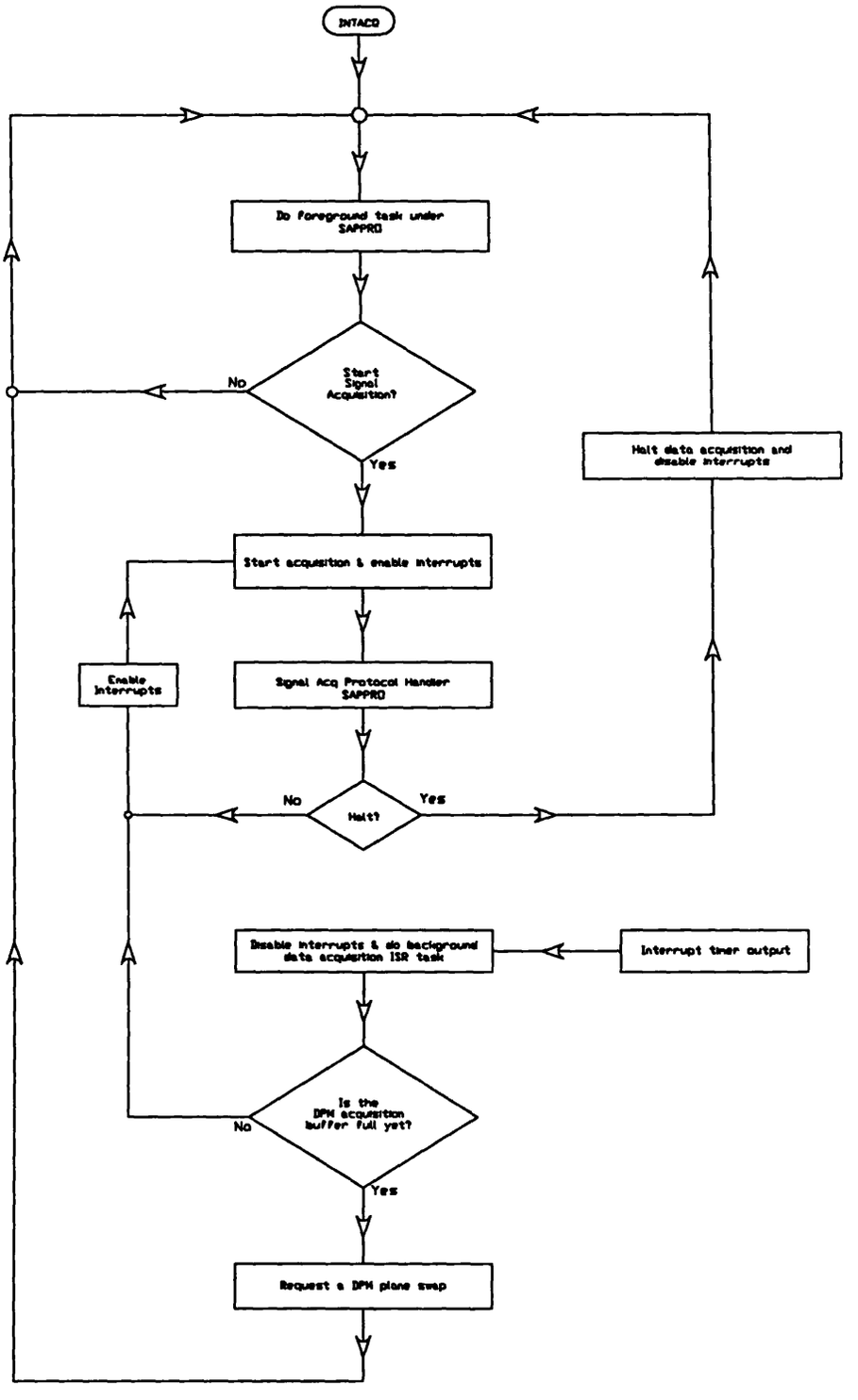


Fig. 3.19 The sequence of events for interrupt-driven data acquisition INTACQ.

3.11.2 SAP Protocol Handler

The transfer of control codes and status information between the SAP and host follows the protocol defined in Sec. 3.10.4. The SAP-resident protocol handler (SAPPRO) is responsible for handshake communications with the host, using the control, status and data I/O ports defined in Sec. 3.10.4.

SAPPRO enters a standby mode in between function calls and data-acquisition duties, where it polls the control port $\overline{\text{RDY}}$ bit. An active $\overline{\text{RDY}}$ flags that the host has sent a command. In this case, SAPPRO releases the SAP to perform the required task. The control port R/$\overline{\text{S}}$ bit indicates whether the host is performing a read or write operation. The SAP control word interpreter (SAPCWI) is then used to decode the control word and branch off to the addressed function call.

3.11.3 Interrupt Driven Data-Acquisition

For maximum efficiency, the SAP implements data-acquisition as a background task using an interrupt driven scheme. SAPPRO is invoked as a foreground task in between samples, releasing the SAP to perform tasks set by the host. The interrupt service routine INTACQ is implemented each time a new sample is required (Fig. 3.19).

The SAC provides high-quality automatic signal conditioning at relatively low cost. This is achieved by multiplexing the input signals through a single amplifier stage under program control. As the multiplexer activates an input channel, the pre-set gain and offset words for that channel will be loaded into the respective MDACs to provide the necessary signal conditioning (Sec. 3.8.1).

3.12 An Application Program Interface

Using the software modules described in the previous sections, an application program interface (API) has been written to configure the SPC generic kernel for real-time spectral and time-domain analysis of clinical data. The main application domains that were considered included analysis of blood pressure and flow signals in cardiovascular studies, assessment of respiratory dynamics during high frequency jet ventilation and clinical studies of electromyographic signals. Simulation studies for performing cardiovascular (Kabay, 1985) and electromyographic (Sehmi, 1988) studies had initially been implemented around the SPC host kernel. These studies were a pre-cursor to the work undertaken by the author to develop the SAP front-end system and interface to the host.

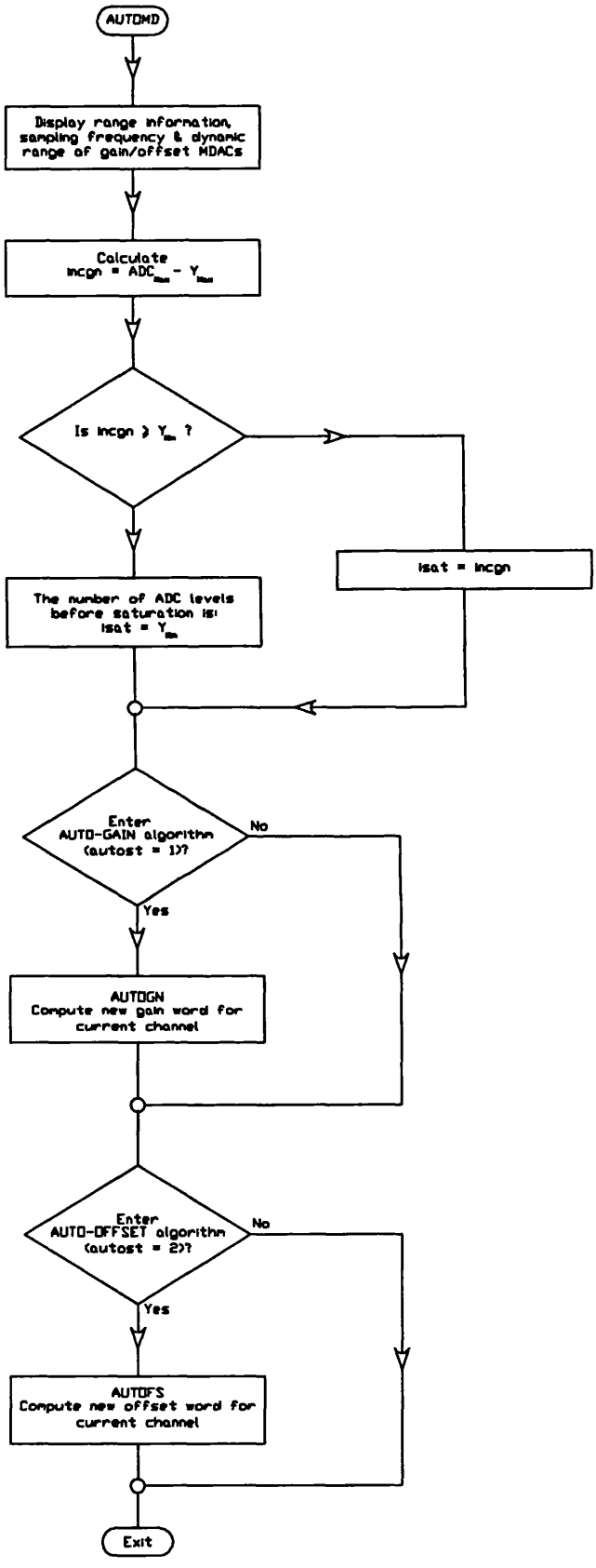


Fig. 3.20 AUTOMD is used to provide auto-ranging gain control on the input channels.

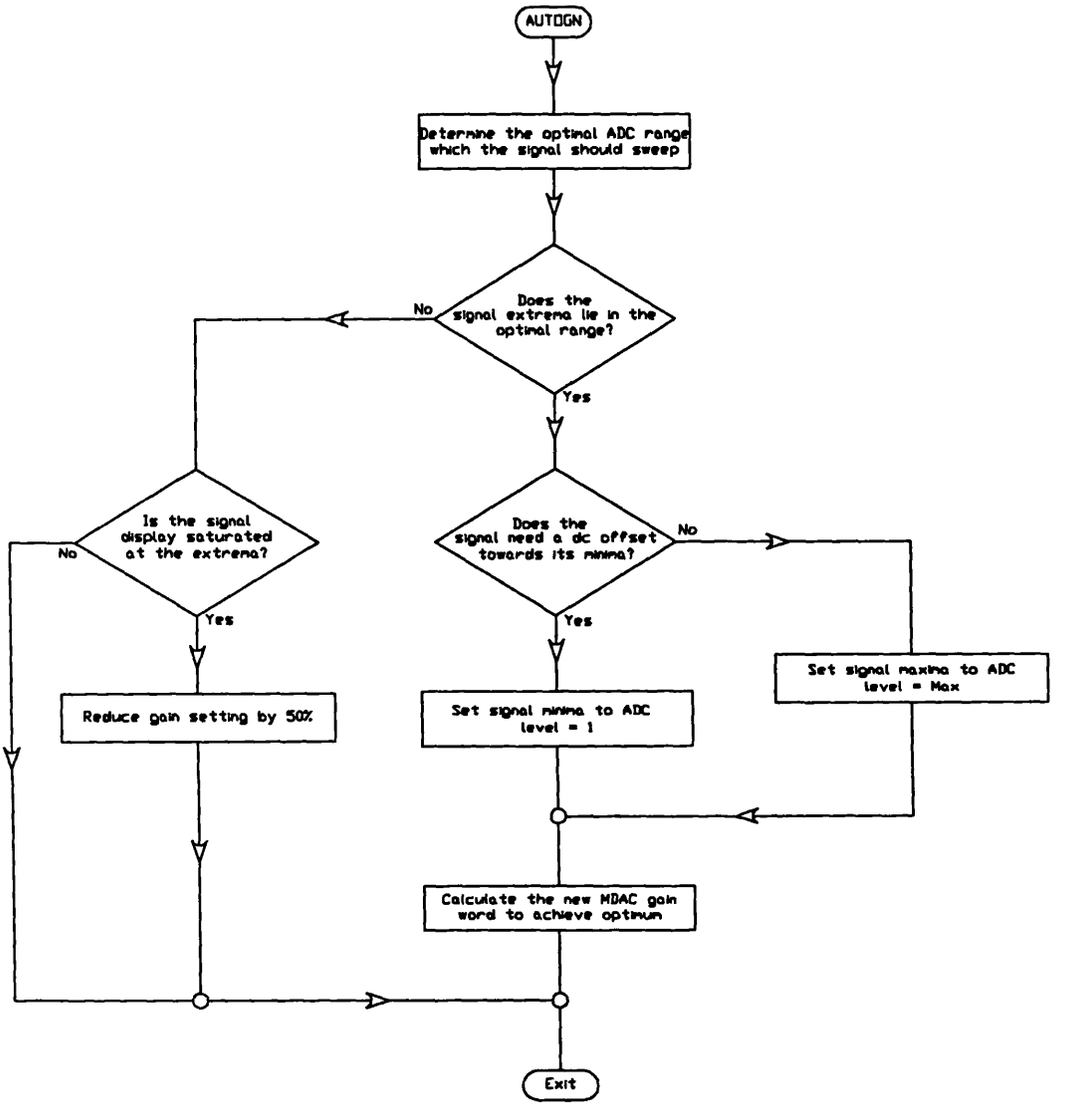
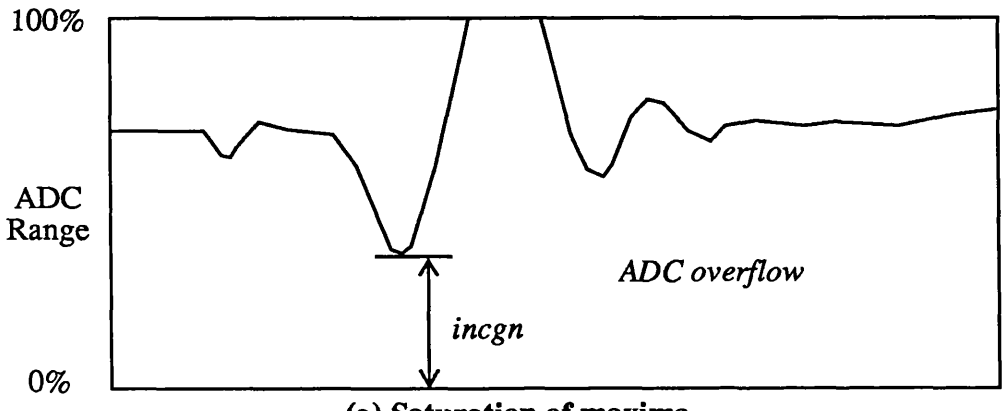
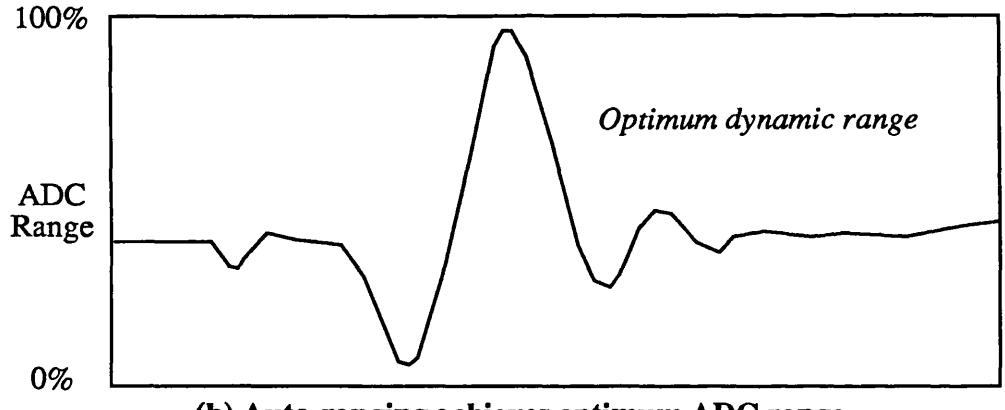


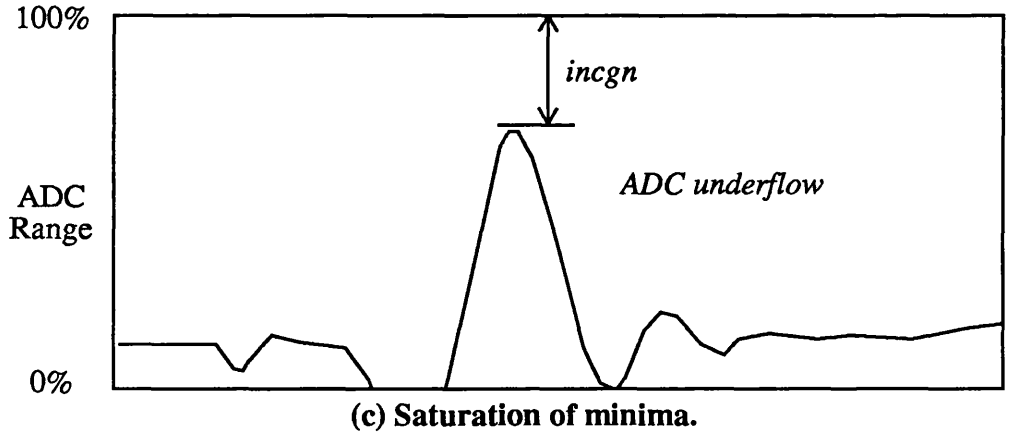
Fig. 3.21 AUTOGN implements the auto-ranging gain control algorithm.



(a) Saturation of maxima.



(b) Auto-ranging achieves optimum ADC range.



(c) Saturation of minima.

Fig. 3.22 Examples of possible scenarios requiring auto-ranging gain/offset adjustment.

3.12.1 Software Controlled Data-Acquisition

This section describes the API algorithms implemented for controlling data-acquisition and auto-ranging signal conditioning.

A. Automatic Analogue Signal Conditioning

The concept of digitally programmable signal conditioning was introduced in Sec. 3.8.1. This section describes the algorithms used to implement automatic gain and offset calibration of input channels.

Auto-ranging signal conditioning was intended to free the user from specifying and calibrating gain/offset amplifiers. This mode of SPC operation was identified as an essential component of the user-friendly interface, where input channels could be automatically set to optimise the signal-to-noise ratio of measurements.

AUTOMD (Fig. 3.20) is the main routine used for gain/offset control of input channels. The *autost* flag determines the type of signal conditioning to be applied. The auto-ranging algorithm is best described with reference to the examples shown in Fig. 3.22.

The viewport shown in Fig. 3.22 defines the dynamic range of the ADC. Optimal gain settings are assumed when the signal extrema occupy the whole viewport without overflowing the ADC (Fig. 3.22b). The auto-ranging algorithm has the objective function of driving *incgn* to zero, which is used as a measure of optimal signal conditioning. The operations performed by the algorithm can be summarised with the following rules:

- **search for extrema**: identifying the extrema that is closest to saturation,
- **if both extrema are saturated**: halve the gain and repeat previous check,
- **else**: apply offset to signal to centralise it within viewport,
- **compute gain word**: to increase the signal amplitude range.

The flowcharts of Fig. 3.20 and Fig. 3.21 provide a more detailed listing of the above rules. Each pass through AUTOMD leads to gain/offset calculation based on the value of *incgn*.

In cases where saturation has occurred (Fig. 3.22a and c), the signal must be offset by an amount *incgn* from the saturated end. The flowcharts define *isat*, a variable whose sign determines the offset direction. Using the above rules, AUTOMD enters into an iterative procedure whereby gain and offset words are updated to the SAC until the optimal signal range is obtained.

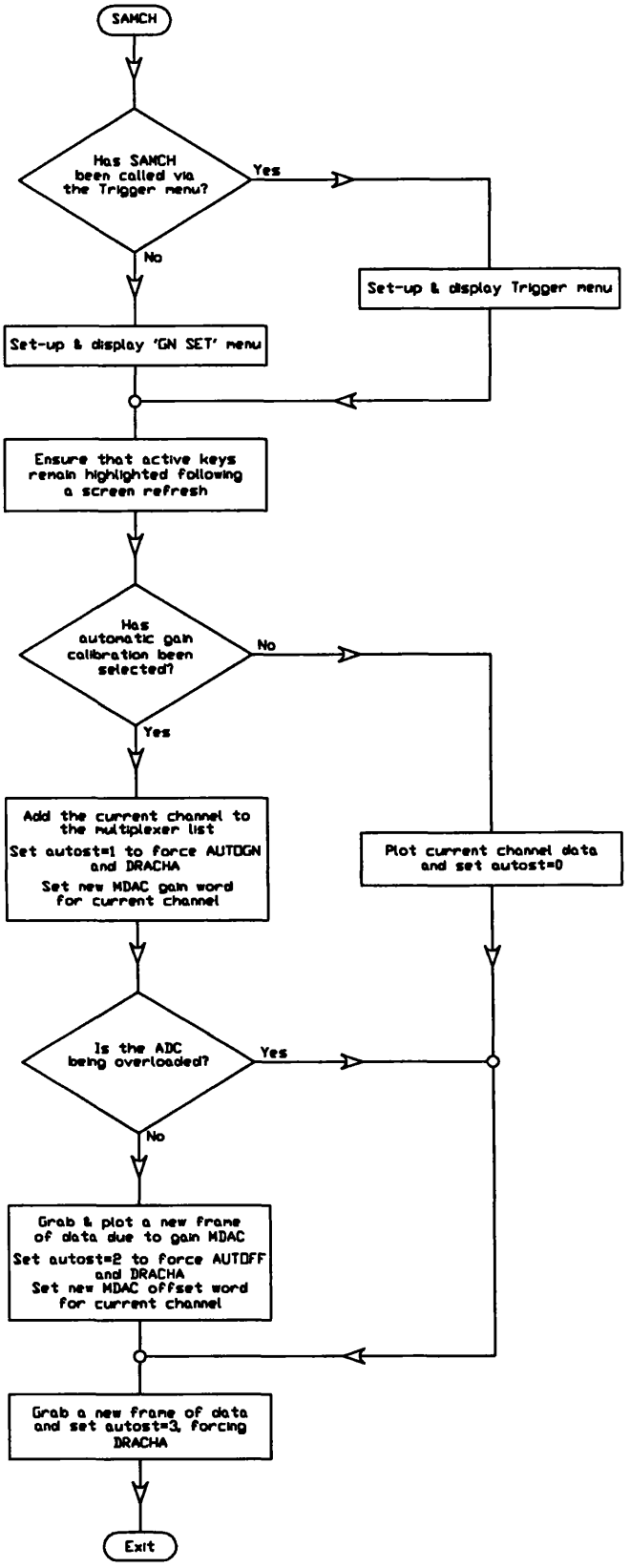


Fig. 3.23 SAMCH coordinates auto-ranging and updates the SAC amplifiers with the new gain/offset words.

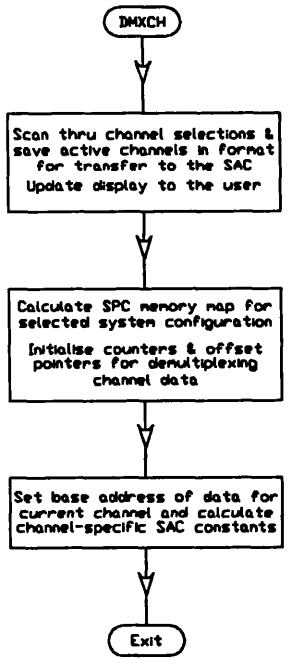


Fig. 3-24 DMXCH is used to demultiplex input signals from data memory.

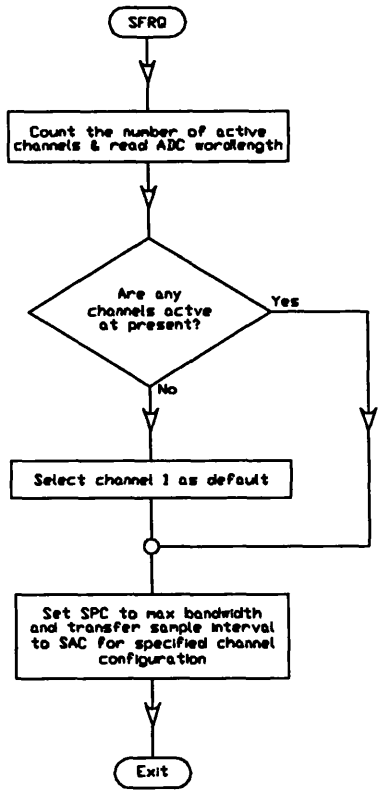


Fig. 3.25 SFRQ automatically sets the sampling frequency of the SAC.

The user activates auto-ranging gain/offset adjustment by selecting the AUTO button via the GN SET menu. Auto-ranging an input channel automatically activates it for data-acquisition during the main analysis by programming the analogue multiplexer. SAMCH (Fig. 3.23) is the main routine used during auto-ranging.

B. Digitally Multiplexed Data Storage

The input signals to the SAC are digitally multiplexed through the signal conditioning unit and into the ADC. Digitised data samples from the ADC are then stored into the DPM buffer ready for transfer to the host. The task of demultiplexing the DPM data must be handled by the API. DMXCH (Fig. 3.24) is the routine for determining the base address of the first sample in memory belonging to a given input channel. This routine provides automatic memory boundary alignment depending on the selected ADC wordlength and also provides DPM overflow management.

The data-acquisition sample buffer is limited by the DPM to a maximum of 16 KB. However, the buffer length is a SAP variable which can be reduced to support single frame data capture. For example, this is advantageous during gain calibration or trigger template selection where fast screen updates are required.

The demultiplexing scheme minimises the CPU overhead normally associated with sorting algorithms. Once the base address pointer has been computed using DMXCH the input samples are accessed by auto-incrementing the pointer with a fixed offset. This dynamic data management scheme provides the maximum sample memory for any given channel configuration.

C. Sampling Frequency Calculation

The sampling frequency of the SAC is automatically selected for a given bandwidth. SFRQ (Fig. 3.25) is the routine used to set the sampling frequency based on the number of active channels and the required ADC resolution. Other aspects of sampling are described in Sec. 2.2 and Sec. 3.8.

The minimum sampling rate is determined by the Nyquist sampling theorem. However, there are certain advantages to sampling slightly higher than the Nyquist rate. For example, the SPC sampling frequency is calculated as:

$$f_s = 2.56 \times f_{max}$$

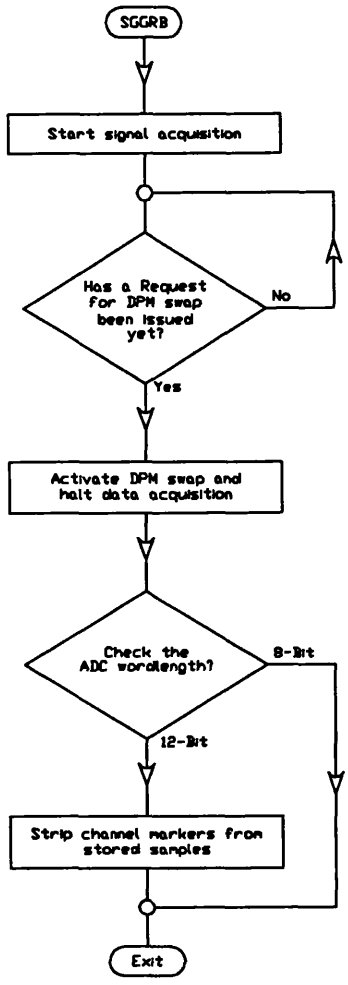


Fig. 3.26 SGGRB is used during real-time data-acquisition.

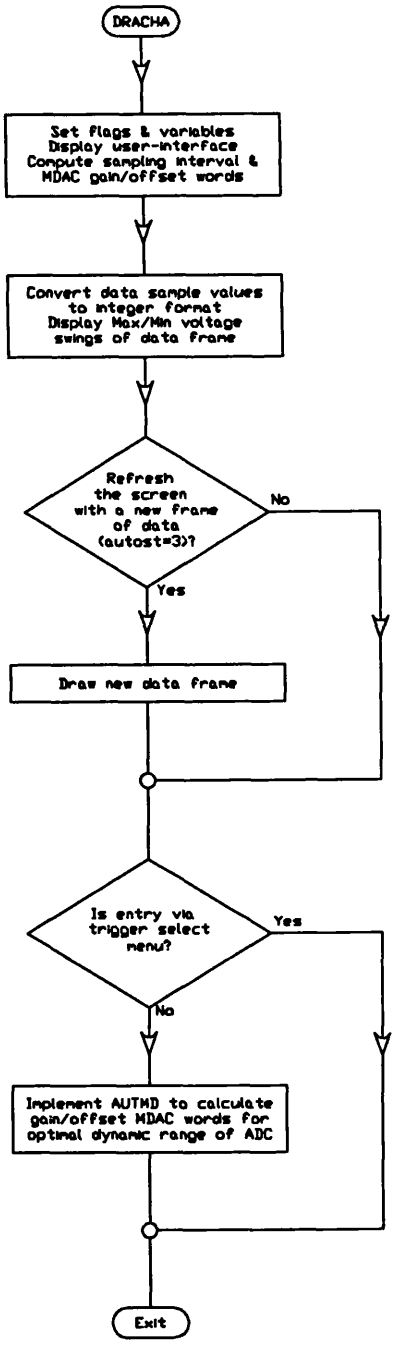


Fig. 3.27 DRACHA manages the graphic user-interface and display of signals.

with frequency resolution $\Delta f = f_s/N$, where N represents the record-length. Assuming N=256, $\Delta f = f_{max}/100$ and if 100 lines are to be displayed on the screen, the maximum signal frequency is automatically f_{max}. The SPC implements this method of calculating the sampling frequency since the full signal bandwidth is automatically available for display.

D. Real-Time Data-Acquisition

Real-time data-acquisition is controlled using the subroutine SGGRB (Fig. 3.26). Using the DPM (Sec. 3.7) and the host-to-SAP communications protocol (Sec. 3.10.4), the SPC is capable of performing real-time data-acquisition on four channels over a maximum bandwidth of 1 KHz.

DRACHA is the routine responsible for drawing signals to the screen and managing the user-interface (Fig. 3.27). This includes conversion of ADC levels into voltages. Since each input channel will have its individual gain and offset settings, DRACHA must ensure that the correct scaling factors are used during calculations.

3.12.2 Coherent Time-Domain Averaging

The SPC uses a coherent time-domain averaging technique for enhancing the signal-to-noise ratio of measurements. Many physiological signals - including ECG, blood pressure and flow, intra-thoracic airways pressure, etc. - are repetitive in nature and often contaminated with uncorrelated random noise. Based on this *a-priori* knowledge, a coherent averaging algorithm was devised for performing multi-channel signal averaging. The performance of this approach is optimal if the noise is uncorrelated, stationary and additive (Rompelman and Ros, 1986a). Other factors require the signal to be time-invariant and the elapsed time from the stimulus (or reference point) to the waveform of interest remains constant (Rompelman and Ros, 1986b).

Coherent averaging consists of the summation of successive repetitions of a signal in such a way that the signal reinforces itself, while the incoherent noise component tends to cancel out. The averaging process must firstly be triggered from before the signal averages can be computed.

For example, consider the case where the SPC may be used for determining hydraulic impedances in the cardiovascular system. The signals of interest include the ECG, blood pressure and blood flow. All three signals are repetitive and are time-locked to each other. In this case, the ECG is used as the trigger event to synchronise the channels. The trigger detection algorithm must be trained to *fire* on detection of a user-defined trigger template (e.g. the 'R-wave' of an ECG complex). Since all input channels are assumed to be time-locked with each other, the trigger point provides temporal alignment between channels so that the average of each channel can then be computed.

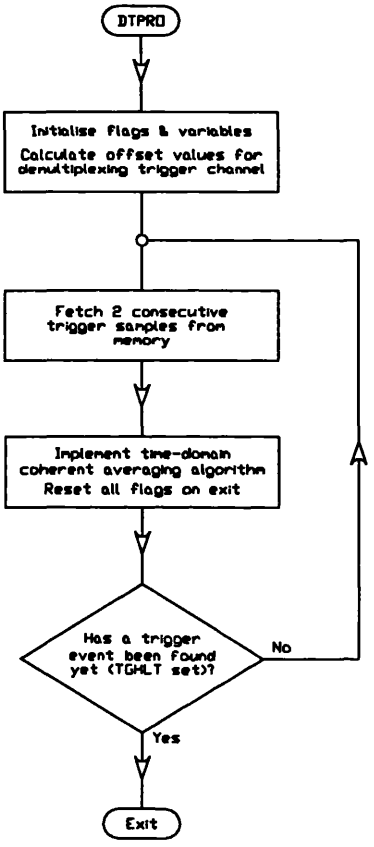


Fig. 3.28 The trigger protocol handler (DTPRO).

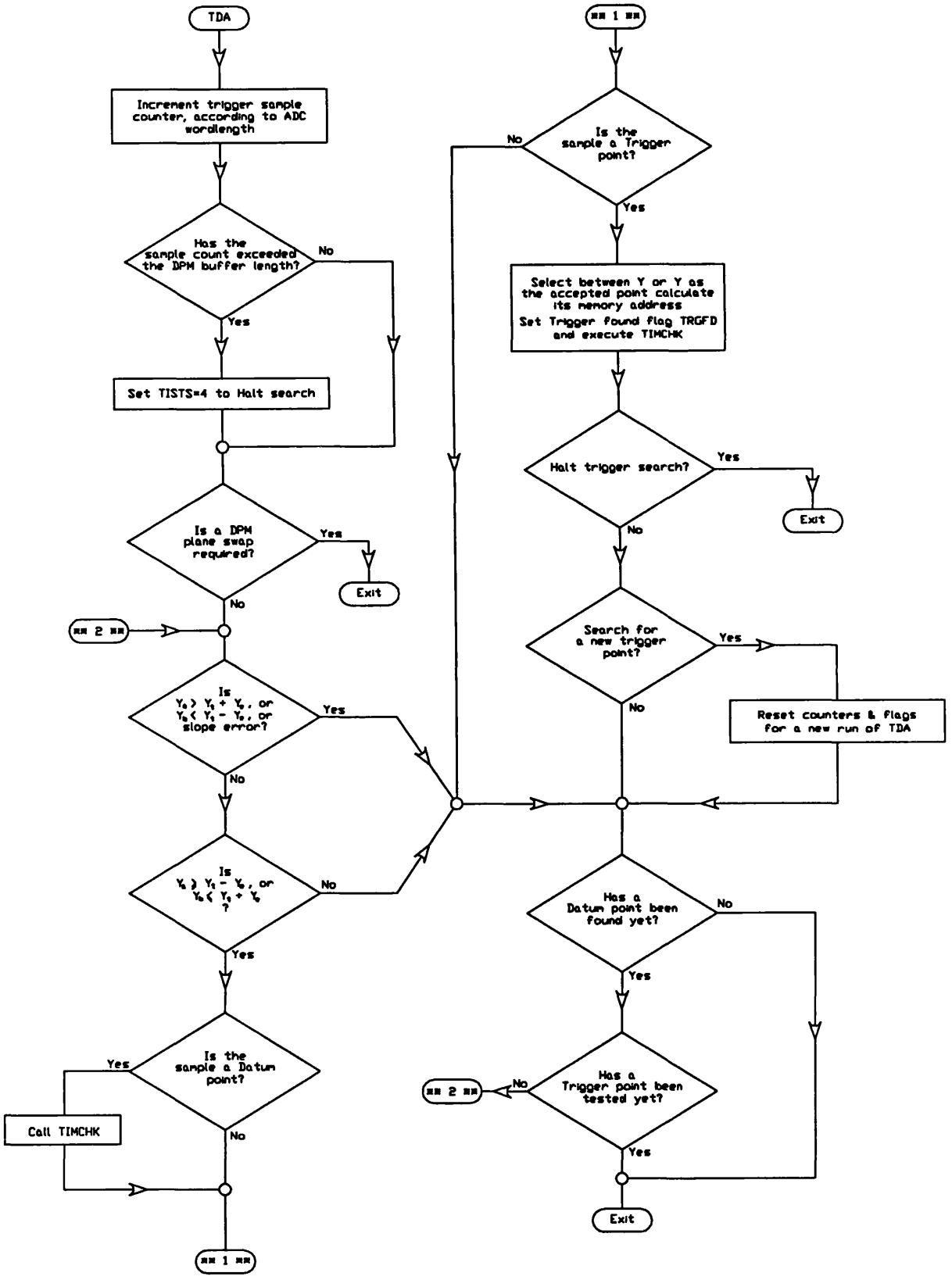


Fig. 3.29 The threshold detection algorithm (TDA).

The following sections describe a flexible algorithm for real-time coherent signal averaging. The algorithm was implemented in assembly language and exists as an independent sub-module to the API. The averaging scheme was successfully tested for real-time performance, using all four input channels of the SAC, with a signal of up to 1 KHz bandwidth.

A. Trigger Template Definition

The main component of the averaging scheme is the trigger algorithm which uses the concept of a *trigger template* to provide a trigger mechanism which is accurate, robust and easy-to-specify. This approach requires the user to specify a set of trigger conditions from a sample frame of the trigger channel (Sec. 3.12.3E). DTPRO is the trigger protocol handler which is activated to perform two main tasks (Fig. 3.28):

- interaction with the user to specify a trigger template;
- searching for an input trigger sequence that is a close match to the trigger template.

The protocol handler also performs memory allocation for trigger samples, transfer of sample points to the trigger search algorithm and management of trigger status flags. DTPRO provides a simple interface between the trigger algorithm and the API. Once DTPRO satisfies a trigger condition, the coherent averager is activated to perform averaging on the active data channels.

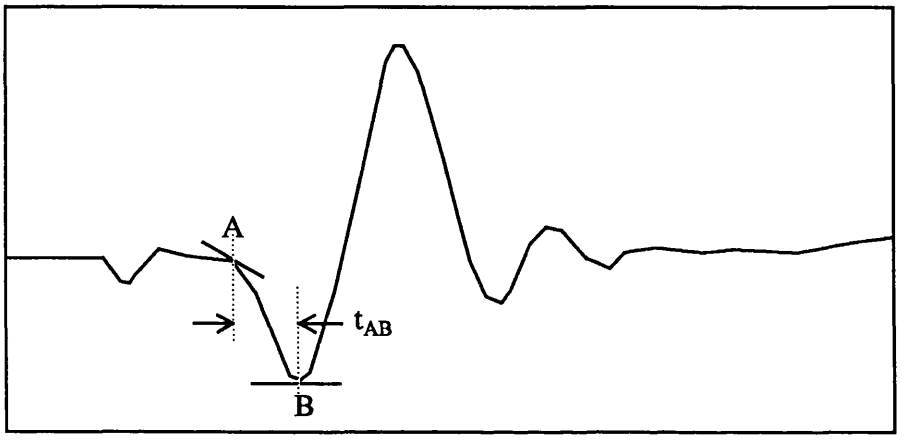
A *trigger event* is defined with reference to Fig. 3.30a in the following way:

- two threshold voltages must be selected from the trigger template. These points are defined as the datum (A) and trigger (B) points, where $t_A < t_B$;
- the slope of the tangent at each threshold point must be defined as being positive, negative, or zero (corresponding to a turning point);
- the time interval between points A and B (t_{AB}) is calculated by the trigger algorithm. The datum (A) represents the *fiducial point* from which timings are measured and the trigger (B) should be selected to coincide with a significant point on the trigger template.

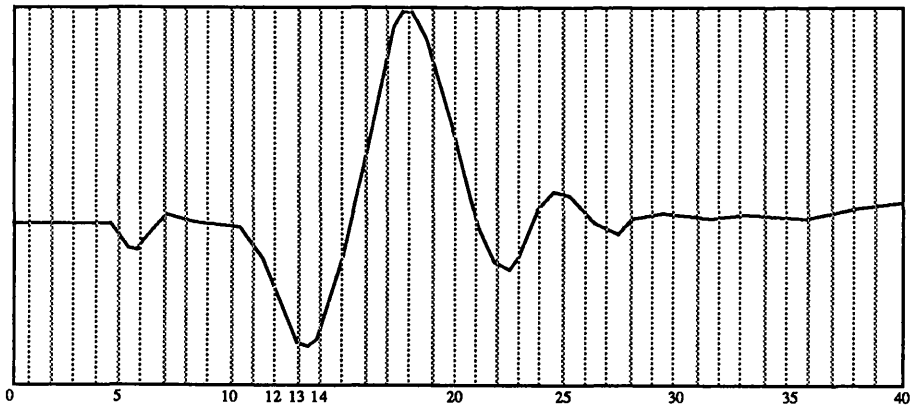
The next section describes the trigger search algorithm in greater detail.

B. Threshold Detection Algorithm

The trigger search is implemented by the *threshold detection algorithm* (TDA) defined in Fig. 3.29. The TDA operates on data from the trigger channel, searching for datum and trigger points that match the definitions in the trigger template. When a trigger search is



(a) Trigger template definition.



(b) Digitised trigger template of (a).

Fig. 3.30 Trigger selection criteria.

activated, the algorithm will initially search for a datum point. This follows from the rule that the datum must precede the trigger ($t_A<t_B$) and provides greater efficiency by eliminating the extra processing required for detecting *false* trigger points.

The sequence of tests performed by the TDA routine in validating a trigger event are:

- slope detection; this is a simple test that provides large data reduction with minimal computational effort;
- threshold detection; this is a search for the datum and trigger point amplitudes defined in the trigger template;
- application of the trigger timing criterion (Sec. 3.12.2C).

These tests are performed on a pair of sequential trigger samples. For example, consider the trigger signal shown in Fig. 3.30 with datum-point A (negative slope) and trigger-point B (zero slope). Let us assume that the TDA has found the datum (A) at x_{10} and is now searching for the trigger (B) at x_{13}. The algorithm will reject the samples x_{11} and x_{12} as potential trigger points since they fail on the *slope-test*. Sample pair x_{13} and x_{14} will pass the slope test and will then be tested for threshold levels. To compensate for the effects of trigger jitter due to sampling errors (Rompelman and Ros, 1986b), the threshold test selects the sample whose threshold level is closest to that defined in the trigger template. In this case, x_{13} is preliminarily selected as the trigger point. The final test checks to see if t_{AB} (Fig. 3.30a) matches with the value calculated from the trigger template. If all the tests are successful, then x_{10} is selected as the fiducial point of averaging. Other scenarios can be simulated and tested using the algorithms described here.

The above example describes the simplest possible outcome. In reality, once a datum has been found, two possibilities can arise:

- the search immediately locates a trigger point (as above), in which case the trigger timing criterion of Sec. 3.12.2C is applied; or
- the search locates another *apparent* datum point.

The latter situation is most likely to occur in the presence of overwhelming background noise. The TDA must determine which datum point to select. This is done by perform the trigger timing test on all apparent datum points. The datum point producing the closest match to the template is then accepted as the true fiducial point.

A circular buffer is implemented to keep track of all datum points found during the search for a trigger point. The buffer can store up to 40 datum points. If this maximum is exceeded, the buffer wraps-around to overwrite the oldest datum point. Since the circular buffer operates on a Last-In First-Out (LIFO) basis, where the top of the queue represents the most recent datum to be found. UPIDT is the routine used to manage the circular buffer

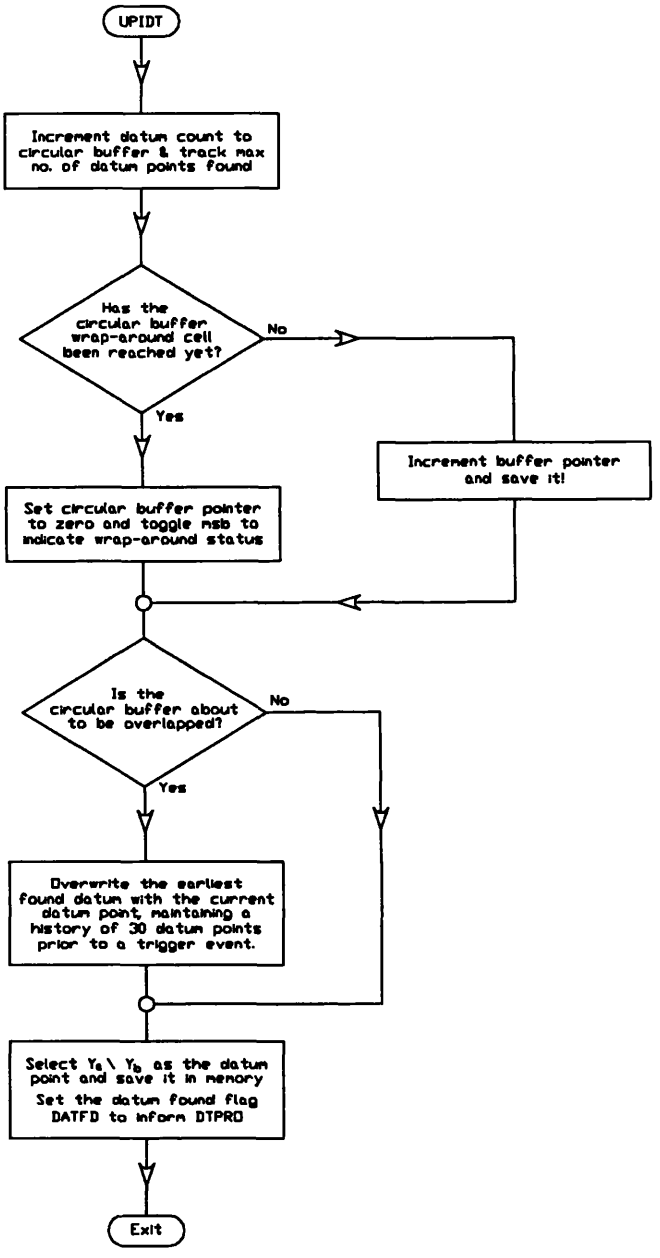


Fig. 3.31 UPIDT keeps track of all possible datum points using a circular buffer.

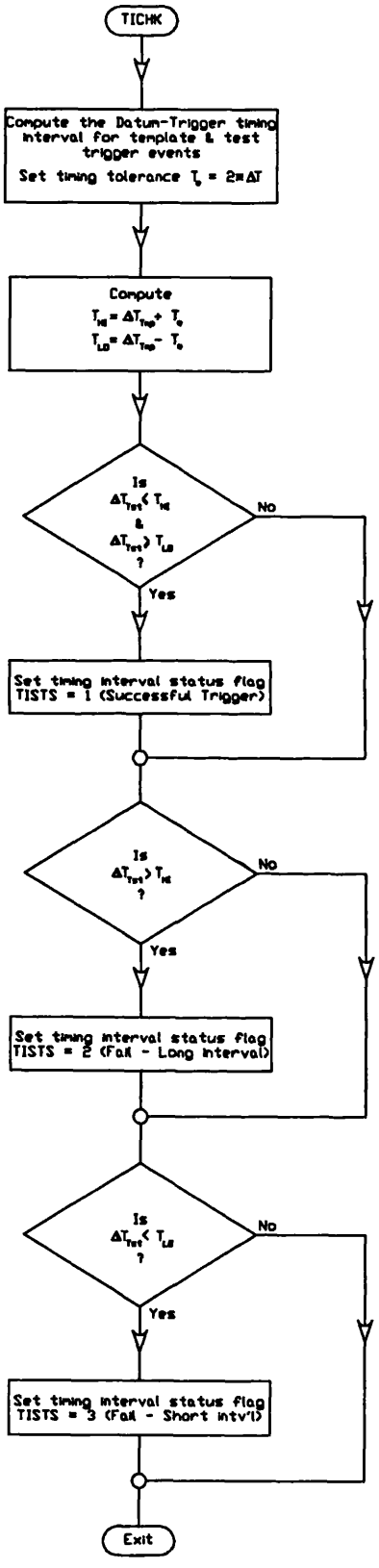


Fig. 3.32 TICHK performs the trigger timing criteria tests of Sec. 3.12.2C.

(Fig. 3.31). Once a trigger point is found, each datum point is recalled from the queue and the trigger timing criteria of Sec. 3.12.2C applied to determine the true fiducial point. A successful trigger event is followed by the averaging operation.

C. Trigger Timing Criteria

The trigger timings defined in Sec. 3.12.2A is handled via the time interval checking routine TICHK (Fig. 3.32). This test determines whether or not a datum point is accepted as a trigger event. The time interval between datum and trigger points (t_{AB}), as defined in the trigger template, is used as a quantitative measure for assessing the integrity of a trigger selection. The algorithm includes a *timing uncertainty factor* to compensate for any errors introduced as a result of the sampling process. The uncertainty factor is expressed in terms of the sampling interval, i.e. a tolerance of $2/f_s$. For example, point A (Fig. 3.30a) will be accepted as a trigger event provided that $-2/f_s \leq (t_{AB} - t_{TEMPL}) \leq +2/f_s$.

The TDA routine calls TIMCHK (Fig. 3.33) to test the trigger timing criteria. The timing data associated with the datum and trigger points are passed onto TIMCHK, which then carries out the test. The timing interval status flag TISTS returns a value indicating the outcome of the test. The TISTS flag has three possible outcomes, and these are listed below:

a) if the test is successful, TISUC will be executed (Fig. 3.34a). The search for a trigger event is halted by setting the TGHLT flag and the address of the datum point is passed back DTPRO so that averaging can be done.

b) if the test has failed as a result of the timing interval being shorter than the template interval ($t_{AB} - t_{TEMPL} < -2/f_s$, TISHT will be executed (Fig. 3.34b). If further datum points are currently available in the circular buffer, the flag TSRPT is set, and the timing tests repeated with the next datum in the queue. However, if no further datum points exist the flag NWTRG is initiate a new trigger point search. The datum points currently in the circular buffer will not be purged. This is used to reject trigger point artefact.

c) if the test has failed as a result of the timing interval being longer than the template interval ($t_{AB} - t_{TEMPL} > +2/f_s$, then TILNG will be executed (Fig. 3.34c). If several datum points exist, any datum points in the circular buffer failing due to (b) will be retained in case they satisfy future trigger timing criteria and the remaining buffer contents are purged.

The above summarises all possible outcomes of the trigger timing test and provides the necessary memory management of useful datum points.

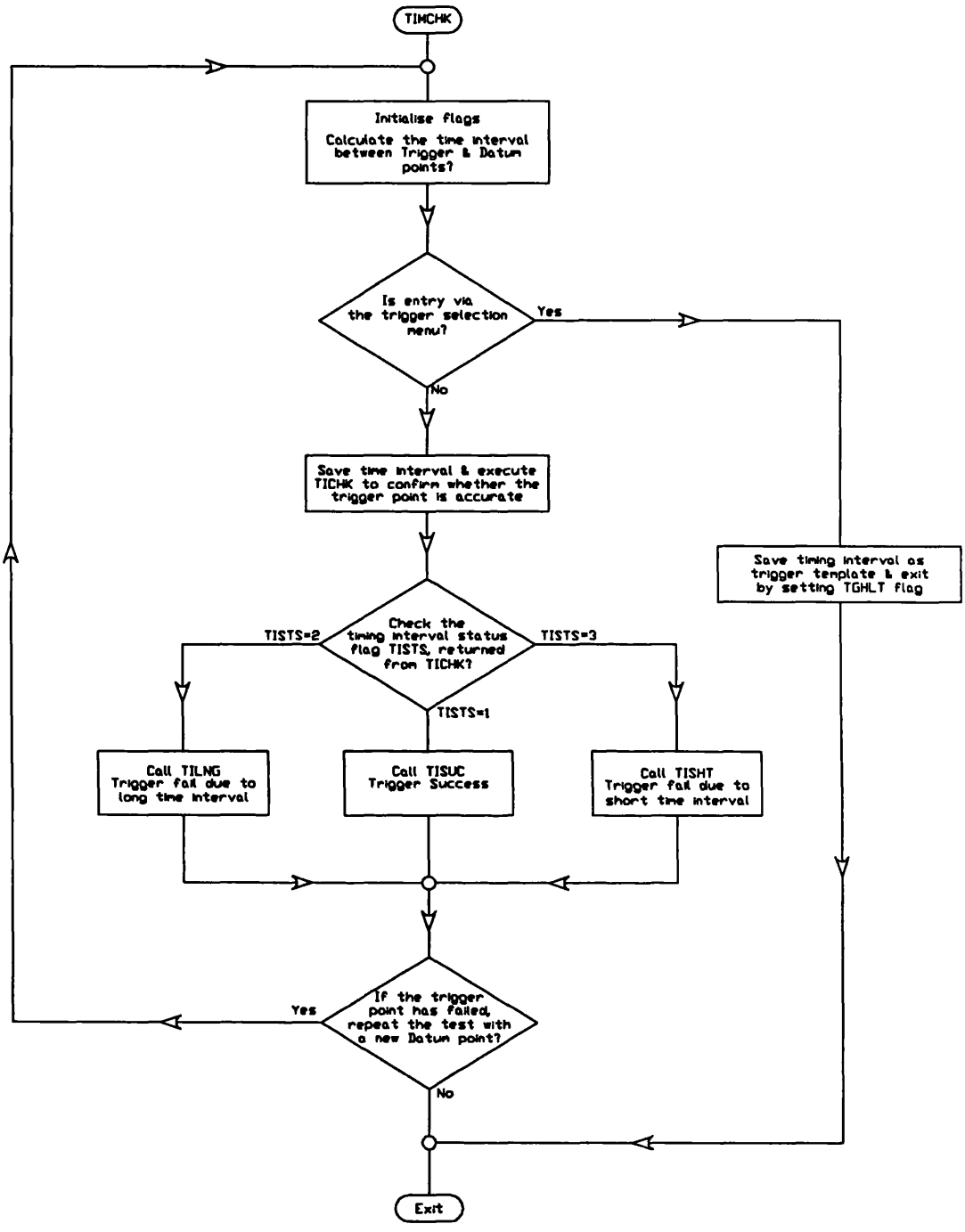
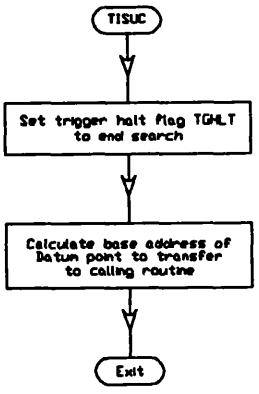
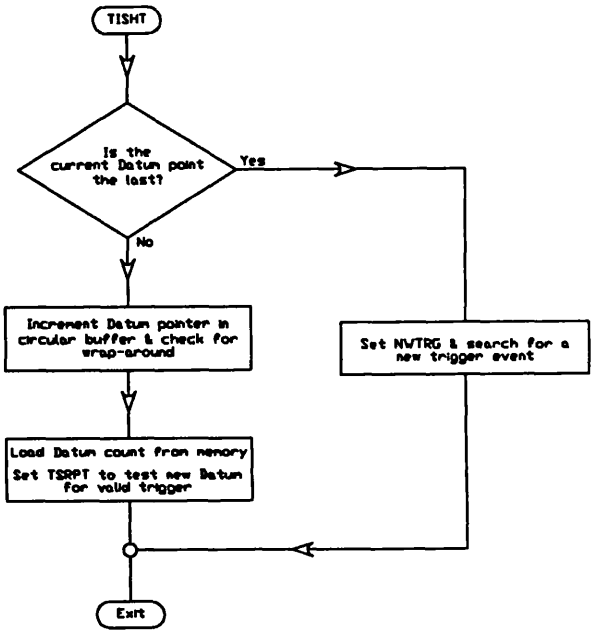


Fig. 3.33 TIMCHK executes a function (Fig. 3.34) based on the status flag TISTS returned by TICHK.

(a) TISUC



(b) TISHT



(c) TILNG

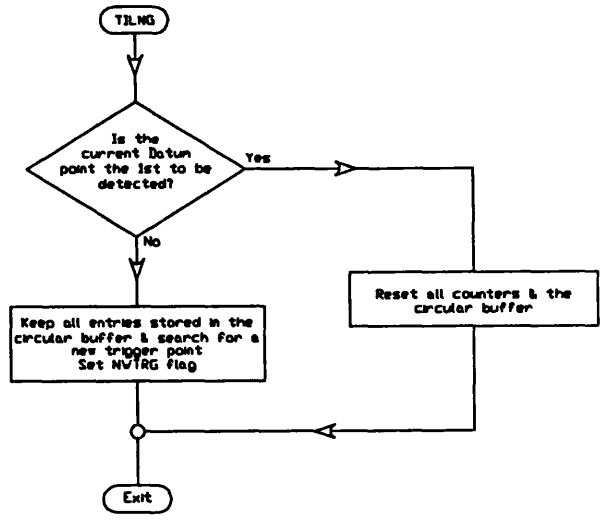


Fig. 3.34 The functions available to TIMCHK based on the TISTS flag.

3.12.3 Measurement and Control of Dynamical Systems

The following section describes the user-interface associated with the API mentioned in Sec. 3.12.2. Plate 3.2 shows the graphic user interface associated with the schematic menu system of Fig. 3.2. In this example, the tree is seen to be three levels deep. The main level consists of the menu that allows access to the six major functional modules: annotation, display/data manipulation and storage/retrieval of data and setting-up of measurement, SAP and trigger parameters.

A. Menu Definitions

The root menu is the top-level user-interface which defines the main SPC functions (Plate 3.2a). Selections made at this level will spawn a sub-menu containing further options.

Due to the limited RAM on the host, certain menus had to be implemented using a memory overlay scheme. These specifically include the LABEL and ARCIVE functions. An overlay manager is used to maintain the SPC context during overlay functions.

Two softkeys are reserved specifically for calling HELP information and exiting from menu selections. In some cases, however, the help softkey is used for controlling other functions. This was necessary due to the limited number of softkeys available. The following sections describe menu functions in greater detail.

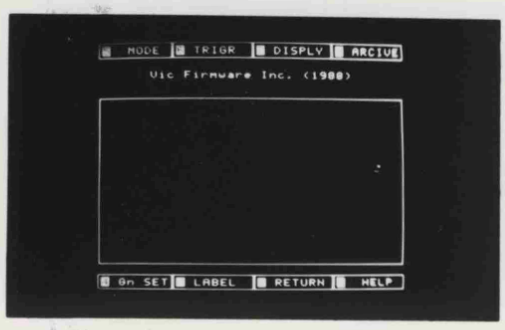
B. Label Menu

The LABEL menu (Plate 3.2b) provides the user with a simple text editor for labelling analogue input channels. Four arrow keys are provided for scrolling a text cursor over a graphical keypad. The SELECT key is used to accept a character into the label buffer. To select a new channel for labelling, the user must scroll the cursor off the top or bottom line of the keypad. The cursor keys can then be used to select the next channel to be labelled.

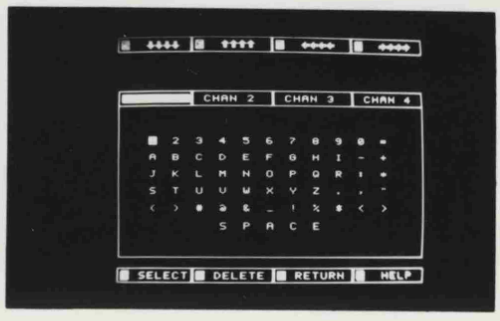
The channel labels are displayed as a title whenever the time or power spectrum of a channel is displayed.

C. Gain Calibration

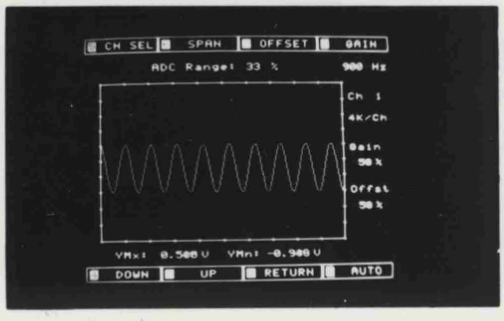
This menu (Plate 3.2c-d) must be activated at least once during the course of an experiment to specify the gain/offset settings to be applied to input channels. Although the SAC is preset with default signal acquisition parameters (Sec. 3.11.1), the host will take the safety measure of ensuring that the gain calibration menu is entered at least once during the course of an experiment, so that the input channels are correctly specified.



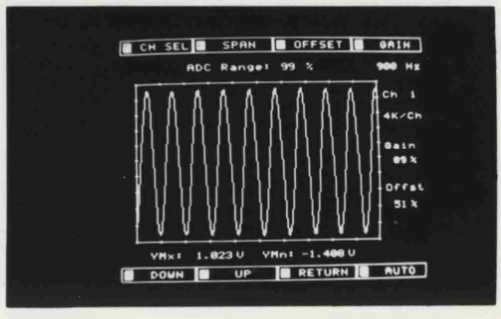
(a) Root-level menu.



(b) The LABEL menu used for annotating input channels.



(c) The GN SET menu used for auto-calibrating the input channels.



(d) The result of applying auto-ranging to (c) above.

Plate 3.2 The menu formats (a-d) associated with the application program interface described in Sec. 3.12.

The AUTO key initiates auto-ranging gain calibration on the selected channel. In this mode, the SPC enters into a recursive gain/offset calibration phase using the algorithms described in Sec. 3.12.1. The CH SEL key is used to select a channel for calibration.

Using keys 3,4,6 and 7, the user may manually fine-tune control the gain/offset level of a given channel. The SPAN key selects the bandwidth of measurement signals thereby controlling the SAC sampling frequency (Sec. 3.12.1C).

From Plate 3.2c-d we can see how auto-ranging has optimised the dynamic range of the signal from 33% (uncalibrated) to almost full-scale ADC range. This was achieved after a few frames of iteration.

D. SAP Mode Options

This menu provides control over SAC hardware characteristics. Three options are currently supported, including:

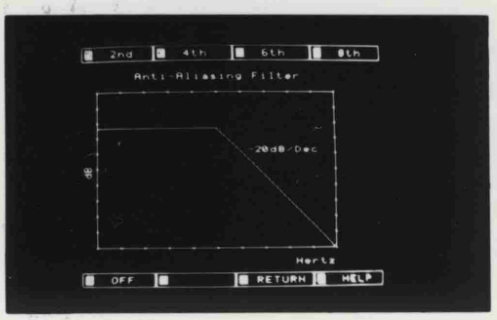
- anti-alias filter selection (Sec. 3.8.3). The FILTER key spawns a sub-menu (Plate 3.2e) which provides options for controlling the stopband attenuation of the AAF module. The filter roll-off is graphically displayed to the user.
- switching the ADC resolution between 12- and 8-bit words (Sec. 3.8.2);
- software reset control over the SAP (Sec. 3.6.2).

E. Trigger Template Definition

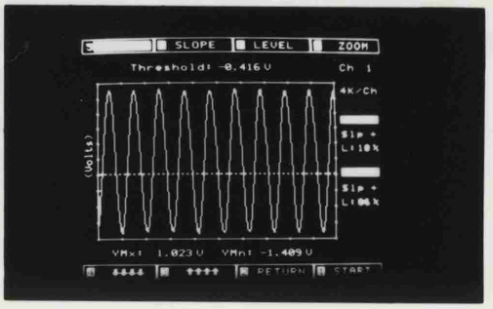
The trigger menu (Plate 3.2f) is used to specify a trigger template for use by the coherent averaging algorithm described in Sec. 3.12.2. Channel 1 is the designated trigger source. The START key is used to capture and display *potential* trigger templates to the user. A trigger template is accepted by selecting key 1 to halt trigger capture.

Once a template has been selected, keys 3 and 4 may be used to scroll a horizontal marker to the desired datum and trigger points (Sec. 3.12.2B). The SLOPE key is used to select the slope of a tangent at each of these points. The status area shows the current amplitude and slope values for datum and trigger points.

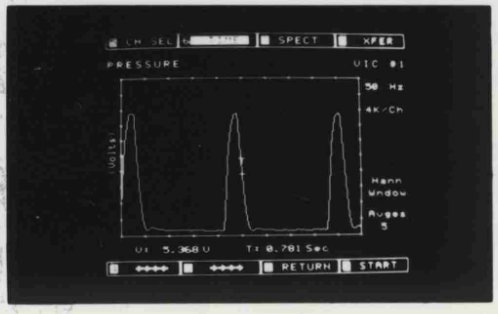
A real-time zoom facility provides a method of improving the resolution of the trigger template. By changing the sampling rate on the trigger source, the ZOOM key can be used to focus on a particular trigger pattern by expanding (or compressing) the trigger signal in time.



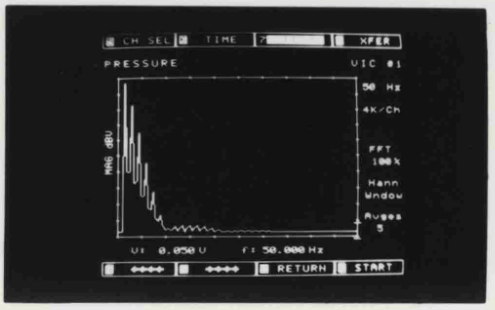
(e) Anti-alias filter selection and specification menu.



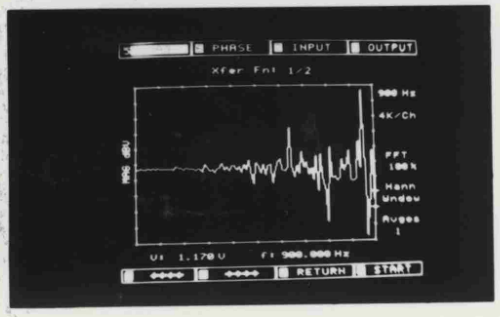
(f) The trigger template specification menu used for coherent averaging.



(g) The DISPLAY menu with real respiratory pressure input.



(h) The power spectrum plot of (g).



(i) The XFER menu used for performing transfer function calculations.

Plate 3.2 The menu formats (e-i) associated with the application program interface described in Sec. 3.12.

F. Display Channel Data

The DISPLAY key provides the measurements and signal processing data options. This menu (Plate 3.2g) can only be selected if the GNSET and TRIGR functions have been set. The user can START performing measurements at any time. All defined input signals will be automatically averaged (Sec. 3.12.2) in real-time and the results will be plotted to the screen once acquisition is halted.

The DISPLAY key can also be used to manipulate archived data selected via the ARCHIVE key. In this mode, the SPC must restore the system state to that during which the measurements were made to allow post-processing of measurements.

All signal processing functions, such as power spectra of inputs or transfer function relationships between inputs, are computation intensive operations that typically require an FFT calculation per channel. Since the SPC takes approximately 45 seconds to perform a 256-point FFT, this type of analysis must be performed off-line (Plate 3.2h).

When a signal is displayed on the screen, a cursor can be used to trace through points on the signal whilst their x and y coordinates are displayed. Using keys 5,6 and 7 of this menu, the user can view the time or power spectrum display of any input channel.

If two or more channels are active, the XFER menu (Plate 3.2i) can be used to perform transfer function measurements between any pair of inputs. Keys 7/8 are used to specify the input/output relationship between channels. Keys 5/6 can then be used to view the resulting transfer function magnitude and phase plots. For increased accuracy, the transfer function calculations compensate for phase errors between channels introduced by the measurement system.

3.13 Discussion

This chapter has presented the design concepts of a flexible and user-friendly computer system known as the *signal processing computer* (SPC). The paper by Hailstone et al. (1986, Appendix 1) lists the team of people who have contributed to the SPC project. The personal contribution of the author has been to take the SPC from a simulation tool (Sehmi, 1988; Kabay, 1985) to a fully functional multi-processing computer system. The major contribution has been in the development of the SPC signal acquisition processor (SAP), host-to-SAP communications interface, signal acquisition card and programmable anti-aliasing filters. These units were integrated with the host processor and dual plane memory (DPM) to provide a flexible and extendible instrumentation system.

The SPC philosophy was based on low technology devices arranged to provide a novel interface to standard hardware under software control. For example:

- the DPM uses standard logic devices to achieve a data transfer rate far superior to direct memory access (DMA) techniques using complex controller devices;
- the SAC provides a computer-interfaced analogue signal conditioning system to overcome many of the design challenges faced by analogue design engineers;
- a simple communications protocol for transferring control and status messages between the host and SAP;
- software algorithms for controlling real-time data-acquisition and coherent averaging;
- software to provide a system solution to some of the general problems of measurement recording and signal processing.

Since its initial conception, the hardware and software used to implement the SPC has been superseded with new development tools that offer greater functionality, higher bandwidth and faster development times - all at lower cost. Using personal computers with commercially-available add-on cards (e.g. for data-acquisition, digital signal processing and graphics output) and high-level software development tools, one can achieve the SPC design philosophy of a real-time instrument for data-acquisition and signal processing in specific problem domains.

The SPC system now fulfils the original specification set out by Hailstone et al. (1986) and can be used for data-acquisition and signal processing in several application areas. However, by current day standards, it suffers from several disadvantages:

- the application-domain is limited to the analysis of periodic deterministic signals, since the coherent time-domain averaging technique is used;
- poor CPU performance during computation intensive operations makes real-time spectral-analysis impossible;
- a slow graphics display system means that real-time screen updating of signals is difficult to achieve;
- limited memory and slow storage devices restricts the overall complexity of application programs.

These disadvantages are due to bandwidth limitations of the SPC hardware. In Chapter 5, the author will show how these limitations can be overcome by using modern, cost-effective technology. It is shown how the overall design philosophy of the SPC has been adapted to an IBM-PC based system for real-time measurement and modelling of high-frequency jet ventilation in anaesthesia. The system uses real-time signal processing hardware and software algorithms implemented around a flexible generic kernel to assist in identifying the transfer function relationships between respiratory signals.

3.14 References

Hailstone, J.G., Jones, N.B., Parekh, A., Sehmi, A.S., Watson, J.D. and Kabay, S. (1986): "Smart instrument for flexible digital signal processing", *Med. & Biol. Eng. & Comput.*, **24**, 301-304.

Kabay, S. (1985): "A diagnostic computer system and algorithms for biomedical and engineering applications", *Internal Report*, University of Leicester.

Lago, P.J.A. and Jones, N.B. (1983): "Turning points spectral analysis of the interference myoelectric activity", *Med. & Biol. Eng. & Comput.*, **21**, 333-342.

Parekh, A.K. (1986): "*Computer analysis of EMG turning ponts process*", PhD Thesis, University of Sussex.

Rompelman, O. and Ros, H.H. (1986a): "Coherent averaging technique: a tutorial review - Part 1: Noise reduction and the equivalent filter", *J. Biomed. Eng.*, **8**, 24-29.

Rompelman, O. and Ros, H.H. (1986b): "Coherent averaging technique: a tutorial review - Part 2: Trigger jitter, overlapping responses and non-periodic stimulation", *J. Biomed. Eng.*, **8**, 30-35.

Sehmi, A.S. (1988): "*New environments for neurophysiological investigations*", PhD Thesis, University of Leicester.

Chapter 4
Mechanical and Physical Characteristics of the Respiratory System

4.1 Introduction

This thesis is concerned with the development of environments for real-time measurement and control of dynamical systems. The application area selected for this study was high-frequency jet ventilation (HFJV) of patients receiving intensive-therapy. This chapter aims to describe the anatomy and physiology of the respiratory system. Particular attention is given to the morphometry of the tracheobronchial tree since this has important consequences on the transport and distribution of inspiratory gases - and hence on respiratory function. Consideration is given to the physiological aspects of HFJV and its effects on pulmonary mechanics and the mechanisms by which gas transport is maintained. Acoustic modelling of the respiratory system is presented to clarify the physiology and mechanics of ventilation during HFJV. Finally, existing methods of respiratory impedance measurement are compared with a new measurement technique developed by the author.

4.2 Structure and Function of the Respiratory Airways

4.2.1 Anatomy of the Tracheobronchial Tree

The tracheobronchial tree comprises a complex series of branching tubes which decrease in radius and length as we move from the trachea towards the lungs. Moving distally from the trachea, the total cross-sectional area is seen to increase rapidly due to the logarithmic increase in the number of branches and their less rapid decrease in diameter. The respiratory tree exhibits a high degree of geometrical organisation covering several orders

of magnitude in size from the trachea to the alveolar sacs. These structural characteristics are important to respiratory function with the consequence that disorders in airway structure compromise the viability of the subject.

Most models of the respiratory system are an oversimplistic approximation which do not give sufficient insight into the performance of the actual system. Weibel's (1963) model of the bronchial tree uses a hierarchical classification scheme for identifying successive generations of airways from the trachea (generation 0) to the alveolar sacs (generation 23). Fig. 4.1 shows how the trachea (generation 0) bifurcates asymmetrically into the main, lobar and segmental bronchi (generations 1-4) respectively. Following this are the

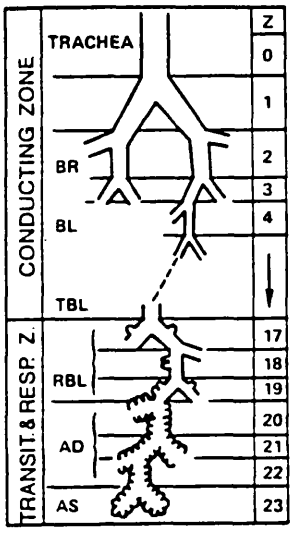


Fig. 4.1 The bronchial airways according to Weibel (1963).

small bronchi (generations 5-11) which extend through seven generations. Up to this point the air passages maintain patency through the cartilage within their walls. However, from the eleventh generation onwards the airways are embedded into the lung parenchyma and the cartilage disappears. The patency is now provided by the elastic recoil of the parenchyma, which holds the airways open at a diameter that is dependent on the lung volume. Since the rate of increase in the number of bronchioles (generations 12-16) is far greater than the decrease in calibre of its branches, we see a large increase in the total cross-sectional area. Down to the smallest bronchioles the airways have served in conduction and humidification of gases. In the three generations of respiratory bronchioles (generations 17-19) there is a gradual transition from conduction to gas exchange via the alveoli in their walls. The terminal air passages include the alveolar ducts (generations 20-22) and alveolar sacs (generation 23). The main function of the respiratory airways is to transport oxygen from the external air to the blood, and carbon dioxide in the reverse direction.

Table 4.1 outlines the main characteristics of Weibel's symmetric model. It can be seen that the total cross-sectional area of the respiratory tract is minimal up to the third generation. However, the area at the terminal bronchioles approaches about 30 times that at the level of the large bronchi. Parker et al. (1971), using an asymmetrically dichotomous branching model of the alveolar ducts, have estimated the number of distal respiratory bronchioles at $225x10^3$ each of which give rise to about 100 ducts and sacs. The alveoli total $300x10^6$ in number, with an average of between 10 and 16 alveoli/duct or sac.

Airway	*Generation Number*	*Diameter (mm)*	*Number of Branches*
Trachea	0	18	1
Main bronchi	1	13	2
Lobar bronchi	2 → 3	7 → 5	4→ 8
Segmental bronchi	4	4	16
Small bronchi	5 → 11	31	32 → 2000
Bronchioles	12 → 16	10.5	4000 → 65k
Respiratory bronchs	17 → 19	0.5	130k → 500k
Alveolar ducts	20 → 22	0.3	1m → 4m
Alveolar sacs	23	0.3	8m

Table 4.1 Characteristics of bronchial airways (Weibel, 1963).

The terminal bronchioles are the most distal structures in the airways not to bear alveoli. They give rise to 3 generations of respiratory bronchioles which bear alveoli. Several generations of alveolar ducts arise from the alveoli, and after a variable number of divisions they terminate as alveolar sacs. The *acinus* is the name given to that segment of lung supplied by a terminal bronchiole, with its respiratory bronchioles, alveolar ducts and sacs, and alveoli.

4.2.2 Geometry of the Respiratory Airways

The geometry of the lung airway in mammalian species has been well studied (Weibel, 1963). Measurements of airways geometry mainly obtained from airway cast dissections of injected silicone rubber. Once the silicone has set, the airways are removed, the tissue digested, and the cast trimmed to the level of airways that are required. The length, diameter, branching angle, and angle of inclination to gravity of each segment are then measured. Attempts at modelling the bronchial structure-function relationship have meant that exhaustive sampling of mammals have been necessary in order to characterise the geometry of the airways. Generally, these measurements have been limited to subsections of the airways.

The work of Horsefield and Cumming (1968a) and Horsefield et al. (1971) have shown that the bronchial tree, between the trachea and the alveoli, exhibits self-similarity and asymmetry. Also, Parker et al. (1971) observed that the two daughter branches arising at a bronchial segment bifurcation often differ in diameter, length, and angle - leading to a highly heterogeneous structure. Based on these observations, Mandelbrot (1983) proposed the analogy between respiratory airways geometry and a *fractal*; since they both exhibit heterogeneity and self-similarity.

Due to the heterogeneity in branch size and number, interpretation of published data is difficult since no specific method of ordering the branches exists. Despite these problems, many workers have presented comparable results which demonstrate a close relationship between the diameter of a branch, the number of distal respiratory bronchioles it supplies, and its generation number.

All observations on the pattern of branching in the bronchial tree indicate either symmetric or asymmetric dichotomy, or a combination of both. Fig. 4.2 shows a dichotomously branching pattern, where a parent branch divides to produce two daughter branches. In a symmetric dichotomy the daughter branches have the same length and diameter. An asymmetrically dichotomous branching system is one in which there is variation in the diameters or the lengths of the branches in a given generation, or a variation in the number of generations down to the end branches, or any combination of these.

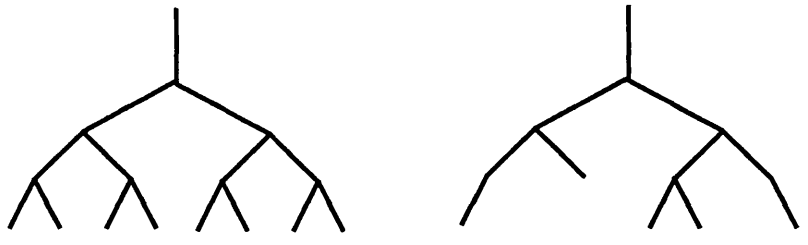


Fig. 4.2 A dichotomously branching pattern.

The Weibel model is oversimplistic in its assumption of symmetrically dichotomous branching down to the distal alveolar regions. A more realistic model is that of Horsefield and Cumming (1968a), where the branching pattern in the human airways between the trachea and the lobular branches is represented by asymmetrical dichotomy. The number of branches in this model increases by a factor of 1.38. Parker et al. (1971) extended this

model by proposing a symmetrically dichotomous branching pattern between the lobules and the distal respiratory bronchioles, where the number of branches doubles with successive generations.

4.2.3 Gas Mixing and Ventilation Distribution

Because of the dramatic increase of total cross sectional airway toward the periphery of the lung, the convective velocity of inspired gases decreases during quiet breathing from a few metres per second at the trachea to almost zero within the alveolar sacs. Somewhere in the bronchial tree, mass gas transport by convection and diffusion are of the same order of magnitude. Gomez (1965) identified this *critical zone* of the lung to be at the entrance of the acinus.

The main limitation of the Weibel model is the assumption that the bronchial tree has a symmetrical structure. This implies that not only are the lengths and cross sections of the airways the same, but the flows in all ducts of the same generation order must also be identical. A more realistic approach is the asymmetric structure proposed by Parker et al. (1971), where the branch points subtend units of different volumes or the cross sections of the daughter branches are not equal or the convective flows are not the same. The structural or functional asymmetries in the critical zone of the lungs has some significance in the quantitative description of gas transport.

When an O_2 molecule reaches a branch point, the probability that it goes into one or the other of the daughter branches depends on the pressure gradient generating a flow of convection and a concentration gradient generating a flow of diffusion. The O_2 molecules enter each branch by convection in proportion to volume flow, which is proportional to the volume of each unit. Therefore, convective flow per se does not generate inhomogeneous concentrations. However, because diffusion flow is proportional to cross-section, which is equal in both units at the branch point, the number of O_2 molecules entering the small unit, divided by its volume, is larger. This results in an inhomogeneity in O_2 concentration between the two parallel units. It must be noted that convective and diffusive variables are linked by a continuity relationship: if more O_2 molecules go into one branch (due to a larger diffusion flow), fewer molecules remain available to flow into the other branch irrespective of the convective flow. The quantitative formulation of the transport at a bifurcation leads to the concept of interdependence of diffusion and convection (Paiva and Engel, 1979).

The interdependence at branch points situated outside the critical zone is negligible. If the branch points are proximal to the diffusion front, the transport is purely convective during most of the inspiration; which implies zero diffusion flow. The consequence of a purely convective flow at these branch points is that the amount of inspired gas entering each

parallel unit is proportional to the flow and determined purely by the respiratory mechanics. At the other extreme, convective velocity approaches zero, and the parallel units with a very peripheral branch point equilibrate rapidly by diffusive mixing. Their concentrations will be equal, and the units can be considered as a single unit with a volume equal to the sum of the two volumes. When two units are identical but the flows at their entry differ, the situation corresponds to units with different compliances.

The pattern of change in gas concentrations leaving the acinus is a consequence of a complex interaction between convection and diffusion at each of the serially distributed branch points and does not merely reflect a particular concentration gradient at end inspiration. The intra-acinar gas inhomogeneities, although substantial, contribute to a small degree to overall impairment in gas exchange. Therefore, in normal subjects, the acinus may functionally constitute a single gas exchange unit.

Mead et al. (1955) proposed the theory that the time constant of the different parallel pathways in the lung determined whether or not the pathways behaved synchronously at different respiratory frequencies. Otis et al. (1956) modelled the respiratory system as an electrical RC circuit consisting of parallel units. During sinusoidal forcing, a system consisting of multiple RC circuits behaves as a single pathway only if the time-constants of the individual pathways are equal. Sweeping the excitation frequency causes a proportionate change in impedance of each pathway, so that the distribution of flow (and change in volume) will not be altered. If the time constants are not equal, then at higher frequencies the impedances of the separate pathways depend increasingly on the resistances, so that low resistance pathways receive more flow. The inhomogeneity of flow distribution has both a spatial component (unequal ventilation-volume ratios) and a temporal component (asynchrony of filling and emptying of lung regions with pendelluft between them). As a consequence, the effective compliance and resistance of the total system fall with increasing frequency. Qualitatively these concepts apply even when the pressure-flow and pressure-volume relationships are nonlinear and the forcing is not sinusoidal.

Due to the large range of path lengths from carina to alveoli, it would be remarkable if the total resistance to each unit of the same volume were equal. It would be even more remarkable if the resistances were unequal but the time constants equal because of counterbalancing differences in the compliances of the subtended units. The likely time constant inequality within the lung needs to be reconciled with the observation that in most normal subjects dynamic compliance and pulmonary resistance are not frequency dependent up to 60 breaths per minute (BPM). Hence, during normal physiological breathing frequencies, asynchronous behaviour is unlikely even for a large variation in time-constants.

Another reason proposed to account for the apparent synchrony in the face of probable time constant inequality is that due to mechanical interdependence of air spaces in the lung. The interacting forces between the units helps to counterbalance any affect that tends to make a unit change volume at a different rate than that of the surrounding lung. Mead et al. (1970) modelled this interdependence as a decrease in effective compliance with increasing asynchrony.

In patients with airflow obstruction, Mead (1969) suggested that asynchronous behaviour could be due to phase differences between the dead space and the parenchyma. In cases with obstruction in peripheral airways such phase differences could account for measured values of compliance and resistance at different breathing frequencies in patients with airways obstruction. Frequency dependent compliance is frequently seen in lung disease when pulmonary resistance is still normal.

The interaction between diffusion and convection at peripheral branch points could produce non-uniformity of alveolar gas composition even when expansion of the acinus is homogeneous. Models representing the asymmetrical anatomy predict a complex interdependence of gas composition between branch points in series and airways in parallel. The sensitivity of intra-acinar gas composition to the details of anatomic structure constitutes one of the major weaknesses in the understanding of gas mixing and distribution within the lung. With improved morphometric techniques and computer-aided 3-D reconstructions, accurate anatomic data will become more available.

4.3 High Frequency Jet Ventilation

High frequency ventilation (HFV) is a form of mechanical ventilatory support that differs from conventional modes of ventilatory support in both its relative tidal volume (V_T) and respiratory rate. The major physiological considerations that need to be investigated are the effects of HFV on pulmonary mechanics and the mechanisms by which gas transport is maintained. Table 4.1 summarises the many forms of HFV that exist, covering an arbitrary range of ventilation frequencies. As Smith (1982) pointed out, these definitions of ventilation do not necessarily signify a transition between different gas transport mechanisms. Drazen et al. (1984) have defined HFV as ventilation at a respiratory rate equal to or greater than four times the normal resting respiratory rate of the subject.

The differences between the various types of HFV can be described in terms of the method of gas movement. With HFPPV, ventilation takes place with fresh gas but without gas entrainment, while in HFJV gas is forced into the airway in the form of a jet of gas from a high pressure source, resulting in the simultaneous entrainment of a second gas (Young, 1989). During HFO, gas in the airway is oscillated back and forth in a sinusoidal fashion, with a fresh gas flow-by located between the oscillator and the patient (Sjostrand, 1983).

Ventilation Type	*Respiratory Rate (Breaths/Minute)*
HFPPV - High Frequency Positive Pressure	60 - 110
HFJV - High Frequency Jet Ventilation	110 - 400
HFO - High Frequency Oscillation	400 - 2400

Table 4.1 Range of frequencies covered by high frequency ventilation.

Care must be taken during HFV to ensure that a low V_T is applied when high respiratory frequencies are used. This is necessary for two reasons:

- applying a large V_T at high respiratory rates could result in hyperventilation;
- the mechanical work, which would have to be done by or on the respiratory system to achieve such rapid rates with large V_T, could be prohibitively large.

When ventilating with a small V_T but larger than anatomic dead space volume we must consider how ventilation will be distributed at high respiratory rates, and the effect of these rapid rates on non-ventilatory functions of the lung. Alternatively, for cases with V_T smaller than the anatomic dead space we must consider what physical processes are acting to transport gases from the alveolar zone to the airway opening.

4.3.1 Mechanics and Distribution of Ventilation

The main purpose of gas flow within the airways is to create the conditions leading to gas transport. Also, irrespective of the tidal volume V_T or respiratory rate used, the overall gas exchange achieved is related to the matching of blood flow and ventilation in various regions of the lung. Therefore, to understand the physiology of gas exchange during HFJV we must examine the factors that may influence airflow and blood flow distribution. As discussed in Sec. 4.2.3, all transport mechanisms rely on two fundamental processes: bulk convection and molecular diffusion. The dominant mechanism is dependent on the local airway geometry and flow conditions.

Despite the effort of many workers to determine the mechanisms of gas exchange under specific conditions of HFV, there is still no clear-cut agreement on the exact mechanism underlying HFV. Neither bulk convection or molecular diffusion alone can account for the effective gas mixing in the lung using a tidal volume less than the deadspace. A combination of effects may suggest an explanation for the augmented gas exchange in HFJV. Asynchronous alternation, longitudinal dispersion and pendelluft flow at terminal alveoli may also affect gas exchange. This study has adopted the hypothesis (Smith and

Lin, 1989) that ventilation at a rate approaching the resonant frequency of the airways is the cause of augmented gas mixing during HFJV. Further consideration of this hypothesis - modelling and validation - are considered in Sec. 4.4, Sec. 7.2 and Chapters 5 and 6.

The physiological aspects of HFV are considered in Sec. 7.2, where an outline is given of the factors affecting gas exchange during artificial ventilation and criteria are specified for selecting control variables for implementing automatically controlled ventilation.

4.3.2 Clinical Application of HFJV

Clinically, HFJV has been used successfully in laryngoscopy, bronchoscopy, oral surgery with complicated airways management, extracorporeal shock-wave lithotripsy and in the treatment of bronchopleural fistula and hyaline membrane disease. Although, HFV has not replaced conventional low rate ventilation, it has become an additional mode of effective ventilation.

Several comparative studies found little difference between HFJV and IPPV:

- A study by Carlon et al. (1983) could find no clinical difference between HFJV and IPPV. The main difference was the tidal volume which was one third of that with IPPV and CO_2 clearance was slightly but significantly better in HFJV, PO_2 was higher in IPPV. Apart from a higher cardiac index during IPPV, other haemodynamic variables was identical with the two types of ventilation.
- Brader et al. (1981) performed cardio-pulmonary resuscitation (CPR) using HFJV with transtracheal catheterisation. They showed that HFJV maintained equally adequate gas exchange and carotid artery blood flow values during CPR as with IPPV.
- Frey et al. (1980) reported on the effect of prolonged ventilation with HFJV. The average survival time of the HFJV group was longer than that of the IPPV group. In addition, the IPPV group showed a much earlier decrease in PO_2.

Jet ventilation techniques are widely accepted for bronchoscopies and laryngoscopies under general anaesthesia. The technique provides good oxygenation and ventilation with good airway control and excellent operative conditions. A definite benefit, particularly during laryngo-microsurgery, is an almost motionless operative field due to the high rates and small tidal volume used (Kim et al., 1986).

Massive trauma of the oro-facial region yield embarrassing problems during anaesthetic management in regard to maintaining the airway. Miller et al. (1982) using percutaneous transtracheal HFJV has achieved adequate oxygenation and ventilation.

Derderian at al. (1982) report successful management of bronchopleural fistula using HFJV with the beneficial effect of decreasing the air leak. The main advantage in addition to adequate ventilation and oxygenation is simplified and comfortable weaning from

mechanical ventilation in patients requiring respiratory support. Relatively low airway pressure may reduce the risk of barotrauma in the case of adult respiratory distress syndrome (ARDS) and hyaline membrane disease.

The effectiveness of HFJV during extracorporeal shock-wave lithotripsy has been attributed to the reduction in mean airway pressure and kidney movement. This minimises parenchymal damage to the kidney as well as a reduction in shock waves required for effective disintegration of stones (Carlon et al., 1985).

The predicted advantages of the use of HFJV on patients with impaired haemodynamic status have not yet been widely confirmed on a clinical basis. Dedhia (1981), based on results from six patients undergoing open-heart surgery, showed that ventilation with low intra-thoracic pressure had a major affect in maintaining adequate ventilation and oxygenation with minimal interference with the haemodynamics.

4.4 An Acoustic Model of the Respiratory System

The difficulty of applying conventional concepts of respiratory mechanics to high frequency ventilation (HFV) has limited its application as a form of mechanical ventilatory support. Recent studies have adopted a new acoustic model of the respiratory system in order to clarify the physiology and mechanics of ventilation.

The majority of workers have concentrated on the measurement of input acoustical impedance (Ishizaka et al., 1976; Fredberg and Hoenig, 1978) and with the measurement of sound transmission through the airways (Jackson and Olson, 1980; Goncharoff et al., 1989; Wodicka et al., 1989) in the frequency range 1 kHz - 20 kHz. The results from these high frequency studies are summarised in Sec. 4.4.1.

In contrast, much work has been carried out on low frequency investigations of the mechanical response of the respiratory system (Michaelson et al., 1975; Smith and Lin, 1989) for frequencies less than 20 Hz. These studies have been concerned with the relationship of stimulus-response data to the structure and function of the respiratory system in normal and abnormal states. Sec. 4.4.2 describes the low-frequency acoustic model and its application in this study.

4.4.1 Wide-Band Acoustic Modelling

The relationship between the acoustic wavelength (λ) and the physical dimensions of the airways (d) for high frequencies (>1 kHz) is given by $\lambda \ll d$. In this range of frequencies the wavelength of pressure oscillations in the airways is less than or of the order of lengths characteristic of the spatial extent of the airways. The resulting pressure oscillations have

temporal and *spatial* implications; the analysis of the acoustic system requires a *distributed-parameter* approach similar to that found in electrical transmission-line theory.

Consideration of the acoustic response of the respiratory airways at high frequencies has shown that local resonances can be excited within the airways (Fredberg and Hoenig, 1978; Goncharoff et al., 1989; Wodicka et al., 1989). These resonances are a result of the changes in acoustic impedance seen by the waves as they travel through the airways. The acoustic impedance of an airway is defined by its geometric and mechanical properties. In particular, branch points are associated with a change in cross-sectional area between the parent and daughter airways which results in an acoustic impedance mismatch. Using transmission-line theory (Glazier and Lamont, 1958), it can be shown that these junctions act as a site of reflection where incident acoustic waves are partially reflected to produce standing wave oscillations. The fraction of reflected waves is determined by the *reflection coefficient.*

The transmission characteristics of high frequency acoustic waves in the lungs are known to be sensitive to changes in lung structure which occur in disease. Goncharoff et al. (1989) observed that the transmission of sound through human lungs in the frequency range 5-20 kHz is dependent on the geometry and mechanical properties of the large airways, lung parenchyma and chest wall. The acoustic filtering effects of the large airways have also been predicted from modelling experiments and measurements on human lungs by other investigators (Jackson and Olson, 1980; Ishizaka et al., 1976). It is hoped that this technique can be used for non-invasive investigation of human lungs and for determining their physiological status.

4.4.2 An Acoustic Model of High-Frequency Ventilation

This study is concerned with modelling and identification of respiratory dynamics during HFJV (Sec. 4.3). This mode of ventilation falls into the general category of high frequency ventilation (HFV) and differs from conventional modes of ventilation in several respects. In particular, whilst the use of a simple first order RC lung model is valid for conventional ventilation frequencies, it is not applicable for higher frequencies. In this case, inertia must be taken into account.

Smith and Lin (1989) proposed that a patient receiving HFV could be modelled as an acoustic system in which a source of acoustic waves (the jet ventilator) radiates into an acoustic load (the patient and ambient surroundings). In this case, the relationship between the acoustic wavelength (λ) and the physical dimensions of the airways (d) for the range of ventilation frequencies covered by HFV (<25 Hz) is given by $\lambda \gg d$. In contrast to wide-band acoustic testing (Sec. 4.4.1), the wavelength of pressure oscillations in the airways are much greater than the dimension of the airways and the *spatial* variation

spatial variation of pressure becomes negligible; the system is subject to a time-varying pressure signal. The analysis of the acoustic system is simplified in that a *lumped-parameter* RLC model can be used (Kinsler and Frey, 1962). Further discussion of this approach and its application is given in Sec. 7.4.

Assuming a second-order RLC lung model (Smith and Lin, 1989), the respiratory system behaves as a simple damped oscillator which can exhibit resonance if ventilated at its resonant frequency. The degree of resonance is controlled by the mechanical damping in the system which is a function of the:

- resistance to gas flow in the airways;
- viscous and frictional forces in the expansion of the thorax and airways.

In reality, the non-parabolic velocity profiles associated with periodic flows results in a resistance that is frequency-dependent (Finucaine et al., 1975). However, this is assumed to be negligible for the purposes of this study. The damping may be of practical significance for ventilation frequencies approaching the resonant frequency of the system. Inadequate damping can lead to excessive intra-pulmonary gas dynamics and cause harm to the patient.

4.5 Respiratory Impedance

Respiratory impedance measurements by forced oscillations are increasingly used in pulmonary function testing (DuBois et al., 1956; Michaelson et al., 1975) since they permit the evaluation of total respiratory resistance over a wide range of frequencies, which is of interest in diagnosing airway obstruction and detecting mechanical non-homogeneity. The reactive component of the impedance, a function of the airways and chest-wall elasticity and inertia, has been shown to be indicative of chronic obstructive pulmonary disease (Michaelson et al., 1975).

Many investigators have employed DuBois' technique or a modification to study respiratory mechanics in mammals (Brody et al., 1956; Hull et al., 1961; Grimby et al., 1968; Hyatt et al., 1970) where forced sinusoidal pressure oscillations are applied at the mouth using a loudspeaker and measuring the induced flow. The impedance is calculated by comparing the magnitude and phase of pressure to flow harmonics over a selected range of frequencies (Sec. 5.3.2). The results confirm that the respiratory system behaves approximately like a second-order resonant system (Sec. 4.4). However, these observations were over a limited frequency range (3-10 Hz) and had poor spectral resolution.

Michaelson et al. (1975) developed an experimental procedure for rapidly measuring respiratory impedance over an expanded frequency range with a enhanced spectral resolution to previous methods (Sec. 5.3.3). The technique is a modified version of

DuBois' method except that pseudo-random pressure variations are applied at the mouth. The impedance is calculated using FFT-based spectral analysis of pressure and flow signals. Using this approach, investigators have confirmed more accurately the second-order characteristics of the respiratory system.

All existing impedance measurement circuits apply acoustic pressure oscillations at the mouth. As recognised by DuBois et al. (1956), the impedance of extrathoracic airway walls (e.g. mouth, pharynx, larynx) which are mechanically in parallel with the respiratory system shunt flow away from the trachea. The measurement artefact produced by this effect is:

- an under-estimation of respiratory impedance;
- incorrect estimation of the frequency-dependent properties of the impedance;
- over-estimation of the resonant frequency of the airways - corresponding to an impedance which is purely resistive.

The common approach to minimise the problem is to firmly support the patients' cheeks (DuBois et al., 1956; Hayes et al., 1979; Landser et al., 1982). However, the residual error may still be quite large due to upper airway wall motion (Michaelson et al., 1975).

The instrumentation described in Chapter 5 provides an alternative and more elegant method for computing the respiratory impedance of a patient (Sec. 5.3.4). This approach is based on the forced random noise technique of Michaelson et al. (1975), however, the pressure source is introduced into the patient via a high frequency ventilator and jet cannula. White-noise pressure oscillations can then be injected at the carina - bypassing the shunt impedance mentioned above. The properties of the jet cannula can be selected such that pressure and flow oscillations at the carina can be accurately measured at the mouth. Preliminary experiments using lung surrogates have been performed and the results are presented in Sec. 5.3.4.

Another advantage of the instrumentation in Chapter 5 is it allows high resolution, real-time computation of respiratory impedance. The microcomputer-based system of Pelle et al. (1986) is the only reported system which approaches this level of performance. Their system uses the Michaelson technique for computing respiratory impedance spectra and computes a single impedance- and coherence spectrum from 512-points of data over a bandwidth of 0-30 Hz (maximum spectral resolution Δf = 0.25 Hz). The system only performs 4 spectral averages and the off-line calculations take 72 seconds. The relative speed enhancement achieved by the HFJV system described in Chapter 5 can be compared: the system calculates two sets of transfer function and coherence spectra on 1024-points of data over a bandwidth of 0-50 Hz (maximum spectral resolution $\Delta f = 0.125$ Hz) in approximately 60 ms - up to 2048 spectral averages can also be performed in real-time.

4.6 References

Brady, E., Klain, M., Safar, P. and Bircher, N. (1981): "High-frequency jet ventilation versus IPPV in cardio-pulmonary resuscitation for asphyxia in dogs", *Crit. Care Med.*, **9**, 162.

Brody, A.W., DuBois, A.B., Nissel, O.I. and Engelberg, J. (1956): "Natural frequency, damping factor and inertance of the chest-lung system in cats", *Am. J. Physiol.*, **186**, 439-443.

Carlon, G.C., Howland, W.S. and Ray, C. (1983): "High frequency jet ventilation: a prospective randomised evaluation", *Chest*, **84**, 551-559.

Carlon, G.C., Barker, R.L., Benua, R.S. and Guy, Y.G. (1985): "Airway humidification with high frequency jet ventilation", *Crit. Care Med.*, **13**, 114-117.

Dedhia, H.V. (1981): "Haemodynamic effect of high frequency ventilation in open heart surgery patient", *Crit. Care Med.*, **9**, 158.

Derderian, S.S., Rajagopal, K.R. and Abbrecht, P.H. (1982): "High frequency positive pressure jet ventilation in bilateral bronchopleural fistula", *Crit. Care Med.*, **10**, 119-121.

Drazen, J.M., Kamm, R.D. and Slutsky, A.S. (1984): "High frequency ventilation", *Physiol. Revs.*, **64**, 505-543.

DuBois, A.B., Brody, A.W., Lewis, D.H. and Burgess, B.F. (1956): "Oscillation mechanics of lungs and chest in man", *J. Appl. Physiol.*, **8**, 587-594.

Finucaine, K.E., Dawson, S.V., Phelan, P.D. and Mead, J. (1975): "Resistance of intra-thoracic airways of healthy subjects during periodic flow", *J. Appl. Physiol.*, **38**, 517.

Fredberg, J.J. and Hoenig, A.A. (1978): "Mechanical response of the lungs at high frequencies", *J. Biomech. Eng.*, **100**, 57-66.

Frey, D.J.M., Beller, U., Ziermann, T., Lesch, R. and Deilmann, M. (1980): "Prolonged jet ventilation with high-frequency injection", *Excerpta Medica*, **533**, 248.

Gallagher, T.J. (1986): "Current status of high frequency ventilation", *Program*, **171**, 1-4.

Glazier, E.V.D. and Lamont, H.R.L. (1958): *Transmission an propagation*, The Services Textbook of Radio, 5, H.M. Stationery Office.

Gomez, D.M. (1965): "A physico mathematical study of lung function in normal subjects and in patients with obstructive pulmonary diseases", *Med. Thorac.*, **22**, 275-294.

Goncharoff, V., Jacobs, J.E. and Cugell, D.W. (1989): "Wideband acoustic transmission of human lungs", *Med. & Biol. Eng. & Comput.*, **27**, 513-519.

Grimby, G., Takishima, T., Graham, W., Macklem, P. and Mead, J. (1968): "Frequency dependence of flow resistance in patients with obstructive lung disease", *J. Clin. Invest.*, **47**, 1455-1465.

Hayes, D.A., Pimmel, R.L., Fulton, J.M. and Bromberg, P.A. (1979): "Detection of respiratory mechanical dysfunction by forced random noise impedance parameters", *Am. Rev. Respir. Dis.*, **120**, 1095-1100.

Horsefield, K. and Cumming, G. (1968a): "Morphology of the bronchial tree in man", *J. Appl. Physiol.*, **24**, 373-383.

Horsefield, K. and Cumming, G. (1968b): "Functional consequences of airway morphology", *J. Appl. Physiol.*, **24**, 384-390.

Horsefield, K., Dart, G., Olson, D.E., Filley, G.F. and Cumming, G. (1971): "Models of the human bronchial tree", *J. Appl. Physiol.*, **31**, 207-217.

Hyatt, R.E., Zimmerman, I.R., Peters, G.M. and Sullivan, W.J. (1970): "Direct writeout of total respiratory resistance", *J. Appl. Physiol.*, **28**, 675-678.

Ishizaka, K., Matsudaira, M. and Kaneko, T. (1976): "Input acoustic-impedance measurement of the subglottal system", *J. Acoust. Soc. Am.*, **60**, 190-197.

Jackson, A. and Olson, D. (1980): "Comparison of direct and acoustical area measurements in physical models of human central airways", *J. Appl. Physiol.*, **48**, 896-902.

Kinsler, L.E. and Frey, A.R. (1962): *Fundamentals of acoustics*, p182, New York, Wiley.

Kim, W.O., Kim, J.R., Park, K.W. and Cho, C.H. (1986): "High-frequency ventilation for suspension laryngo-microscopy under general anaesthesia", *Yonsei Med. J.*, **27**, 25-29.

Landser, F.J., Clement, J. and Van de Woestijne, K.P. (1982): "Normal values of total respiratory resistance and reactance determined by forced oscillations - influence of smoking", *Chest*, **81**, 586-591.

Mandelbrot, B.B. (1983): *The fractal geometry of nature*, New York, Freeman.

Mead, J. (1969): "Contribution of compliance of airways to frequency-dependent behaviour of lungs", *J. Appl. Physiol.*, **26**, 670-673.

Mead, J., Lindgren, I. and Gaensler, A.E. (1956): "Mechanical properties of the lungs in emphysema", *J. Clin. Invest.*, **34**, 1005-1016.

Mead, J., Takishima, T. and Leith, D. (1970): "Stress distribution in lungs: a model of pulmonary elasticity", *J. Appl. Physiol.*, **28**, 596-608.

Michaelson, E., Grassman, E. and Peters, W. (1975): "Pulmonary mechanics by spectral analysis of forced random noise", *J. Clin. Invest.*, **56**, 1210-1230.

Miller, J., Iovin, O.W., Fine, J. and Klain, M. (1982): "High-frequency jet ventilation in oral and maxillofacial surgery", *J Oral Maxillofac. Surg.*, **40**, 790-793.

Otis, A.B., McKerrow, C.B., Bartlett, R.A., Mead, J., McIlroy, M.B., Selverstone, N.J. and Radford, E.P. (1955): "Mechanical factors in distribution of pulmonary ventilation", *J. Appl. Physiol.*, **8**, 427-443.

Paiva, M. and Engel, L.A. (1979): "Pulmonary interdependence of gas transport", *J. Appl. Physiol.*, **47**, 296-305.

Parker, H., Horsefield, K. and Cumming, G. (1971): "Morphology of distal airways in the human lung", *J. Appl. Physiol.*, **31**, 386-391.

Pelle, G., Lorino, A.M., Lorino, C., Mariette, C. and Harf, A. (1986): "Microcomputer-based system to calculate respiratory impedance from forced random noise data", *Med. & Biol. Eng. & Comput.*, **24**, 541-544.

Phalen, R.F., Yeh, H.C., Schum, G.M. and Raabe, O.G. (1978): "Application of an idealised model to morphometry of the mammalian tracheobronchial tree", *Anat. Rec.*, **190**, 167-176.

Sjostrand, U.H. (1983): "High-frequency positive-pressure ventilation", *Int. Anesthesiol. Clin.*, **21**, 59-81.

Smith, B.E. and Lin, E.S. (1989): "Resonance in the mechanical response of the respiratory system to high frequency jet ventilation", *Acta Anaesthesiol. Scand.*, **Supp. 90**, 65-69.

Weibel, E.R. (1963): *Morphometry of the human lungs*, New York, Academic-Press.

Wetzel, R.C., Gordon, J.B., Gregory, T.J., Gioia, F.R., Adkinson, F. and Sylvester, J.T. (1985): "High-frequency ventilation attenuation of hypoxic pulmonary vasoconstriction", *Am. Rev. Respir. Dis.*, **132**, 99-103.

Wodicka, G.R., Stevens, K.N., Golub, H.L., Cravalho, E.G. and Shannon, D.C. (1989): "A model of acoustic transmission in the respiratory system", *IEEE Trans. Biomed. Eng.*, **BME-36**, 925-933.

Young, J.D. (1989): "Gas movement during high-frequency ventilation", *Acta Anaesthesiol. Scand.*, **Supp. 90**, 70-71.

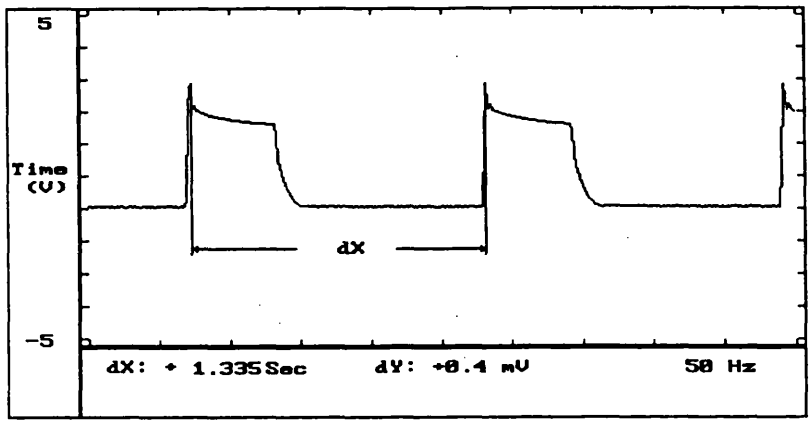
Chapter 5
Instrumentation for Measurement and Control In High Frequency Jet Ventilation

5.1 Introduction

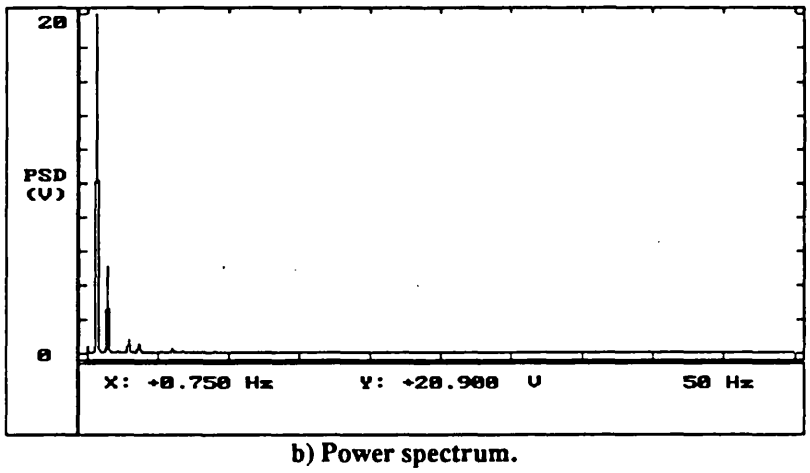
This chapter presents computer-aided instrumentation for real-time measurement and control of high frequency jet ventilation in the area of critical care medicine. The system has been initially developed to test the hypothesis that acoustic resonance of the respiratory airways represents an optimal state for alveolar gas exchange. Real-time signal processing algorithms have been implemented around a user-friendly system to aid in the identification of transfer function relationships between respiratory data. This analysis is the first step towards developing an automatically controlled, self-tuning high frequency jet ventilator.

High frequency jet ventilation (HFJV), introduced in Sec. 4.3, is a recognised form of mechanical ventilatory support used in anaesthesia. The technique differs from conventional modes of ventilatory support in both its relative tidal volume and respiratory rate. Several studies have shown that HFJV is capable of maintaining adequate gas exchange in cases where conventional methods have either failed or proved to be impractical. The main advantages of HFJV include lower peak and mean airway pressures (Heijman et al., 1972), a reduction in pulmonary barotrauma (Keszler et al., 1982) and less disturbance to cardiovascular function (Sjostrand, 1980). A more detailed consideration of the subject may be found in several recent reviews (Smith, 1982; Drazen et al., 1984; Kolton, 1984).

General acceptance of HFJV has been inhibited by lack of understanding of the mechanisms by which HFJV maintains gas exchange, and also by a lack of practical



a) Voltage-time waveform.



b) Power spectrum.

Fig. 5.1 Jet-ventilator binary drive pressure signal.

guidelines for clinicians on when to apply HFJV and what ventilator parameters to set for optimum gas exchange.

One approach to the better understanding of gas exchange during HFJV treats the patient's respiratory system as an acoustic resonator whose characteristics vary over the range of HFJV frequencies (Lin and Smith, 1987). Preliminary results from animal studies support the hypothesis that the respiratory system behaves as an acoustic resonator (Smith and Lin, 1989).

The instrumentation described in this chapter is designed to provide real-time analysis of multi-channel respiratory data within the clinical environment. Respiratory dynamics can be examined with extremely high resolution in a fraction of the time taken by previous workers with only minimal changes to existing jet ventilation procedures. The system will eventually incorporate automatic closed-loop control and management of ventilator parameters to optimise gas exchange in patients receiving anaesthesia.

5.2 System Specification

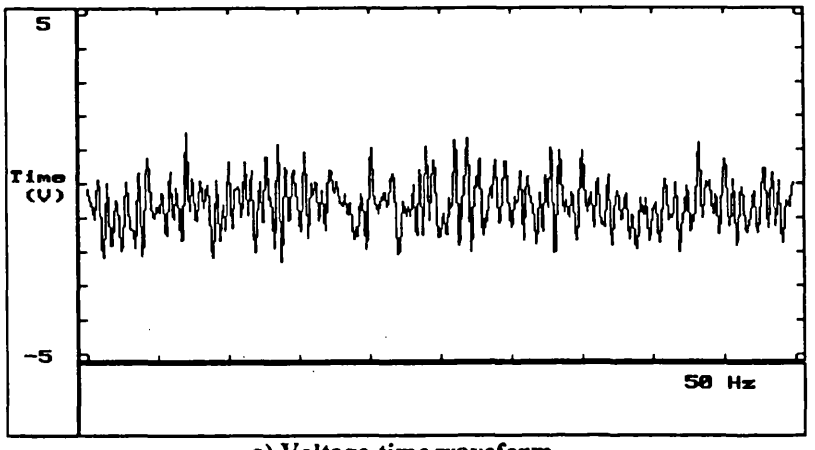
5.2.1 High Frequency Jet Ventilator

The Penlon high frequency jet ventilator used in this study acts as a hydraulic waveform generator that produces binary pressure pulses (Fig. 5.1) with variable amplitude, period and mark-to-space ratio. Such stimuli can, at best, be approximated to sine waves, and swept sine-wave analysis performed to determine the respiratory system dynamics (Sec. 5.3.2). Smith and Lin (1989) who used this method to determine the respiratory dynamics of animals during HFJV obtained some useful results. However, the measurements were tedious, time-consuming, and particularly cumbersome to perform. Another disadvantage with this type of analysis is that it can lead to erroneous results when applied to systems which adapt to the input stimulus or show fatigue.

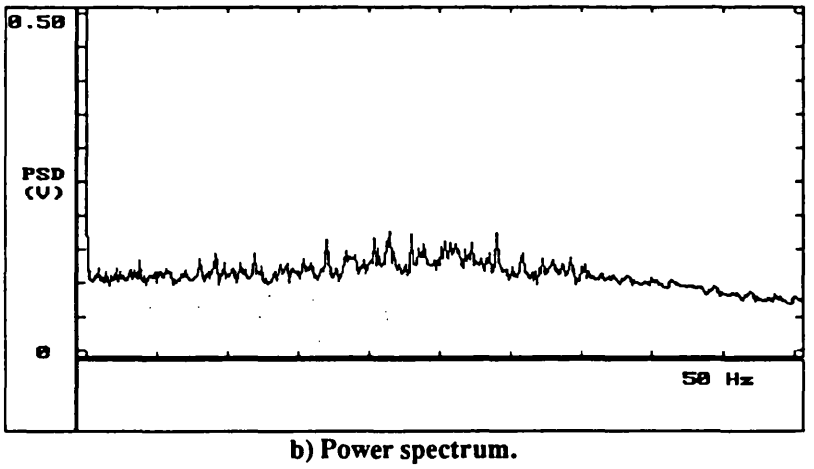
An alternative approach requires modifications to the high frequency jet ventilator (Sec. 5.3.1) such that pressure stimuli of any waveshape can be generated. Band-limited white-noise stimuli, with a flat power spectrum (Fig. 5.2), can then be used to efficiently perform transfer function analysis of the respiratory system (Sec. 5.3.3). This technique inherently provides a method for rapidly determining respiratory dynamics over an expanded frequency range with a frequency resolution greater than is practicable using conventional techniques.

5.2.2 Data Acquisition and Signal Processing

The method of analysing generated data involves sampling of the stimuli and response signals and performing spectral analysis using the fast Fourier transform (FFT). The sampling interval determines the maximum frequency f_{max} which can be analysed



a) Voltage-time waveform.



b) Power spectrum.

Fig. 5.2 Jet-ventilator band-limited white-noise drive pressure signal.

correctly before aliasing becomes a significant problem and is defined according to the Nyquist sampling theorem (Sec. 2.2).

This study requires a maximum of four channels of data acquisition including upper airways pressure/flow signals and abdominal/thoracic wall displacements (Sec. 5.4). The maximum bandwidth of HFJV measurements varies between 30-300 breaths per minute (BPM) or 0.5-5 Hz. Assuming that all significant spectral energy is contained within 10 times the fundamental frequency, this sets the maximum bandwidth per channel at 50 Hz. Thus, to satisfy the Nyquist sampling criterion, the data acquisition module should be capable of digitising analogue data at a minimum rate of 400 Hz.

Several analytical tools are required in order to assess, in real-time, the relationships between respiratory stimuli-response data. These include the ability to compute and display power spectral density functions, two sets of transfer function gain-, phase- and coherency-spectra from the raw time-domain signal (Sec. 5.5).

Since the instrumentation is intended for use by nonexperts, the man-machine interface should be an environment which presents the functional power of the system in a clear and concise manner (Sec. 5.6). A friendly menu-driven system based on the type of system architecture described in Chapter 3 is desirable (Hailstone et al., 1986).

5.2.3 Displacement Measurement Transducer

The present study requires measurement of abdominal and thoracic wall displacements as output variables, in order to test the hypothesis that the respiratory system behaves as an acoustic resonator over the range of frequencies attainable by HFJV. These state-variables can be measured using a strain-gauge transducer or inductance plethysmograph, however, they are known to suffer from several disadvantages (Sec. 5.7). An alternative fibre-optic transducer was developed to overcome these problems.

The development of the fibre-optic displacement transducer was considered to be critical since traditional measurement techniques suffered from artefact and poor dynamic performance. The transducer is non-invasive and provides versatility, relative freedom from artefact, high resolution and linearity over a wide bandwidth. Non-invasive measures of ventilation using chest-wall displacements are relatively easy to perform when compared with invasive techniques such as direct blood gas measurements. Also, the chest-wall dynamics are such that the delay associated with blood gas stabilisation is not incurred.

5.3 Transfer Function Analysis of the Respiratory System

The purpose of the instrumentation described in this chapter is to characterise the respiratory system dynamics in a systematic and efficient manner. The aim is to provide

the framework on which stimulus-response experiments can be carried out on the respiratory system to reveal information about its functional characteristics - such that a hypotheses can be made on its structure.

The instrumentation must take into account inherent characteristics of the system:

- non-linearities, which often are essential for optimal functioning;
- limited durations of experiments because of great variability in the data;
- low signal-to-noise ratio in the measurements.

Since the identification process is carried out through a series of stimulus-response experiments, it is important to design these experiments so as to maximise the information obtained from the system. The following sections describe the hardware changes made to the Penlon jet ventilator to support white-noise excitation of patients' airways and its application as a tool for identifying respiratory system dynamics.

5.3.1 The Modified High Frequency Jet Ventilator

The Penlon high frequency jet ventilator is an electrically and pneumatically operated time-cycled device of the type described by Smith (1985). The ventilator circuit shown in Fig. 5.3 is a modified version of the commercial design, which includes a proportional controller valve to support complex waveshape pressure excitation of patients' airways. A brief description of the ventilator is given below.

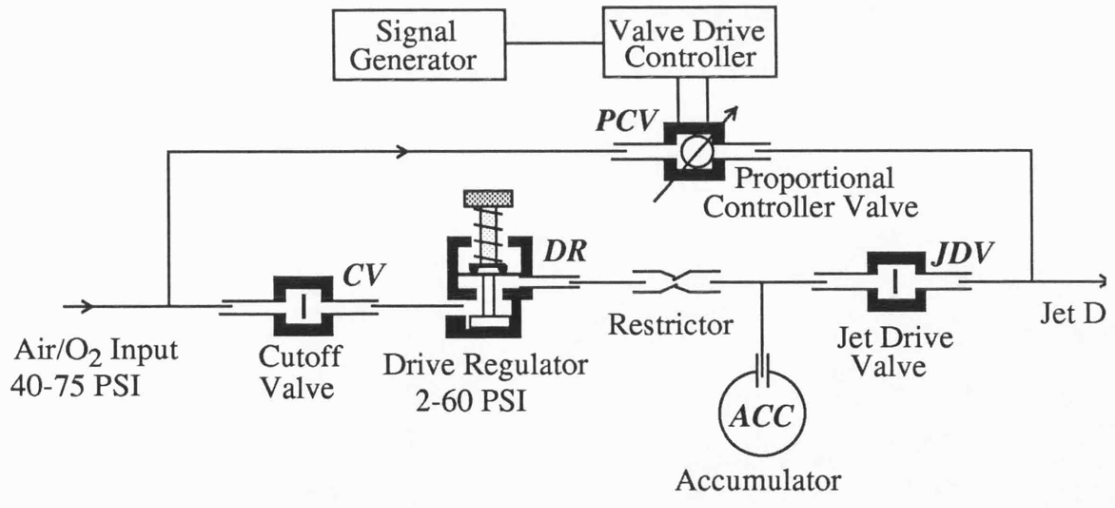


Fig. 5.3 The modified high frequency jet ventilator circuit.

Fig. 5.3 shows a simplified diagram of the ventilator hydraulic circuit. Gas from a high pressure (40-75 PSI) source is delivered into the ventilator via the inlet cutoff valve (CV) and drive pressure regulator (DR) to an accumulator (ACC). An electronic time-base

controller energises the jet drive valve (JDV) so that gas flows from the ACC to the jet line and patient periodically depending on the ventilation rate (variable over the range 40-200 BPM) and inspiration time control. The inspiratory:expiratory (I:E) ratio can be varied over the range 10-60% of the respiratory cycle. The inspiratory flow is determined by the drive pressure regulator and the respiratory airways impedance of the ventilator/ patient system.A proportional controller valve (PCV) is connected in parallel to the jet drive valve to introduce low amplitude pressure oscillations which is hydraulically summed with the normal binary output of the jet ventilator. The PCV drive controller accepts an analogue signal over the range 0-10 V and produces a continuous valve output which tracks the reference input. The frequency response characteristics of the proportional controller valve (Fig. 5.5) are investigated in Sec. 5.3.4.

A white-noise circuit, based on the design of Horowitz and Hill (1983), is used to generate a pseudo-random analogue noise signal with bandwidth selectable over a range of frequencies. The method uses digital low-pass filtering of maximal-length shift register sequences to produce band-limited white-noise. Fig. 5.2 shows a white-noise signal over a bandwidth of 50 Hz.

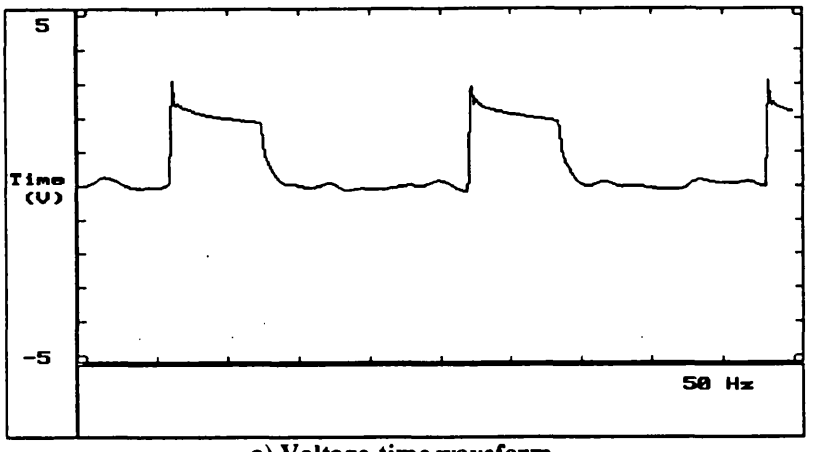
The jet ventilator can be set to operate in one of several modes to produce a) binary, b) proportional, or c) binary plus proportional pressure output. The various ventilator modes provide maximum flexibility, enabling respiratory dynamics testing for different stimuli (Sec. 5.3.4).

5.3.2 Swept Sine-Wave Analysis

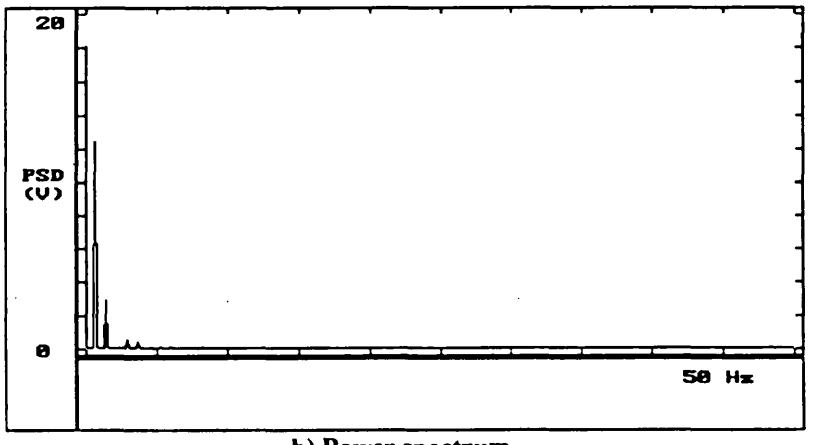
With the swept sine-wave method of identification we must make many arbitrary assumptions about the system in order to postulate a mathematical description of its structure. The identification task is then reduced to estimating the parameters of the model based on data collected from the stimulus-response experiments. The success of this approach depends solely on the skill and imagination of the modeller. Apart from the time-consumed in making the measurements, the task of designing differential equations to fit a given set of data becomes a laborious one. Finally, the model is usually valid for the chosen set of stimuli and not satisfactory for other types of input. Thus, adding new information about the system often means that the modelling process must be repeated from the beginning.

As the pilot study of Smith and Lin (1989) has shown, the swept-sine wave method of analysis was found to be inefficient for use in the clinical environment. In general, the technique suffers from many problems and may be impractical for applications where:

- the system adapts to the applied signal or shows fatigue;
- the amplitude of each *sinusoid* must be adjusted so that it remains within the



a) Voltage-time waveform.



b) Power spectrum.

Fig. 5.4 Jet-ventilator drive pressure signal with additive white noise .

linear range of the system at a given frequency;
- the output variables have long settling times before the system is in steady-state.

Since physiological systems exhibit many of these characteristics, sine-wave testing can lead to erroneous results.

5.3.3 White-Noise Identification

The system identification objectives set out in Sec. 5.2 imply that the *black-box* approach to determining the transfer characteristics of the system may be more suitable. This method of identification does not rely on any assumption about the internal topological structure of the system. This approach becomes a search for the function H[x(t)] where

$$y(t) = H[x(t)] \tag{5.1}$$

and x(t) is the stimulus and y(t) is the response, the identification task consists of estimating the system functional H.

A white-noise stimulus with a flat power spectrum over the frequency range of interest is equivalent to applying a range of sinusoids simultaneously. This technique inherently provides a method for rapidly determining respiratory dynamics over an expanded frequency range with a frequency resolution greater than is practicable with other techniques. This is far superior to using swept sine-wave stimuli, where representation is over a limited and pre-determined region of the function space.

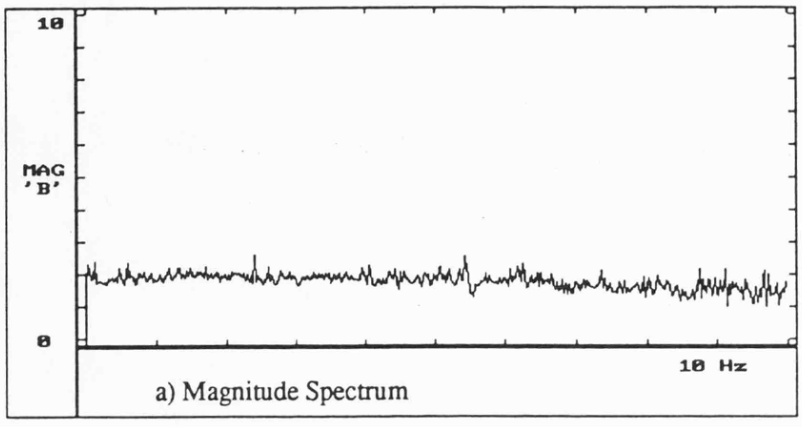
To use the white-noise approach the ventilator must be operated in either *mode b* or *c*. In particular, mode b is useful for:

- determining the frequency response characteristics of respiratory pressure, flow and displacement transducers;
- performing respiratory impedance measurements on patients; significantly simplifying existing methods of measurement.

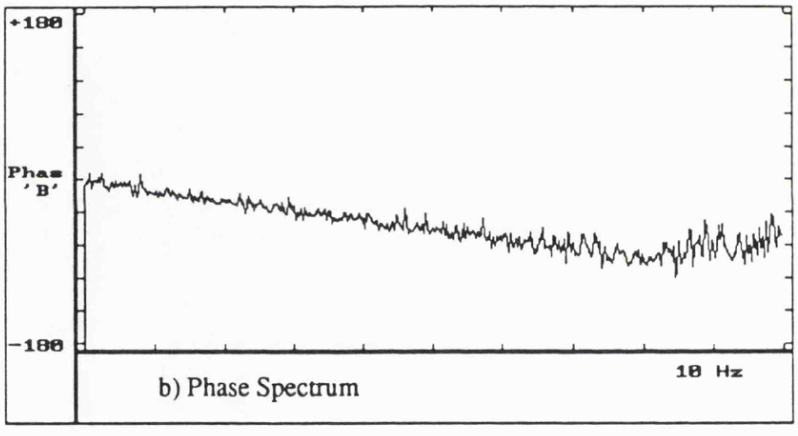
Mode c is intended for routine clinical use since it allows for on-line identification of respiratory dynamics during routine high frequency jet ventilation. Once preliminary decisions are made, the white-noise method can be used for characterising the respiratory airways over a short period of time.

5.3.4 Experimental Results

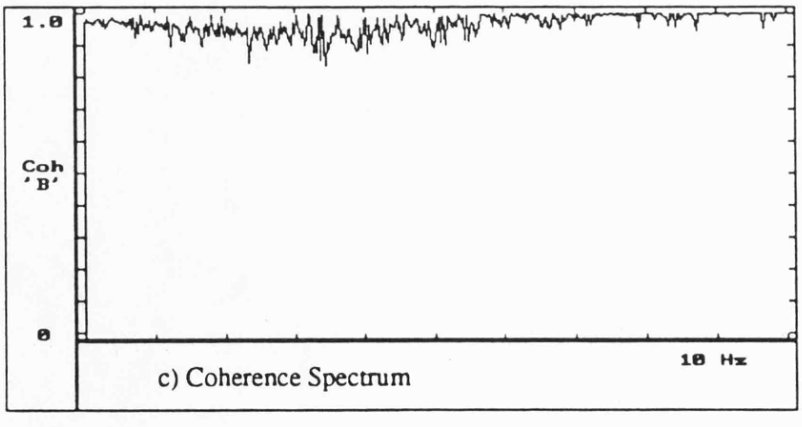
This section describes the experimental procedure used to test the dynamic performance of the modified jet ventilator circuit. The ventilator circuit now supports three independent modes of operation, including:



a) Magnitude Spectrum



b) Phase Spectrum



c) Coherence Spectrum

Fig. 5.5 Frequency response of the controller valve.

Mode a

The ventilator operates according to the specifications of the manufacturer, producing a binary jet drive pressure signal (Fig. 5.1). The drive pressure and its power spectrum are seen to be that of a pulse-width modulated square wave. The proportional controller valve is open-circuited in this mode.

Mode b

The ventilator produces a low-amplitude, band-limited white-noise pressure output (Fig. 5.2). This type of stimulus is intended for frequency response testing of pressure, flow and displacement transducers. The maximum bandwidth was limited between 0-50 Hz, since this easily covers the range of frequencies attainable by HFJV. This mode can also be used to ventilate patients under special circumstances (e.g. during respiratory impedance measurements). However, it should be applied for short durations only, otherwise it could cause harm to the patient by over-inflating the lungs.

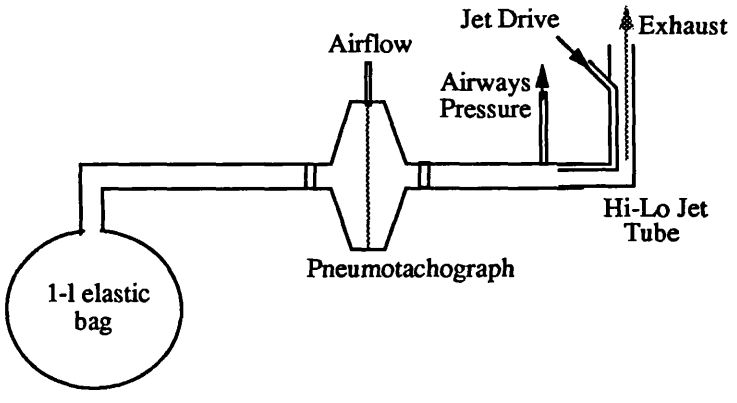
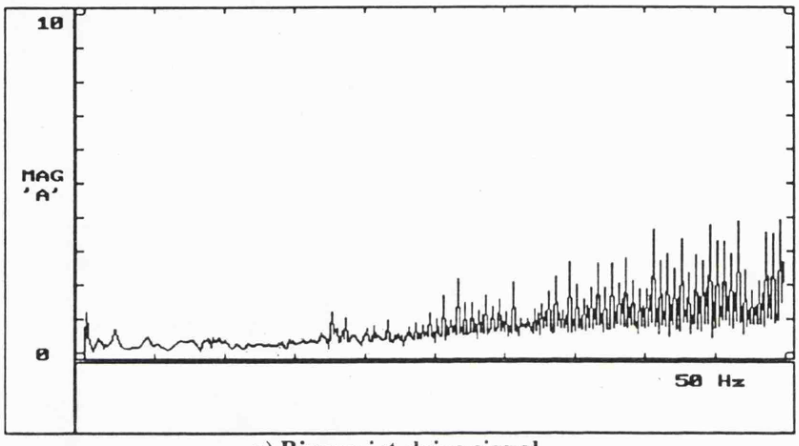


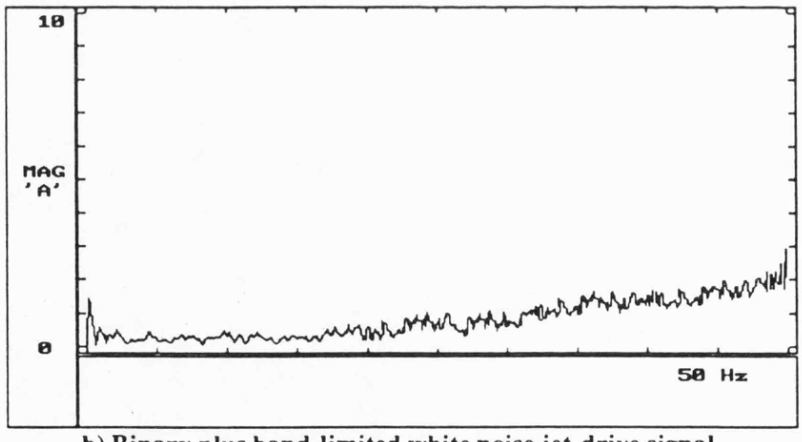
Fig. 5.6 Respiratory impedance measurement circuit.

Mode c

This mode is the result of a linear summation of modes a and b. The drive pressure comprises a square-wave signal plus small amplitude white-noise oscillation. Mode c is extremely useful since the white-noise oscillations are too small to have any affect on routine high frequency jet ventilation. Although the white-noise can be seen on the raw signal (Fig. 5.4a), the power spectrum (Fig. 5.4b) shows no evidence of white noise. This was deliberately incorporated into the system to minimise the relative power of white-noise compared to the binary signal. The system response to the white-noise can be extracted by performing spectral averaging, as demonstrated by the following experiment.The frequency response characteristics of the proportional controller valve was measured with the ventilator set to mode b. The transfer function (Fig. 5.5) was calculated from the ratio of the pressure measured at the output to that at the input to the valve. The linearity of the valve can be seen from a flat magnitude spectrum and a linear



a) Binary jet-drive signal.



b) Binary plus band-limited white noise jet-drive signal.

Fig. 5.7 Airways impedance measurements using two types of stimulus.

phase response over the range of HFJV frequencies. The coherence of near unity over the whole frequency range validates the accuracy of these measurements.

The effectiveness of the white-noise approach to system identification was tested by performing an experiment to determine the input impedance of a lung surrogate. The impedance measurement circuit (Fig. 5.6) shows a 1-litre elastic bag receiving ventilation via the jet lumen of a Hi-Lo jet tracheal tube. Airways pressure is monitored via the proximal pressure transducer mounted on the ventilator. A Fleisch pneumotachograph is connected in series with the elastic bag to monitor the gas flow rate. The airways impedance is calculated as the ratio of airways pressure to flow harmonics.

The objective of the experiment was to investigate the variation of results for different types of stimulus: a) a binary input (Fig. 5.7a) and b) a binary plus white-noise input (Fig. 5.7b). The results indicate that both measurements of impedance exhibit a similar trend. However, the white-noise stimulus has resulted in a markedly *cleaner* impedance spectrum than the case with binary stimulus. This indicates that the low-amplitude white-noise signal has increased the signal-to-noise ratio of measurements by boosting the input energy across all frequencies.

Hence, the white-noise method helps to provide a systematic procedure for characterising a system; it provides characterisation of the system over all possible inputs; it is simple to apply, once preliminary decisions are made and it gives good results over a short period of time.

5.4 Computer-Based Real-Time Measurement System

Since its introduction, the single component digital signal processor (DSP) has become a significant device for tackling high-speed signal processing tasks. The DSP is more than just a peripheral device for performing front-end real-time signal conditioning under the control of a host microcomputer. It features increased functionality and computational power enabling it to take on the role of the central processing function of the system.

The hardware architecture adopted is that of a host IBM-AT personal computer and a front-end, real-time DSP. This dual processor configuration was selected since it separated the overall task into two functional sub-units and then allocated the work amongst the two processors according to their relative strengths.

The host machine is an IBM PC-AT compatible microcomputer, based on the Intel 80286-12 CPU, with a 80287-8 maths coprocessor to increase the speed of floating point computations. The system graphics uses the new IBM colour VGA standard 640x480 pixels with 256 KB of video RAM. The system contains 640 KB of RAM expandable to 4.6 MB, a 40 MB fixed disk and a 40 MB streaming tape backup unit.

A high speed DSP card, occupying a single slot on the host motherboard, comprises a TMS320C25 running at 40 MHz with 64 KWords of combined program and data memory (zero-wait states). A dual port memory system on the DSP allows the host to access the DSP memory at any time, even when the DSP is active on other tasks. An on-board timer can be used to generate vectored-interrupts allowing complex applications to be developed which make full use of the DSP bandwidth.

The TMS320 DSP device offers distinctive features in terms of mathematical capability/ speed and incorporates on-chip the functions needed to achieve maximum throughput and standalone operation. The device architecture was selected since it supports:

- a wide dynamic range with sufficient resolution to achieve a high degree of performance and minimal quantisation/truncation errors;
- maximum throughput due to the high degree of internal parallelism. The pipeline facility can be used to achieve single-cycle program execution;
- on-chip memory blocks organised to support parallel operations. The on-chip RAM is of sufficient size for calculations to be downloaded for zero wait-state execution;
- minimal housekeeping due to the large number of hardware registers available for storing intermediate results;
- a wide variety of commercially available development tools.

An analogue interface card comprising 4-input and 2-output channels connects directly to the DSP via a 50-way ribbon connector. Each input channel has an anti-aliasing filter and sample-and-hold amplifier. This system permits the simultaneous sampling of 4 channels of data which are multiplexed into a single analogue-to-digital converter (12-bit, 3 μs conversion time). This permits 4-channels of data acquisition at a maximum sampling rate of 60 KWords per second. Two 12-bit digital-to-analogue converters (3 μs settling time) are also available for transmitting analogue signals to external devices.

5.5 Algorithms for Real-Time Spectral Analysis

The host is responsible for supervision of the user-interface and redirection of data depending on the requests of the user (Sec. 5.6). This complex series of tasks is coordinated through an event-handler which interrupts the host from its current background activity in response to user requests. In some cases, where data transfer between processors is taking place, a user request may have to be put on an event queue whilst the host completes its current job. All executable code and user-selected parameters are downloaded to the DSP via the host. Data transfer from the DSP to the host is initiated by interrupting the host due to the time-critical nature of DSP data memory.

Spectral analysis of data, using the FFT, is usually the first operation to be performed and is often a preliminary to further processing. However, the FFT is a very computation intensive operation and is the major bottleneck hindering the widespread development of real-time spectrum analysers. The DSP offers several specialised features which make it ideal for computation intensive signal processing applications such as the FFT. Some of its features will be described in the following sections.

5.5.1 Overlapped Data Acquisition and Signal Processing

The DSP handles all data acquisition, storing sampled data in a circular buffer. A record length of 1024 data points is analysed per channel with a 75% window overlap. The *minimum* memory requirement for implementing this overlap scheme on a single channel (Fig. 5.8) is 1280 words, divided into five equal blocks of 256 words (B_0 - B_4). For

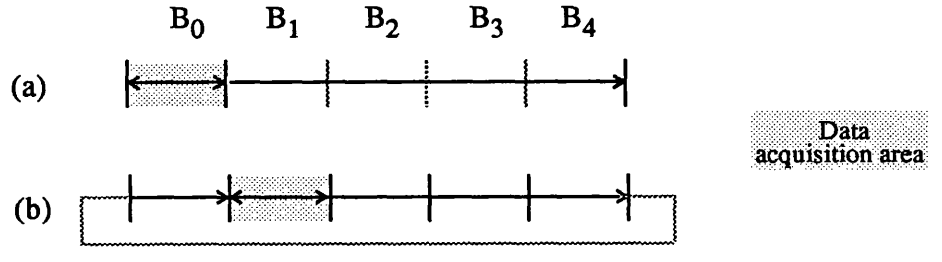


Fig. 5.8 Overlapping sample buffer.

example, blocks B_1-B_4 are assumed to hold past data, whilst B_0 is currently used for storing fresh data. In this case, the age of the data decreases from B_1 to B_4. The signal processing of data contained in blocks B_1-B_2-B_3-B_4 must be completed before the data acquisition buffer B_0 has been filled. Interrupt-driven DSP software is used to implement the data acquisition and signal processing, with the calculations being performed as a background task in between interrupts. The timings for these operations are discussed below. In the next data acquisition cycle, data samples will be stored into memory B_1 whilst blocks B_2-B_3-B_4-B_0 are processed.

The DSP must have performed all processing on the current set of data and transferred the results to the host before the next frame of 256 samples per channel is updated and transferred to the main calculation buffer. With this configuration, data acquisition memory is optimised to be 256 samples per channel. The overlapped FFT scheme is used since it increases the accuracy of spectral estimates (Rabiner and Gold, 1975).

The overall bandwidth of this data acquisition scheme is limited by the following timing relationship:

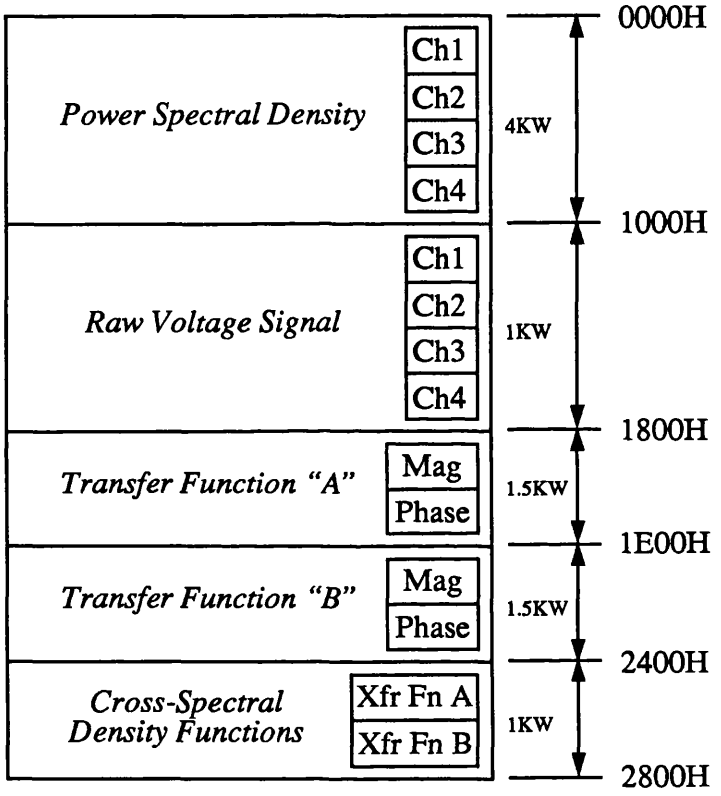


Fig. 5.9 The digital signal processor memory map of data transferred to the Host.

$$t_{DSP} + t_{XFR} + t_{PC} < t_{256} \tag{5.2}$$

where:

- t_{DSP}= time taken to perform signal processing routines ≈ 60 ms,
- t_{XFR}= time taken to transfer data over to the PC ≈ 3 ms,
- t_{PC}= time taken to plot the signal onto the screen ≈ 250 ms,
- t_{256}= time taken to fill 256-point acquisition buffer = $256/f_s$.

The timing for t_{DSP} includes computation of two sets of:

- cross-spectra calculations
- sets of transfer function magnitude squared
- transfer function phase
- transfer function coherency
- the power spectrum of four input channels.

The spectral data transferred to the host (Fig. 5.9) includes 4-channels of raw data and power spectra and two sets of transfer function gain, phase and coherency spectra.

Thus, $t_{DSP} + t_{XFR} + t_{PC} = 60 \text{ ms} + 3 \text{ ms} + 250 \text{ ms} < t_{256}$

where,
$$t_{256} = \frac{256}{f_s} = \frac{256}{2.56 f_{max}} = \frac{100}{f_{max}} \tag{5.3}$$

The maximum signal bandwidth that can be captured is approximately 320 Hz. The main limitation is due to t_{PC} which is imposed by the graphics system. For example, if a graphics coprocessor were used for handling display information (t_{PC} would be negligible) then the overall bandwidth would be f_{max}≈ 5 kHz per channel.

The requirement set out in Sec. 5.2.2 was specified for real-time operation up to a spectral bandwidth of 50 Hz - which is easily achieved with the existing system. The bandwidth is user selectable over the range 1-50 Hz, providing spectral resolutions of the order 0.001-0.050 Hz. This includes plotting a signal onto the screen at intervals determined by the sampling rate (Table 5.1).

5.5.2 FFT Implementation

The fast Fourier transform (FFT) provides an efficient method for calculating the spectral properties of a signal. The maximum frequency which can be analysed is determined by the sampling rate, and this together with the data record length determines the size of Fourier transform required. Since relatively high spectral resolution was required, a 1024-point FFT is performed. Over the range of HFJV frequencies (10 Hz) this will give a

frequency resolution of 0.025 Hz. The frequency ranges given in Table 5.1 applies to all four channels. The display resolution is fixed at 400 lines.

As shown by the calculations of the previous section, the speed at which the FFT is executed plays a crucial part in determining the overall bandwidth of signals that can be analysed in real-time. Papamichalis and So (1986) have benchtested the performance of several FFT algorithms implemented on a TMS32020 device, and have shown that a 1024-point radix-2 FFT takes 32 ms. The DSP needs to perform several computations and the time required is summarised in the definition of t_{DSP} in Sec. 5.5.1. To obtain the required spectral resolution over the desired bandwidth meant that a high-performance FFT algorithm had to be implemented using in-line assembly coding. The technique uses an N-complex point transform and a mixture of radix-2, radix-4 and decimation in time (DIT) butterflies to perform the FFT.

Bandwidth (Hz)	*Sampling Rate (Hz)*	*Resolution (Hz)*
1	2.56	0.0025
2	5.12	0.0050
5	12.8	0.0125
10	25.6	0.0250
20	51.2	0.0500
50	128.0	0.1250

Table 5.1 Range of sampling rates and spectral resolution over selectable bandwidths.

The algorithm achieves its intended purpose of fast computation of the FFT by taking advantage of the internal architecture of the TMS320C25 device and its special bit-reversed addressing mode. A 1024-complex point FFT was implemented which achieved a performance matching that reported for a 256-complex point FFT on a TMS32020 device. To achieve such a performance requires customised tuning of the FFT algorithm. For example, since the first two stages of an N-point FFT involves multiplications with known constants, stages 1 and 2 were implemented in a single-pass with a special radix-4 butterfly. Stages 3-8 are implemented by repeated execution of a 256-complex point FFT kernel using the DSP on-chip RAM for maximum efficiency.

The FFT algorithm is based on an in-place, DIT, radix-2 implementation. Assuming $N = 2^n$, where n = total number of stages of decimation then N = 1024 and n = 10. The special radix-4 butterfly is derived for this case.

Thus, the butterfly operation at any given stage m of the FFT can be defined as:

$$X_{m,i}(k) = X_{m-1,2i}(k) + W_\alpha^k X_{m-1,2i+1}(k) \qquad 0 \le k \le \frac{\alpha}{2} - 1$$

$$X_{m,i}(k) = X_{m-1,2i}(k) - W_\alpha^{k'} X_{m-1,2i+1}(k) \qquad \frac{\alpha}{2} \le k \le \alpha - 1$$

where $W_\alpha^k = e^{-j2\pi k/\alpha}$, $\alpha = N/2^{n-m}$, $k' = k - \alpha/2$, $1 \le m \le n$ and $X_{m,i}$ represents the decimated samples after stage-m of the FFT algorithm $0 \le i \le (N/2^m) - 1$. The DIT redundancy reduction is achieved by dividing the input sequence into odd and even sample sequences. Thus, at stage-1, an N-point sequence is transformed by combining the DFTs of these two N/2-point sequences. The index i represents the number of samples at the input to each sequence. This value of i decreases through each stage of the FFT, until the final stage where the DFT operation is reduced to a single butterfly operation.

After stage-1 of the FFT algorithm we have:

$$X_{1,i}(k) = X_{0,2i}(k) + W_\alpha^k X_{0,2i+1}(k) \qquad k = 0$$

$$X_{1,i}(k) = X_{0,2i}(k') - W_\alpha^{k'} X_{0,2i+1}(k') \qquad k = 1,\ k' = 0.$$

for $0 \le i \le (N/2^1) - 1$, $\alpha = N/2^{n-1} = 2$, where m = 1, n= 10 and N = 1024.

$$X_{1,i}(0) = X_{0,2i}(0) + X_{0,2i+1}(0)$$

$$X_{1,i}(1) = X_{0,2i}(0) - X0_{,2i+1}(0)$$

for $0 \le i \le 511$, since $W_\alpha^0 = 1$.

After stage-2 of the FFT algorithm we have:

$$X_{2,i}(k) = X_{1,2i}(k) + W_\alpha^k X_{1,2i+1}(k) \qquad 0 \le k \le 1$$

$$X_{2,i}(k) = X_{1,2i}(k') - W^{k'} X_{1,2i+1}(k') \qquad 2 \le k \le 3,\ k' = k - 2$$

for $0 \le i \le (N/2^2) - 1$, $\alpha = N/2^{n-2} = 4$, where m = 2, n = 10 and N = 1024.

Thus the first two stages of an N-point FFT can be performed simultaneously with a special radix-4 butterfly which avoids multiplication operations to enhance execution speed (Fig. 5.10):

$$X_{2,i}(0) = X_{1,2i}(0) + X_{1,2i+1}(0)$$

$$X_{2,i}(1) = X_{1,2i}(1) - jX_{1,2i+1}(1)$$

$$X_{2,i}(2) = X_{1,2i}(0) - X_{1,2i+1}(0)$$

$$X_{2,i}(3) = X_{1,2i}(1) + jX_{1,2i+1}(1)$$

for $0 \le i \le 255$, since $W_4^0 = 1$ and $W_4^0 = -j$.

This butterfly can be expanded for the first four input samples to produce:

$$X_{2,0}(0) = X_{1,0}(0) + X_{1,1}(0) = [X_{0,0}(0) + X_{0,1}(0)] + [X_{0,2}(0) + X_{0,3}(0)]$$
$$X_{2,0}(1) = X_{1,0}(1) - jX_{1,1}(1) = [X_{0,0}(0) - X_{0,1}(0)] - j[X_{0,2}(0) - X_{0,3}(0)]$$
$$X_{2,0}(2) = X_{1,0}(0) - X_{1,1}(0) = [X_{0,0}(0) + X_{0,1}(0)] - [X_{0,2}(0) + X_{0,3}(0)]$$
$$X_{2,0}(3) = X_{1,0}(1) + jX_{1,1}(1) = [X_{0,}0(0) - X_{0,1}(0)] + j[X_{0,2}(0) - X_{0,3}(0)].$$

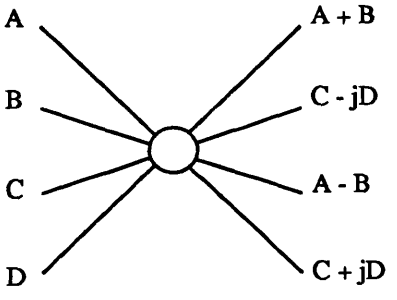


Fig. 5.10 Special radix-4 butterfly.

Later stages of the FFT require multiply-and-accumulate operations on raw-data with non-integer W_α^k *twiddle factor* terms. To maximise the execution speed, the twiddle factors are pre-computed and stored in a look-up table. The butterfly operations are implemented as macro-calls so that the main computation loop comprises in-line code. This provides a further speed advantage by eliminating the overhead associated with conditional branch instructions. Wherever possible, special butterfly operations are used for integer-valued W_α^k terms. Automatic scaling (divide by two for radix-2 and four for radix-4) is incorporated into butterfly operations to avoid any possibility of overflow. Fig. 5.11 shows the flow diagram of the 1024-point FFT kernel based on an in-place DIT algorithm mentioned earlier. The computation loop between stages 3-8 are processed using a smaller 256-complex point FFT kernel which is executed in the DSP on-chip memory. This scheme provides a marked increase in performance since all on-chip memory accesses are single cycle instructions. Data is transferred between external and internal memory at high speed via the pipeline capability of the TMS320 device.

The time required to compute a 1024-real point FFT and its squared magnitude terms using the developed algorithm is 8 ms. This is a four-fold increase in performance compared with the in-line FFT algorithm reported by Papamichalis and So (1986), and an eight-fold increase over that of Burrus and Parks (1985). The latter workers have implemented a similar algorithm in Fortran on a PDP 11 and achieved a timing

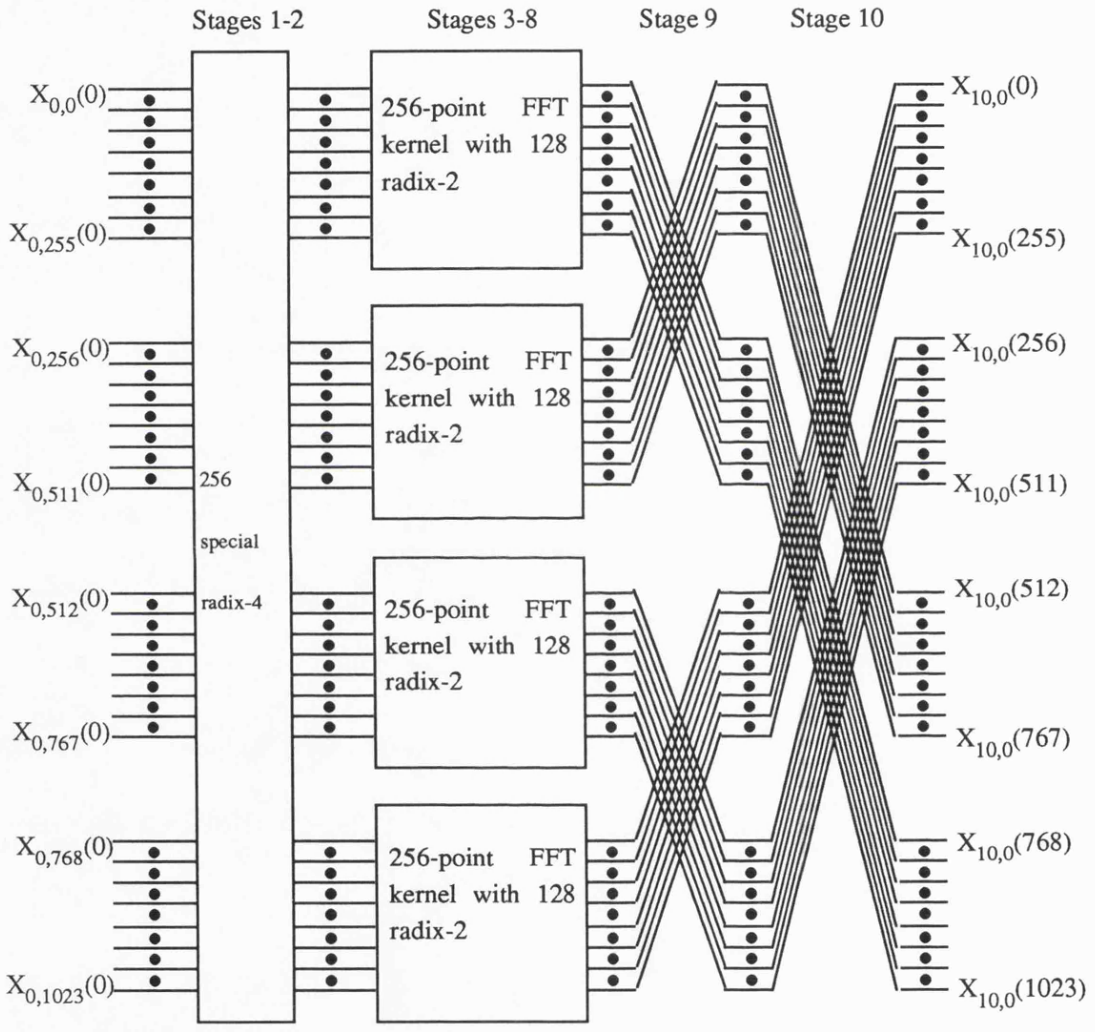


Fig. 5.11 1024-point decimation-in-time radix-2 FFT kernel.

performance of 1400 ms. Table 5.2 provides timings for a 1024-point complex FFT implemented on different systems.

Hardware	*Execution Time (milliseconds)*
TMS320C30 (33MHz)	3.750
TMS320C25 (40MHz)	15.552
Sun 3/260 (25MHz)	104
MicroVAX II (VMS)	302
Compaq 386 (16MHz)	798
IBMPC-AT(12.5 MHz)	1250
PDP-11	1400

Table 5.2 Execution time for a 1024-point complex FFT on different machines.

5.5.3 Transfer Function Analysis

One of the many analytical advantages of working in the frequency-domain is that the integral equations relating the stimulus and response of a system in the time-domain become algebraic equations in the frequency domain. This facilitates the description of both linear systems and nonlinear ones alike.

The original specification set out in Sec. 5.2.2 was that several analytical tools be incorporated into the instrumentation for performing system identification. Since this type of analysis is computationally taxing, it was implemented on the DSP. The theoretical background for the signal processing used in this chapter is given in Sec. 2.6.

The DSP has been programmed to compute the power spectrum of each input channel and two sets of transfer function calculations and coherency spectra. Each transfer function comprises a modulus and phase spectrum. Also available are two sets of cross-covariance functions which can be used for calculating Nichols and Nyquist plots. These calculations are all performed in real-time and the memory map of Fig. 5.9 shows the format in which the data is made available to the host. The transfer function calculations developed around the theory presented in Sec. 2.6.3. Since the DSP is a fixed-point device with no support for trigonometric functions, an efficient algorithm for computing the transfer function phase spectrum was required. The remainder of this section develops an inverse-tangent look-up table method for rapid computation of the phase-angle of a complex number. The phase angle θ of a complex number of the form $z = (x + jy)$ can be calculated using the trigonometric relation $\tan\theta = \frac{y}{x}$.

Using the Q_{15} number notation defined in Rabiner and Gold (1975), we can represent $-180° \leq \theta \leq +180°$ as an integer in the range $-32767 \leq \theta \leq 32767$.

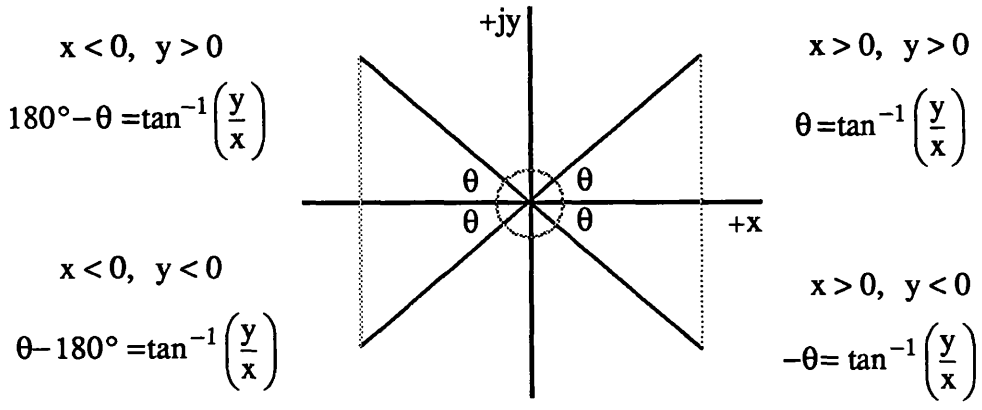


Fig. 5.12 A method for computing the argument of a vector.

The Argand diagram of Fig. 5.12 shows a method for computing the phase θ of the vector $(x + jy)$ depending on the quadrant it occupies. Hence, by defining $0° \leq \theta \leq +89°$ we can calculate θ for any value in the $-180° \leq \theta \leq +180°$. Using a look-up table of 2048 θ-values over the range $0° \leq \theta \leq +89°$, we can achieve a maximum phase resolution of 0.044°. A system had to be devised which could be used to compute $\theta = \tan^{-1}\left(\frac{y}{x}\right)$ from values of the ratio $\left(\frac{y}{x}\right)$:

$$\left(\frac{y}{x}\right) = \left(\frac{57.9n}{2048}\right) \tag{5.4}$$

Since $0 \leq \tan\theta \leq 57.29$ over the range $0° \leq \theta \leq +89°$ and $0 \leq n \leq 2047$.

The phase look-up table values are calculated using:

$$\theta_n = \left(\frac{16383}{90°}\right)\tan^{-1}\left(\frac{y}{x}\right) = 182.04\tan^{-1}\left(\frac{n}{35.75}\right) \tag{5.5}$$

Hence, the correct phase angle for a given ratio of $\frac{y}{x}$ is found by calculating the index value n, where $n = \left(\frac{36y}{x}\right)$.

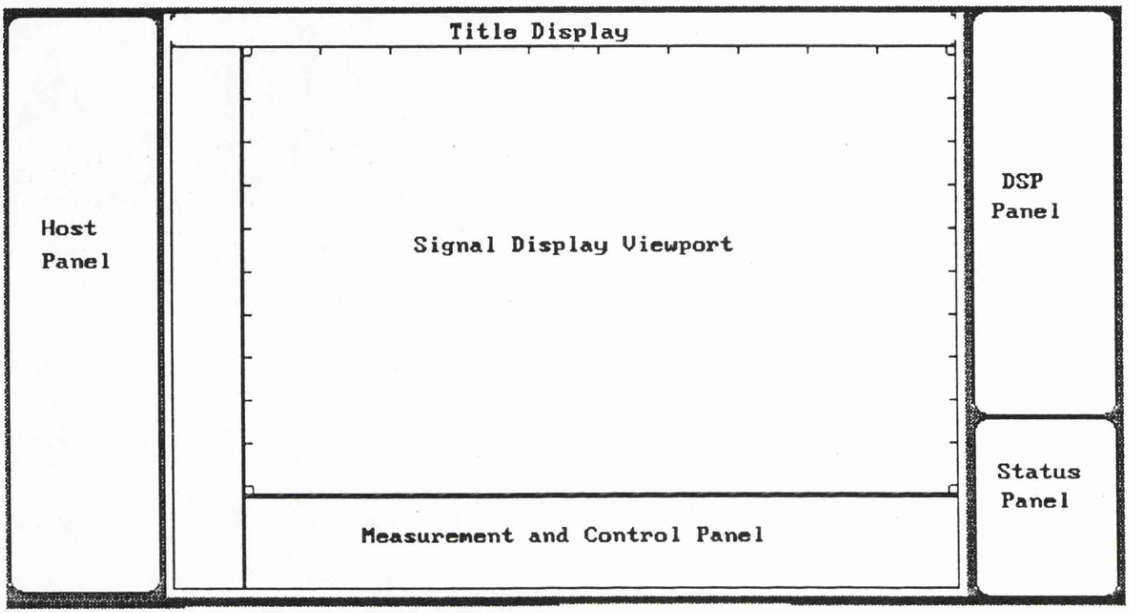


Fig. 5.13 The instrument soft panel template.

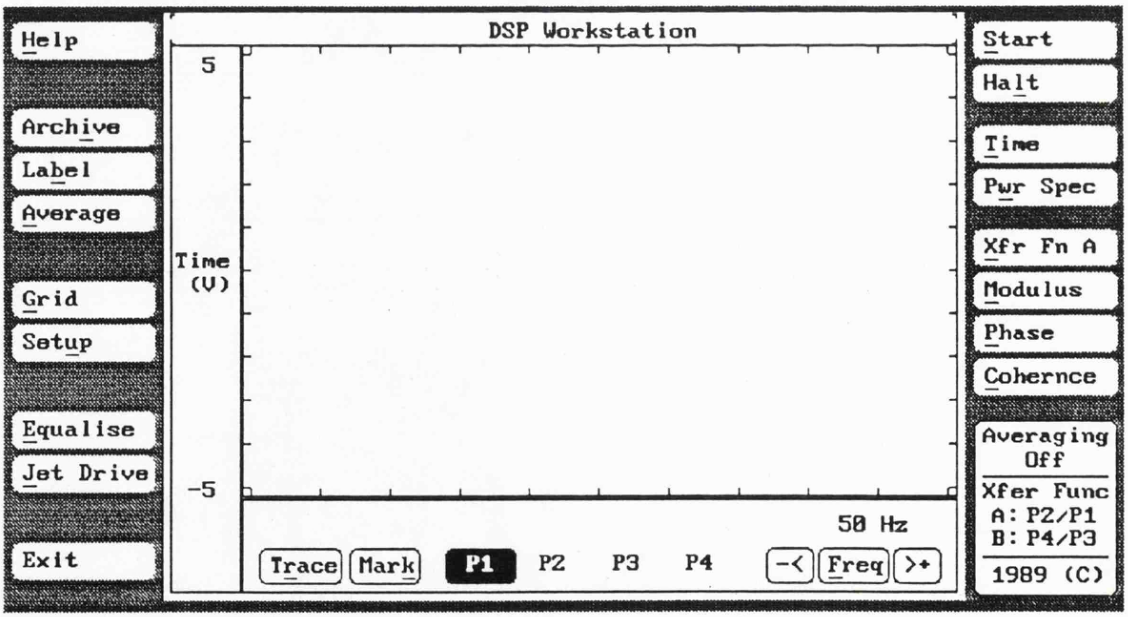


Fig. 5.14 The instrument panel.

5.6 Man-Machine Interface

The man-machine interface is used to mask the advanced system architecture that specifies how measurements and signal processing are structured and managed within the computer instrument. All tasks and basic functions are orchestrated by a generic kernel which has access to many facilities and data resources resident on the host.

The man-machine interface was a major design feature that required significant investigation before fundamental characteristics could be identified. For example, the interface must be:

- practical and must work well on available hardware and capable of transferring data to other software packages such as databases, spreadsheets and wordprocessors;
- flexible and extendible to support processing and control applications that may be required in the future;
- easy-to-learn and easy-to-use such that the learning-curve is kept to a minimum, and once users are proficient with the system they are not hampered by an inflexible user interface.

These objectives have been fulfilled and wherever possible exceeded by the man-machine interface described in the remainder of this chapter.

The use of a general-purpose personal computer as the primary interface to specialised instrumentation is the philosophy that has been adopted throughout this study. Instruments traditionally have been contained in a dedicated box with a display and buttons on the front-panel for control and measurement. The interface described here permits for *faceless instrumentation* to be centrally controlled via a graphics-based front-panel implemented on the screen of a personal computer.

A *soft-panel* program was developed to provide the user with an interactive graphics mechanism for manipulating the instrument. The soft-panel display mimics the buttons and displays seen on a normal instrument front panel (Fig. 5.14). The screen partitioning (Fig. 5.13) adopts a consistent format to improve the functionality of the instrument. The main elements of the user-interface are divided into button, status and display panels including:

- an instrument title area reserved for displaying a character-based title;
- a DSP button panel for activating data acquisition and signal processing functions;
- a host button panel for controlling system configuration and archive management;
- a signal viewport for plotting signals with axes and grids that are automatically labelled in the units of measurement. This viewport is also the area in which pop-

up windows appear for functions that require further information;
- a measurements control and status panel for selecting bandwidth of measurements and tracing a cursor across the signal while their respective X-Y coordinates are displayed in the status area;
- a status panel which provides information about signal averaging and the transfer function relationships between input channels.

The menu system (Fig. 5.14) comprises a number of task-oriented buttons. All DSP panel buttons have an LED which is activated when a button is selected. For example, if the LEDs associated with the *Start* and *Pwr Spectra* buttons are active, this should inform the user that the system is currently acquiring data and displaying the power spectrum of the channel currently shown to be active in the measurements and control panel. Hence, the man-machine interface manipulates graphic objects to display the current state of the instrument in a concise and efficient manner.

A *ganged status display* is used to show the current waveform to be active in the signal viewport. The input ports are labelled P1 to P4, where the active port is highlighted by inverting its video attributes. The left/right arrow keys can be used to select the desired input port for display on the signal viewport.

The host and DSP buttons are activated by an associated *hot-key* which is clearly underlined on the button label. User inputs are all passed to an interrupt-driven event handler which will store the button status onto a FIFO event-queue. As soon as the host is freed from higher priority activities, it will read the contents of the event-queue and execute the selected button function. In this way, the user may select any sequence of buttons with the knowledge that they will be executed in sequence by the host.

Complex multi-mode instruments that use traditional front-panels have to choose between a) implementing a large number of controls and displays, many of which are not relevant to an individual measurement and only contribute to front-panel clutter, or b) using the same controls and displays for multiple functions, each of which requires its own graphics screened in different colours on the panel. Both of these options are potentially confusing to the user.

The man-machine interface developed for this study provides a mechanism whereby only those components that are relevant to the current measurements are displayed. For example, if a button requires further information, a context-sensitive window manager will generate a pop-up window with a message of the required actions and a series of buttons to perform those actions. Once the correct parameters have been set, the window manager will shut-down the window and restore the screen and machine to its new state. This scheme avoids major re-definition of the screen and thereby avoids a complex appearance to the system.

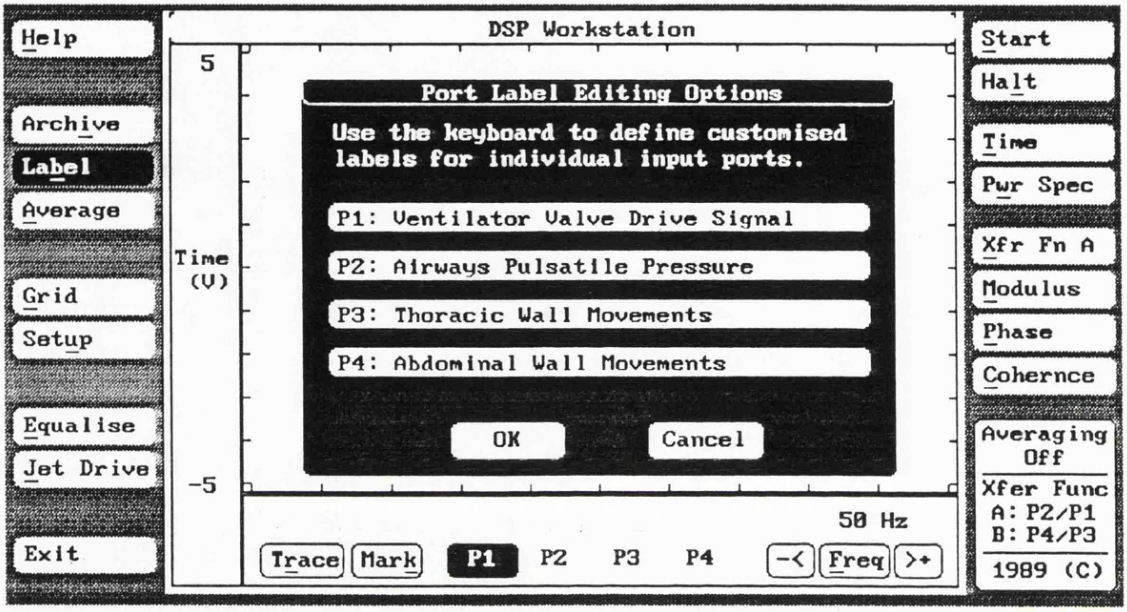


Fig. 5.15 The Label Menu.

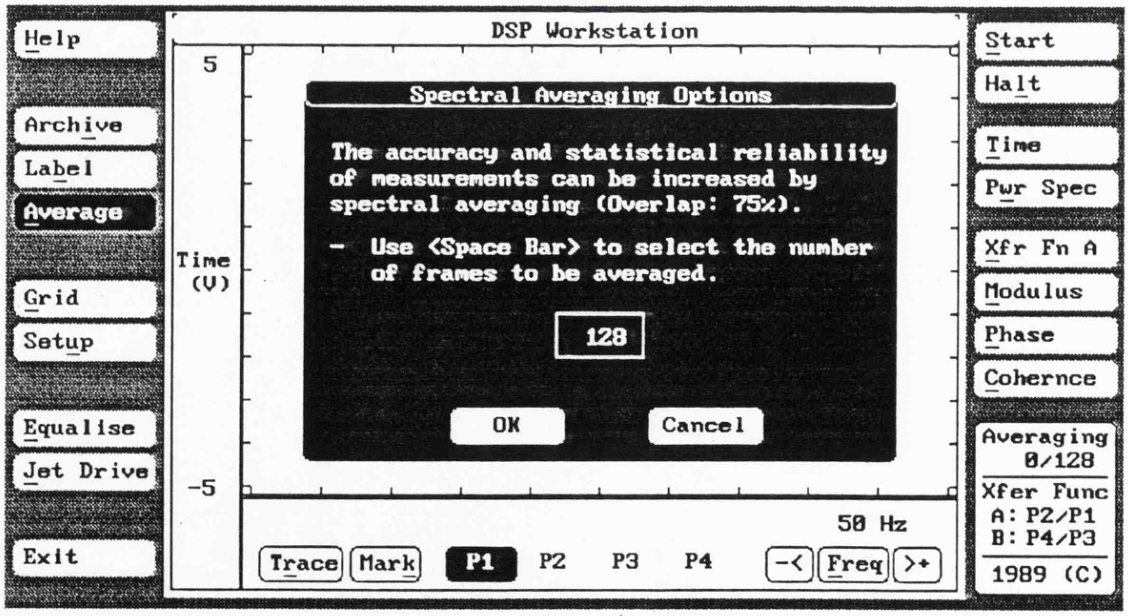


Fig. 5.16 The Average Menu.

The following sections provide a brief description of the available options.

5.6.1 Host Software Organisation

The host software consists of some generic code and some button-specific code. The generic component implements the system kernel, window management, softkey processing, display handling, and a variety of library functions accessible by the instrument specific code. Other host duties include management of the DSP and data acquisition.

The level of interaction between host and DSP has been kept minimal to avoid the complexity involved in developing a formal communications protocol (Sec. 3.10). Data transfers are direct to the DSP memory via the dual-port memory. The main control duties of the host are to:

- start and halt data acquisition,
- set DSP sampling rate and frequency pre-scaler values,
- set the channels to be used for the transfer function calculations.

The DSP interrupts the host when data needs to be transferred for display to the user or storage onto disk. The format in which data is transferred is shown in Fig. 5.9. The timing for this transfer operation was discussed in Sec. 5.5.1.

5.6.2 Host Button Panel

A. Label Button

The *Label* button activates the pop-up menu shown in Fig. 5.15 and is used for annotating the input channels. This is a useful facility since each input port can be given a unique label identifying its source or other characteristics. A line editing capability allows for labels to be composed from the system keyboard; a delete, insert, clear and scroll edit facility is incorporated into this menu. The user may switch between labels by using the up/down arrow keys or simply by entering a carriage return. The current label is always highlighted when it becomes active for editing and a cursor appears to show the text entry position. The port labels are automatically saved, together with any experimental data, following an archive operation (Sec. 5.6.2D). This has the advantage that other users can easily keep track of the input sources selected during a given experiment.

B. Average Button

The advantages of signal averaging were discussed in Chapter 2 and can be activated by selecting the *Average* button (Fig. 5.16). The instrument implements a scheme known as spectral averaging. This technique can be used to improve the signal-to-noise ratio of measurements in accordance with the theory set out in Sec. 2.6. The number of frames to

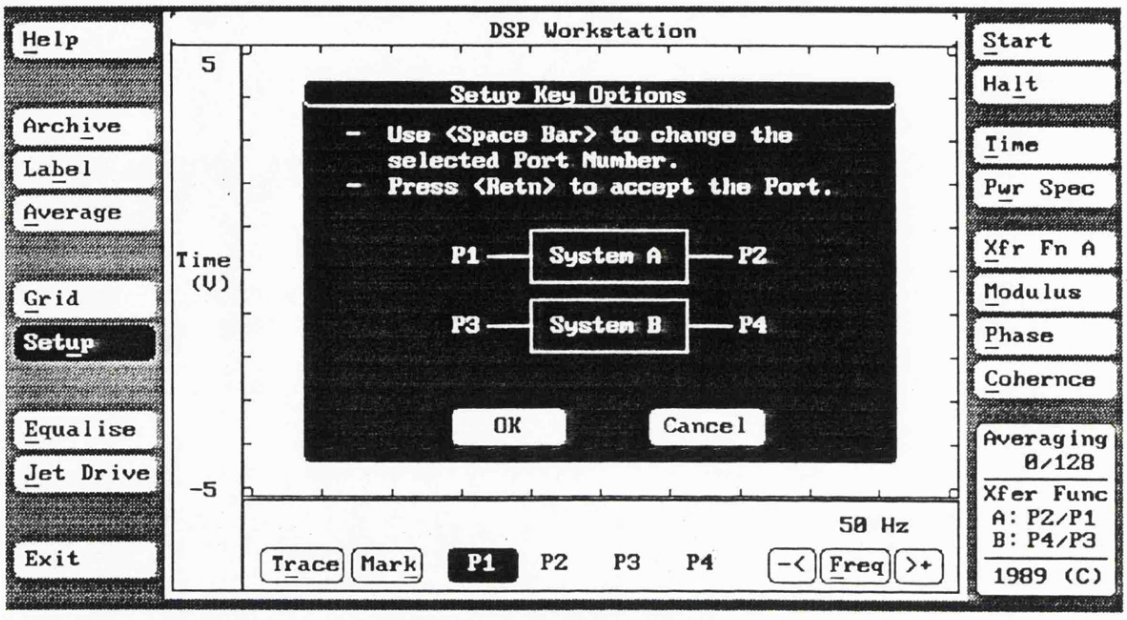


Fig. 5.17 The Setup Menu.

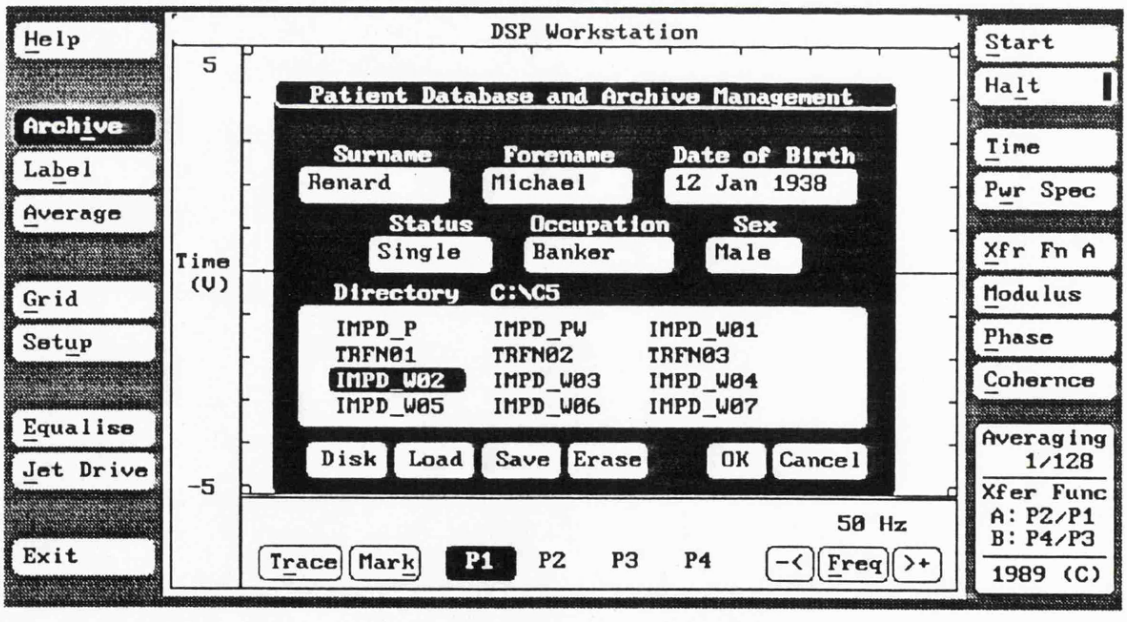


Fig. 5.18 The Archive Menu.

be averaged can be selected as any integer power of 2 falling in the range 2-2048; this provides the user with the flexibility to specify the required degree of accuracy. The averaging scheme also implements a 75% overlap facility (Sec. 5.5.1) to minimise errors introduced by the spectral estimation process. The total number of frames to be averaged is shown in the status panel (Fig. 5.13) together with sub-total of averaged frames at any given instant. The data acquisition software monitors the status of averaging and automatically halts acquisition when the number of frame averages equals the selected number of total averages.

C. Setup Button

The instrument has the capability to perform two sets of transfer function calculations. The *Setup* menu (Fig. 5.17) provides a simple object-based approach to defining the channels to be selected for the transfer function calculations. The menu comprises two *black-box* systems with input stimulus to the left and output response to the right. The user can select any permutation of input/output by following the instructions given in the menu. The status panel shows the defined transfer function relationships that are currently being performed by the DSP.

D. Archive Button

The *Archive* button (Fig. 5.18) is selected when data needs to be saved, recalled or deleted from disk. The upper half of the *Archive* menu contains a database system with a number of fields for storing personal patient information. The database makes use of two files for storing experimental results. First, a patient-details file is used for logging the configuration of the system active during the experiment. Second, a data file is used to store the results of data captured and processed during the experiment. Both files are stored in a format such that further analysis can be easily carried out by transferring the data to commercial spreadsheet or database packages. The lower half of the menu displays a sub-window containing the existing archive files together with the current drive and sub-directory of the archive. The user can recall a file by selecting it with the special highlight-cursor (e.g. the cursor is seen to highlight the file IMPD_W02 in Fig. 5.18). The cursor can be used to scroll through the archive directory to select a given file. Where the number of files exceeds the display capacity of the sub-window, the cursor keys can be used to scroll through a new page of files. The *Archive* menu also contains a number of self-explanatory buttons to *Load*, *Save* and *Erase* files. The *Save* button produces a pop-up window (Fig. 5.19) requesting a filename for storing the data. Confirmation and diagnostic messages are displayed to the user to prevent inadvertent loss of data.

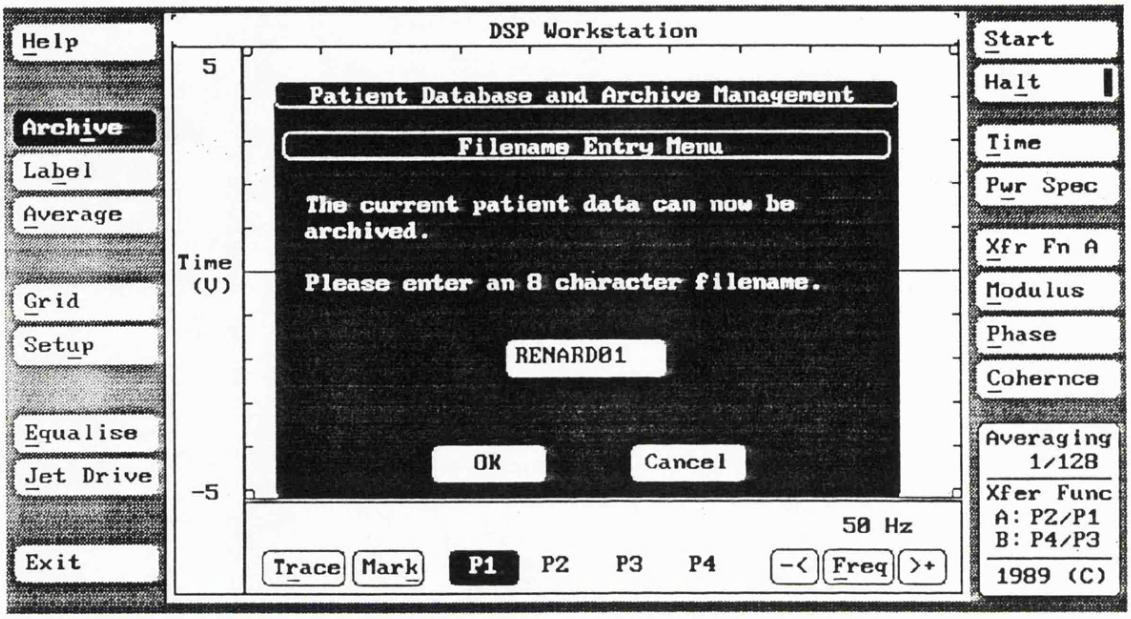


Fig. 5.19 The Save Menu.

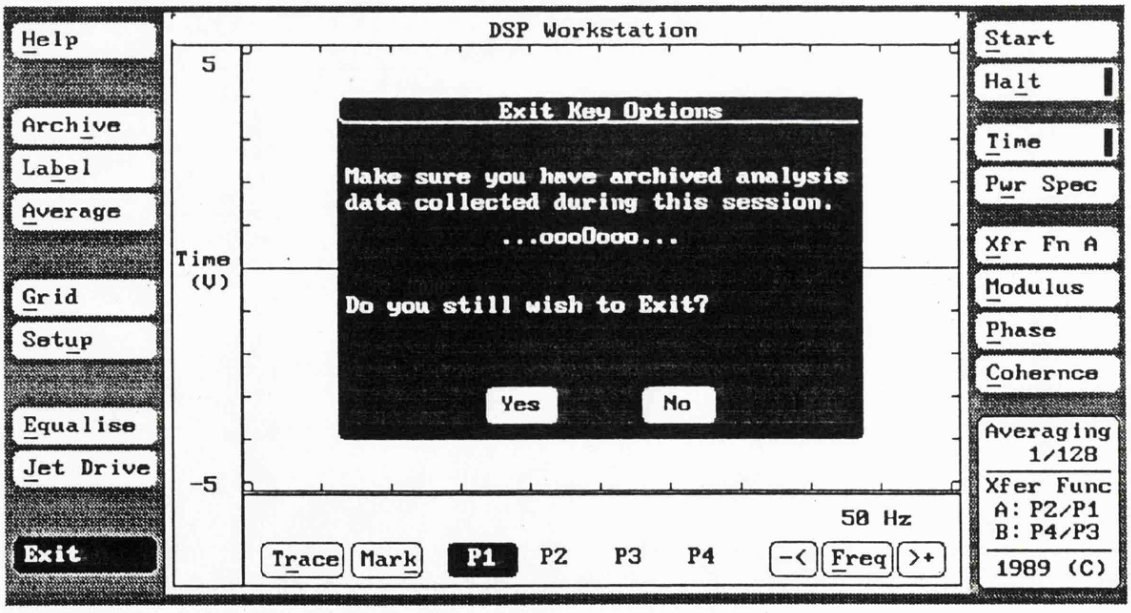


Fig. 5.20 The Exit Menu.

E. Miscellaneous Buttons

Other host functions on the instrument include:

- a context-sensitive *Help* button to provide guidance to the novice user;
- an *Equalise* button to provide automatic frequency response compensation for transducers and other measurement devices;
- a *Jet Drive* button which provides a future option to implement adaptively controlled high-frequency jet ventilation;
- the *Exit* button to shutdown and terminate the program (see Fig. 5.20).

Context sensitive help refers to information that applies to a specific situation at a particular moment, as opposed to general information that is non-specific. The user-interface has been designed to be intuitive, where the user is given a small hint or short explanation to describe a function or explain the required action to minimise delays or distraction from the current task.

5.6.3 DSP Button Panel

All data acquisition and signal processing functions are controlled by the DSP button panel. The theoretical basis for the signal processing function are covered in Secs. 5.4-5.5 inclusive. All buttons within the DSP panel have an LED which *lights-up* whenever a button is selected. The *Start* and *Halt* buttons are used to control data acquisition on the DSP. Captured data can be displayed to the screen in many formats, depending on the type of signal processing that is currently active. The available signal processing functions include:

- a voltage-time display of input signals by selecting *Time*;
- power spectrum calculations of the input signals using *Pwr Spec*;
- transfer function analysis of signals using *Modulus*, *Phase* and *Coherence*;
- the *Xfr Fn* button is used to switch the transfer function displays between system A and system B.

The user can select between all the different display formats irrespective of the status of the *Start* and *Halt* buttons. The button LEDs together with the graph annotation indicate the type of signal on display (including vertical axis units and any scaling factors). This formatting is all handled automatically and tracks the DSP button selections.

5.6.4 Measurements Control and Status Panel

The *Trace* and *Mark* buttons allow the user to scroll a marker along the path of any waveform active in the signal viewport. The X-axis and Y-axis values for any point on the waveform are automatically displayed in the measurements panel along with the correct units. The *Mark* button is useful for setting a reference point on the waveform

from which relative offsets (and absolute values) of time, frequency, and amplitude can be measured. A *Grid* button is used to superimpose a grid array onto the signal for easy reference to coordinate values.

A ganged status panel highlights the particular input port to which the current time-domain or power spectrum screen plot belongs. Input channels are labelled as ports P1 through to P4. This display is invalid for transfer function plots. The left/right arrow keys may be used to select the desired input port for display on the signal viewport.

The bandwidth of measurements is selected via the *Freq* button which allows the user to scan through a range of frequencies from DC ~ 1 Hz to DC ~ 50 Hz. Table 5.1 summarises the range of sampling rates and attainable frequency resolutions for a given bandwidth selection.

5.7 Transducers for Monitoring Respiratory Data

This section provides a brief overview of the types of transducers used in this study. The respiratory variables of interest include airways pressure, flow and abdominal/thoracic wall displacements (Sec. 5.2.2). The first two variables are relatively easy to monitor using standard techniques. Airways pressure can be directly monitored using the strain-gauge pressure transducer aboard the Penlon jet ventilator. Instantaneous respiratory air flow can be monitored using a Fleisch pneumotachograph head (Gould, Bilthoven) connected to a capacitive differential pressure transducer. Abdominal/thoracic wall displacements, however, are not so easy to monitor (Sec. 5.2.3).

The traditional method of measuring chest wall movement is based on a strain gauge device (Lectromed Respiration Transducer, type 4320, flat frequency response to 30 Hz with 36 dB/octave rolloff) mounted on an non-elastic belt which straps closely around the patient. The belt is fastened to each end of the strain gauge to form a closed loop. Chest wall movements associated with respiration alter the tension in the belt, and thereby produces a change in strain. These changes in strain are translated into a signal that is related to the volumetric change in the patients' abdomen.

Alternative methods include the inductive plethysmograph belt (Respitrace Ambulatory Monitoring) and the Glasgow inductive vest (Hanning et al., 1978). For example, the latter transducer produces an output that is related to the variation in cross-sectional area of the coils (and hence, variations in inductance) due to respiration. Both techniques provide a well-established quantitative method for measuring ventilation in the form of tidal volume (V_E) (Millman et al., 1986) and respiratory flow (Hill et al., 1982). However, they suffer from several disadvantages. The main problems are:

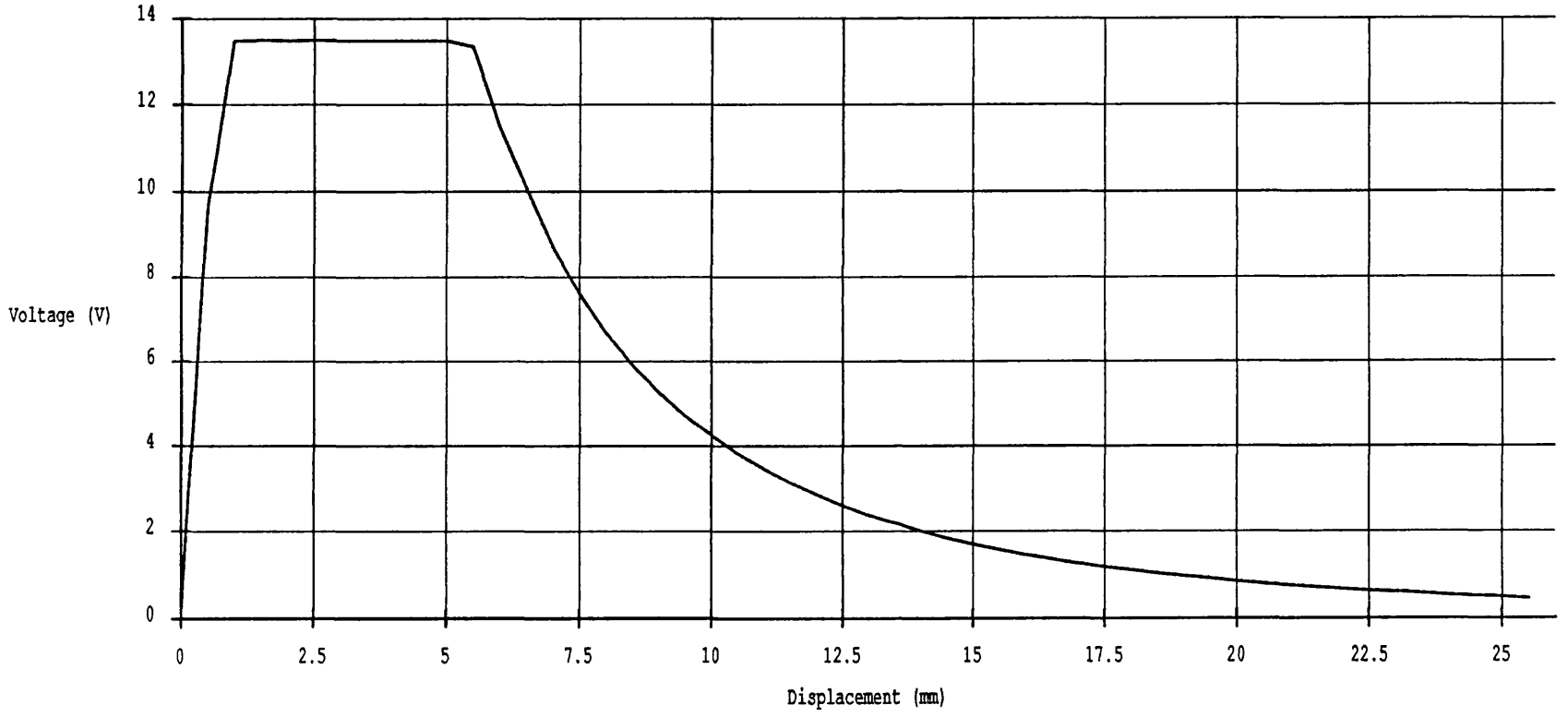


Fig. 5.22 Static calibration curve of fibre-optic displacement transducer.

- since these devices must be worn be strapped around the patient, a) the overall chest wall impedance is artificially increased, restricting respiration; and b) patient access is obstructed;
- average changes in volume are measured as opposed to direct measurement of chest wall displacement;
- measurements can vary up to 20% depending on transducer placement, hence repeatability has an error factor of 20%;
- calibration is required prior to any measurements;
- the dynamic response of the system is limited to breathing frequencies (no dynamic characteristics are reported in the literature);
- heartbeat artefact is a common source of error which must be filtered from measurements;
- electrical isolation of the patient can be compromised.

Alternative methods of measuring chest wall displacement were considered to overcome the problems listed above. In particular, an ultrasonic displacement transducer of the type used in auto-focusing cameras was considered. However, despite its wide bandwidth, the method had to be rejected since the maximum resolution was limited to 2.5 mm. This was too restrictive considering that chest-wall displacement is typically over a dynamic range of 0-15 mm.

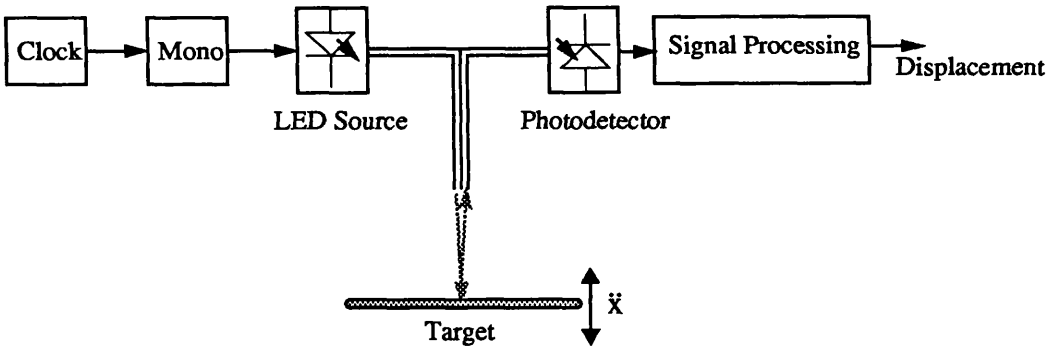
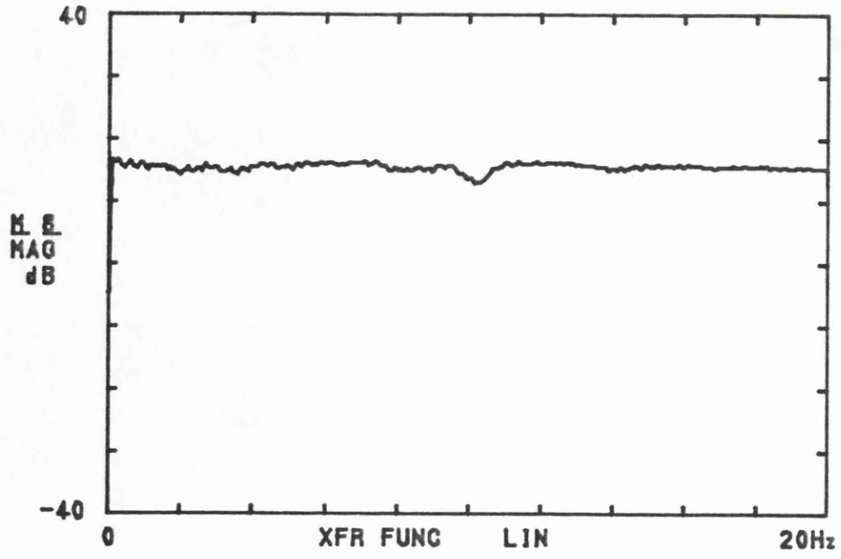
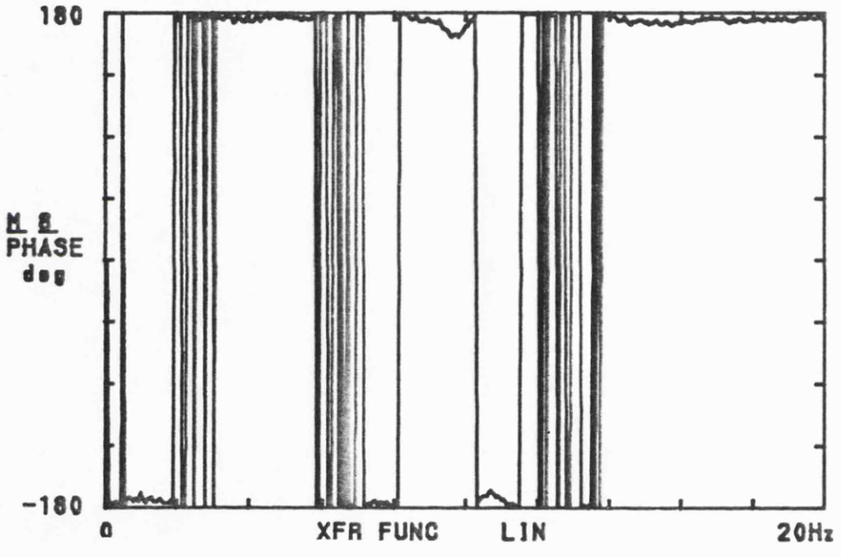


Fig. 5.21 Fibre-optic displacement transducer.

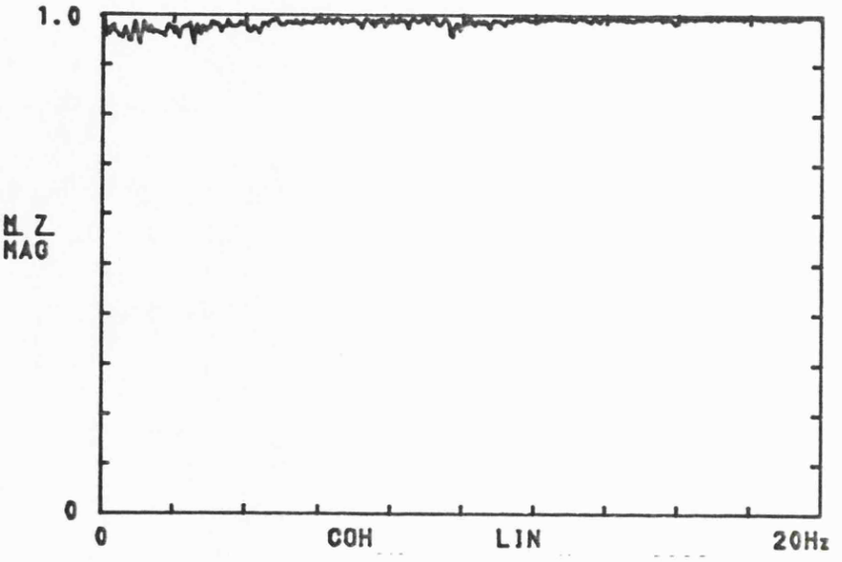
The method finally adopted was the fibre-optic displacement transducer of Dahnoun (1987). The sensitivity and range of the device has been modified for measuring chest wall displacements. This technique was selected since it offered versatility, relative freedom from artefact, high resolution and good dynamic performance. The system (Fig. 5.21) operates by decoding the amplitude modulation produced by reflecting a high frequency light source from the target. A light-emitting-diode (LED) source is modulated to emit a 10 KHz square wave carrier into an optical waveguide. Modulating the optical carrier eliminates artefact due to ambient lighting. The signal received by the photo-detector is an amplitude modulated version of the source signal. A signal processing stage is then used to extract the displacement of the target from the received signal.



(a) Magnitude Spectrum



(b) Phase Spectrum



(c) Coherence Spectrum

Fig. 5.23 Dynamic response of displacement transducer.

The static response of the displacement transducer was tested against a high-precision linear potentiometer device which is linear over the range 0-50 mm. The static calibration curve of the fibre-optic transducer (Fig. 5.22) is seen to be nonlinear over the range 5-25 mm displacement. However, assuming piece-wise linearity, the transducer can perform linearly over a selected range. For example, to make measurements over a 10 mm dynamic range, the transducer equilibrium position could be set at a distance of 17.5 mm from the target. Thus, measurements over the range 12.5-22.5 mm are piece-wise linear.

The dynamic response of the transducer was tested on the same rig used for the static measurements, however, the target was excited using a band-limited white noise input. The frequency response of the linear potentiometer is known to have a flat magnitude spectrum and a linear phase response over the range 0-50 Hz. The objective was to determine the frequency response of the fibre-optic transducer by comparing its performance relative to the reference transducer. The results of Fig. 5.23 shows the transfer function between the two transducer outputs. The flat magnitude and phase spectrum over the range 0-50 Hz shows that the dynamic characteristics of the fibre-optic transducer match the reference transducer over the same frequency range. The validity of this assumption is confirmed by a coherence spectrum which is close to unity over the entire frequency range.

The performance of the fibre-optic transducer provides a significant improvement over conventional methods of displacement measurement. The transducer has been used during clinical trials and the results can be seen from the case studies presented in Chapter 6.

5.8 Discussion

The use of HFJV as a form of respiratory support is known to have certain advantages over conventional modes of ventilation. However, a general acceptance of the technique has been inhibited by lack of understanding of the underlying fluid dynamics, a lack of practical guidelines for clinicians on when to apply HFJV and the ventilator settings that should be used for optimal gas exchange.

The instrumentation developed in this chapter provides a system for real-time investigation of respiratory physiology during HFJV. The computer-system, coupled with the modifications made to a commercial jet ventilator, has resulted in a measurement system which can be used to examine respiratory dynamics with extremely high precision in a fraction of the time taken by previous workers. This is achieved with only minimal changes to existing jet ventilation equipment and procedures. The system is intended to cope with the volume of information that needs to be considered during HFJV and the level of complexity that this method of ventilation entails.

The computer-system uses specialised signal processing algorithms to achieve real-time throughput of the high-volume, computation intensive tasks specified for this study. The philosophy adopted throughout this thesis is to transform a general-purpose personal computer into the primary interface to specialised *faceless instrumentation.* The man-machine interface is a major component of such a system, since it must mask the advanced system architecture that specifies the structure and management of resources within the instrument. The man-machine interface proved to be a non-trivial design exercise. The main design objectives have been fulfilled where a graphics-based man-machine interface acts as a central control panel which mimics the buttons and displays of a traditional instrument panel.

Modifications have been carried out on a commercial high frequency jet ventilator to support complex waveshape pressure excitation of patients' airways. The ventilator can now be used to perform white-noise identification of a patient's respiratory system. This technique has the advantage of providing a systematic procedure for characterising the system over all possible inputs and is simple to apply. Preliminary experiments using lung surrogates have shown the effectiveness of the measurement system in a controlled laboratory environment. The observations validate the efficacy of this approach, with good results being obtained over a short period of time.

A fibre-optic transducer has been developed for monitoring abdominal and thoracic wall displacements during HFJV. The transducer provides a significant improvement over conventional methods of thoracic/abdominal displacement measurement, offers versatility, relative freedom from artefact, high resolution and linearity over a wide dynamic range.

The performance of the instrumentation has also been validated in the clinical environment. The next chapter presents the results of clinical trials in case studies of patients with normal and pathological respiratory systems who have required automatic ventilation as part of their therapeutic regimen.

5.9 References

Burrus, C.S. and Parks, T.W. (1985): "*DFT/FFT and convolution algorithms*". Wiley.

Dahnoun, N. (1987): "Flow and cardiac output measurement", *Internal Report*, University of Leicester.

Drazen, J.M., Kamm, R.D., and Slutsky, A.S. (1984): "High frequency ventilation", *Physiol. Rev.*, **64**, 505-543.

Hailstone, J.G., Jones, N.B., Parekh, A., Sehmi, A.S., Watson, J.D. and Kabay, S. (1986): "Smart instrument for flexible digital signal processing", *Med. & Biol. Eng. & Comput.*, **24**, 301-304.

Hanning, C.D., Smith, H.C. and McA. Ledingham, I. (1978): "Clinical application of inductive plethysmography", *Scot. Med. J.*, **23**, 310-311.

Heijman, K., Heijman, L., Jonzon, A., Sedin, G., Sjostrand, U., and Widman, B. (1972): "High frequency positive pressure ventilation during anaesthesia and routine surgery in man", *Acta Anaesth. Scand.*, **16**, 176-187.

Hill, S.L., Blackburn, J.P. and Williams, T.R. (1982): "Measurement of respiratory flow by inductance pneumography", *Med. & Biol. Eng. & Comput.*, **20**, 517-518.

Horowitz, P. and Hill, W. (1983): "*The art of electronics*", Cambridge University Press, 444-445.

Kolton, M. (1984): "A review of high frequency oscillation", *Can. Anaesth. Soc. J.*, **31**, 416-.

Lin, E.S. and Smith, B.E. (1987): "An acoustic model of the patient undergoing ventilation". *Br. J. Anaesth.*, **59**, 256-264.

Millman, R.P., Chung, D.C. and Shore, E.T. (1986): "Importance of breath size in calibrating the respiratory inductive plethysmograph", *Chest*, **89**, 840-845.

Papamichalis, P. and So, J. (1986): "Implementation of fast Fourier transform algorithms with the TMS32020". In: *Digital Signal Processing Applications with the TMS320 Family*, Texas Instruments, p92.

Rabiner, L.R., and Gold, B. (1975): "*Theory and application of digital signal processing*", Prentice-Hall, 386-388.

Sjostrand, U.H. (1980): "High frequency positive pressure ventilation (HFPPV)", *Critical Care Medicine*, **8**, 345-364.

Smith, B.E. (1985): "The Penlon Bromsgrove high frequency jet ventilator for adult and paediatric use", *Anaesthesia*, **40**, 700-796.

Smith, B.E. and Lin, E.S. (1989): "Resonance in the mechanical response of the respiratory system to HFJV", *Acta Anaesth. Scand.*, **33**, 65-69.

Smith, R.B. (1982): "Ventilation at high frequencies", *Anaesthesia*, **37**, 1011-.

Chapter 6
Clinical Patient Data

6.1 Introduction

This chapter presents the results obtained from clinical trials of the instrumentation described in Chapter 5. The data was collected from patients receiving high frequency jet ventilation (HFJV) as part of their therapeutic regimen and are presented as case studies.

The study population comprises 4 patients of different age, sex and medical history. All patients were admitted to the intensive therapy unit (ITU) of various hospitals within the Leicestershire Health Authority (LHA). The study population was limited mainly by:

- *geographical constraints;* although suitable patients were available within the district, it required transporting of instrumentation and trained personnel between the various hospitals at very short notice;
- *ethical constraints;* even when patients were available, there had to be some ethical justification for using HFJV as opposed to conventional methods of ventilation;
- *physical constraints;* in some cases HFJV tends to have an adverse affect on patient alveolar ventilation and conventional modes of ventilation must be used.

Although the measurements reported here are too few to be of any substantial clinical significance, they provide a preliminary insight into the dynamic behaviour of the respiratory system of patients receiving HFJV. A more substantial study is currently under way and it is hoped that a large population of patient data can be accumulated for future analysis.

6.2 Patients and Experimental Methods

Four critically-ill patients were studied after consent from the patient had been obtained and authorisation was given by the LHA district ethical committee. The patients had different respiratory statuses and these are indicated in the case studies that follow. All patients were admitted to the ITU for ventilatory support in preparation for emergency clinical treatment. The patients were initially ventilated to normocapnia using conventional intermittent positive pressure ventilation (IPPV). They were then intubated with a Mallinkrodt Hi-Lo jet endotracheal tube attached to the modified Penlon jet ventilator circuit described in Sec. 5.3.1.

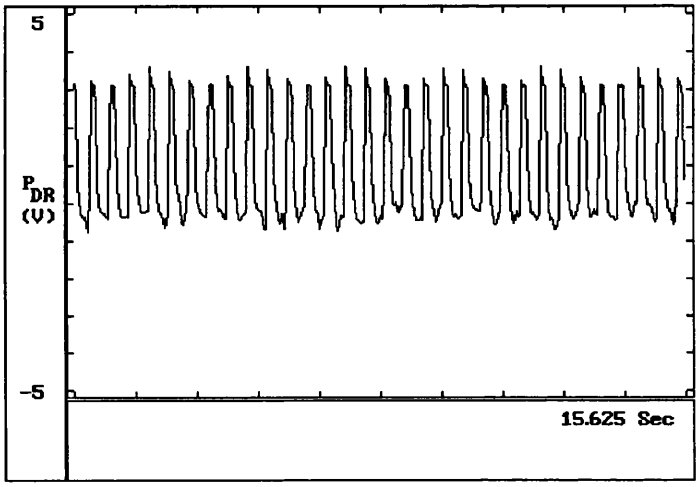
Since the patients were in the ITU, standard clinical monitoring techniques were employed for measuring average blood gas (ABG) pressures. The only deviation from standard practise was that a fibre-optic displacement transducer (Sec. 5.7) was used for monitoring abdominal and thoracic wall oscillations during HFJV. Also, the jet ventilator was operated in mode c (Sec. 5.3.4) such that the jet drive pressure was a linear summation of the standard binary pressure waveform associated with HFJV and a low-amplitude white-noise pressure excitation for performing respiratory system identification. The signals monitored for analysis by the real-time system included:

- jet ventilator drive-pressure output (P_{DP} - PSI);
- patient airways pressure (P_{AW} - cmH_2O);
- chest-wall displacements (δx_{CW} - Volts).

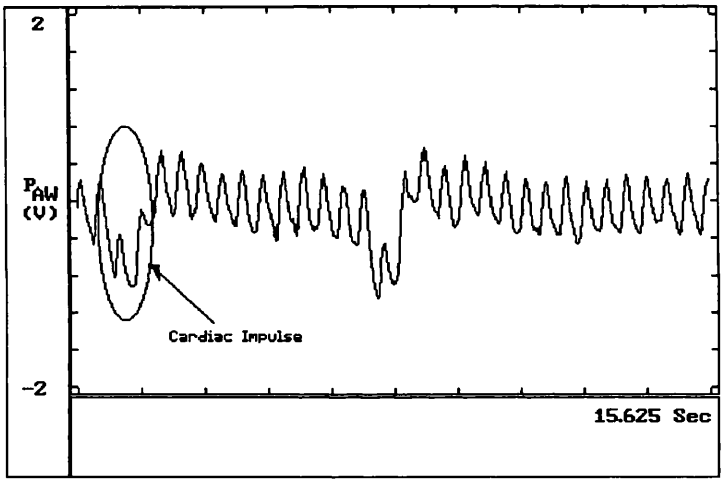
The chest-wall measurements can be converted into absolute units of displacement by using the calibration curves given in Sec. 5.7. These conversions are not performed here, since we are only interested in the dynamic behaviour of the system in relation to other variables.

In some of the cases, a positive-end expiratory pressure (PEEP) of 5-10 cmH_2O is applied to increase arterial oxygenation by restoring the functional residual capacity towards normal, recruiting previously unventilated alveoli and improving compliance. The ventilation frequency is given in breaths per minute (BPM) and ranged from 70-110 BPM. In all cases, a stable ventilation baseline was obtained at an oxygen concentration (F_iO_2) varying between 0.4-0.5. Also, the percentage inspiratory time (I:T) varied between 30-40% depending on the subject.

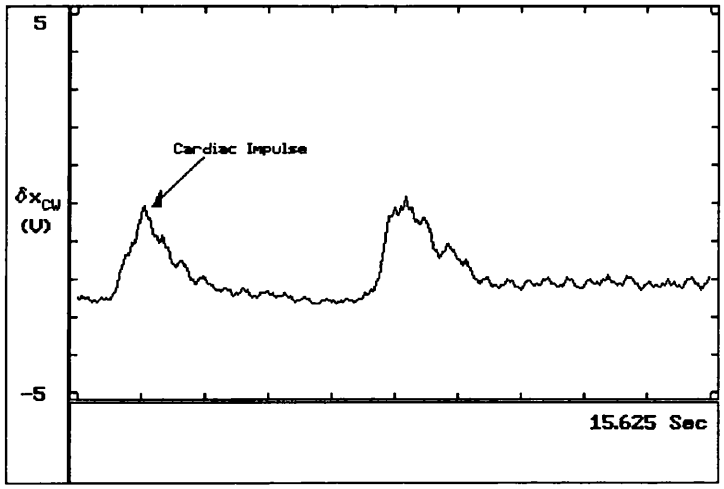
The following sections discuss the individual case studies. The transfer function calculations was taken as the ratio of chest-wall displacements (δx_{CW}) to patient airways pressure (P_{AW}) harmonics. Spectral averaging was performed to improve the SNR of measurements in all cases.



(a) Jet ventilator drive pressure.

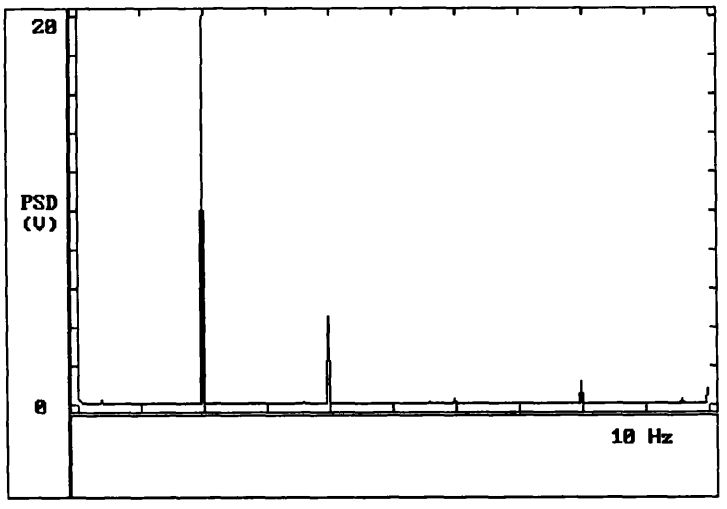


(b) Patient airways pressure.

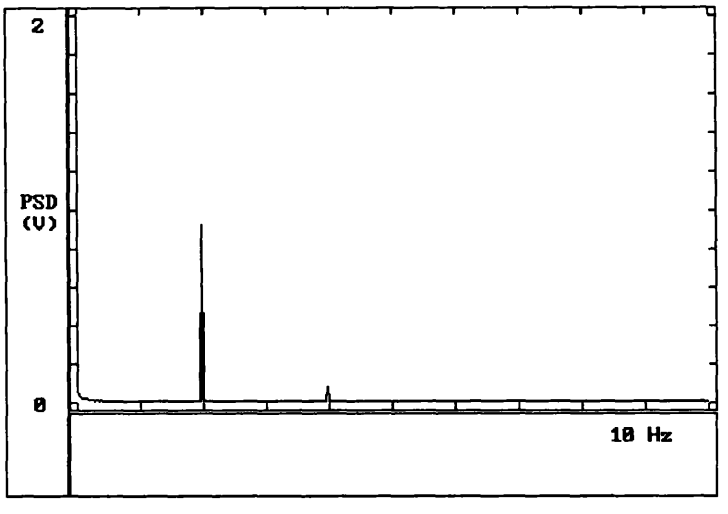


(c) Chest-wall displacement.

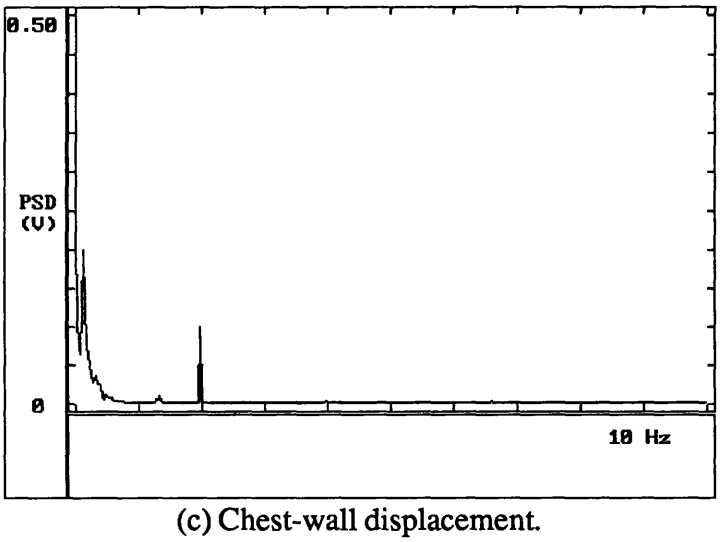
Fig. 6.1 Voltage-time measurements for P01.



(a) Jet drive pressure.



(b) Patient airways pressure.



(c) Chest-wall displacement.

Fig. 6.2 Power-spectrum calculations for P01.

6.3 Case Study P01

Past Medical History

Age: 63 *Sex:* Male *File:* F011_01

This patient is post-gastrectomy for carcinoma of the stomach and he also suffers from chronic obstructive airways disease (COAD). Pre-medical measurement of average blood gases (ABGs) with air was:

$PaCO_2 = 5.8$ kPa, $PaO_2 = 9.8$ kPa.

IPPV

- Conventional IPPV was applied @ 12 BPMx600 ml/breath, where the tidal volume $V_T = 600$ ml/breath;
- Fraction of inspired oxygen was set to $F_iO_2 = 0.4$ (or 40% oxygen concentration);
- Peak airways pressure = 33 cmH_2O, $PaCO_2 = 3.8$ kPa, $PaO_2 = 24.1$ kPa; these pressure indicate that the $PaCO_2$ is too low, whilst PaO_2 too high compared with average levels - hence, the patient was overventilated;
- Chest X-ray reveals pulmonary oedema and chronic bronchitis.

HFJV

Following IPPV the patient was ventilated with HFJV. The ventilator was set to mode a (Sec. 5.3.4) and the patient ventilated with white-noise pressure oscillations only. The resulting ABGs was measured at:

$PCO_2 = 9.89$ kPa, $PO_2 = 27.9$ kPa.

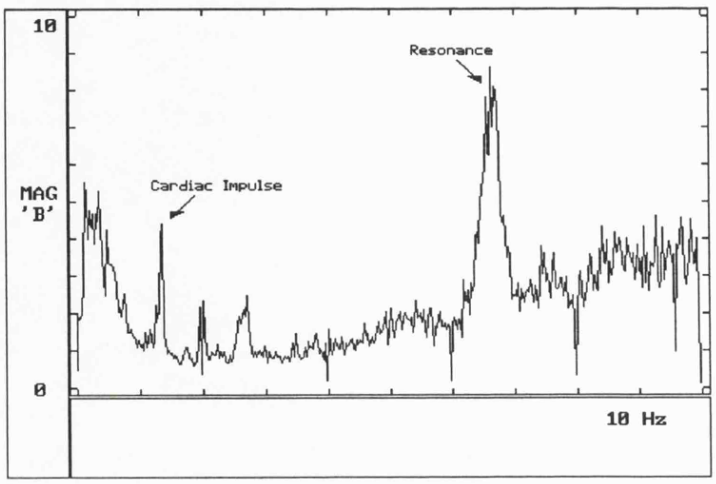
The ventilator was then set to its normal state (mode c) and the patient ventilated. During this time, the real-time analyser was used to capture and calculate the results shown in Figs. 6.1-6.3.

$P_{DP} = 20$ psi @ 120 BPM and I:T = 30% ($F_iO_2 = 0.5$)

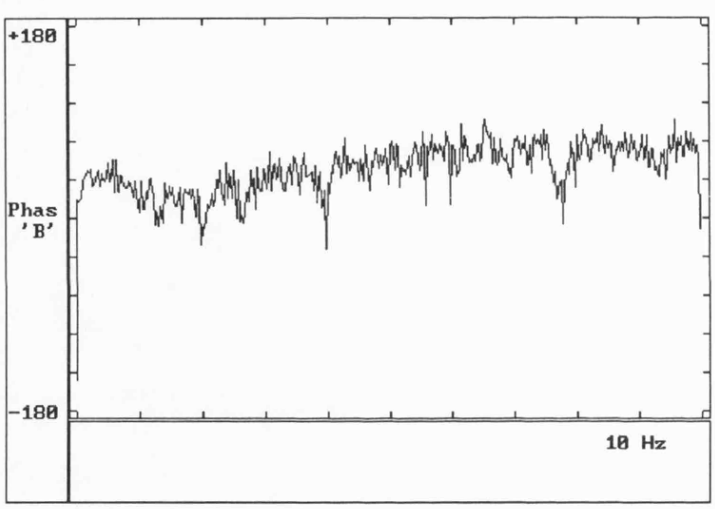
ABGs: $PCO_2 = 7.9$ kPa, $PO_2 = 12.5$ kPa.

Results

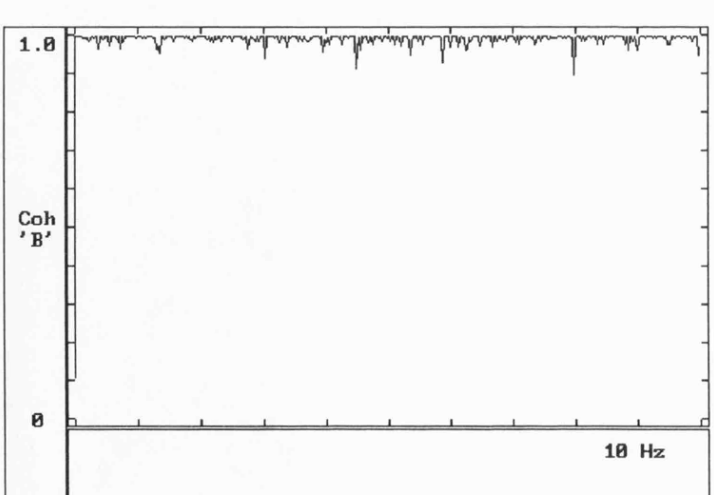
Figs. 6.1-6.2 shows the voltage-time plots and the corresponding power spectra of measured respiratory pressures and displacement. Since the patient was ventilated at approximately 120 BPM we can see a significant peak at about 2 Hz in all the spectra (Fig. 6.2). A particular feature worth noting from Figs. 6.1b-c is the presence of cardiac impulses which have been transmitted through to the pressure and displacement



(a) Modulus spectrum.



(b) Phase spectrum.



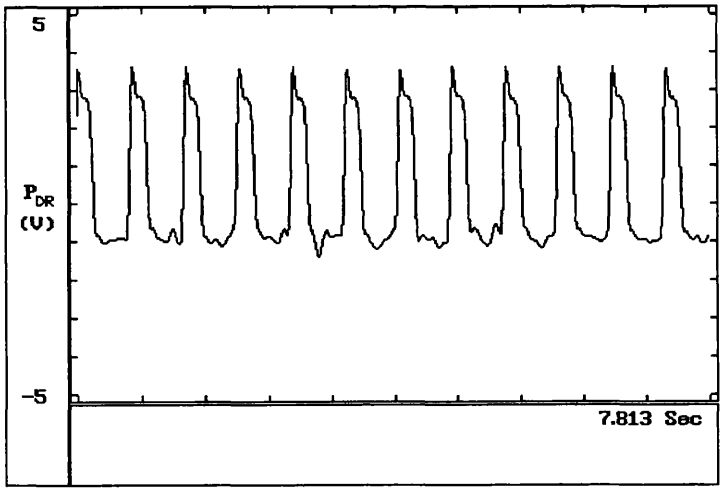
(c) Coherence spectrum.

Fig. 6.3 Transfer function relationships for P01.

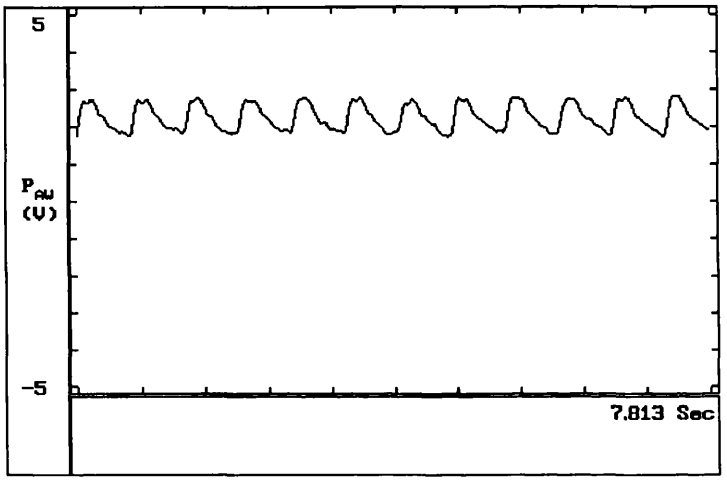
transducers. This phenomenon can be clearly seen from the magnitude spectrum of Fig. 6.3a which has a sharp peak at 1.36 Hz corresponding to the patient heart rate measured at 82 beats per minute. A second harmonic due to the cardiac impulse can also be seen at 2.7 Hz.

As expected, the modulus spectrum (Fig. 6.3a) shows a peak at about 2 Hz corresponding to the ventilation frequency. More significantly, we can see a prominent peak at 6.625 Hz which we have attributed to the acoustic resonance of the patient airways. A high coherency spectrum (>0.9) throughout the frequency range confirms the accuracy of all measurements (Fig. 6.3c). The phase spectrum (Fig. 6.3b) is seen to be positive throughout the frequency range, which implies that the chest-wall oscillations always lead the pressure input.

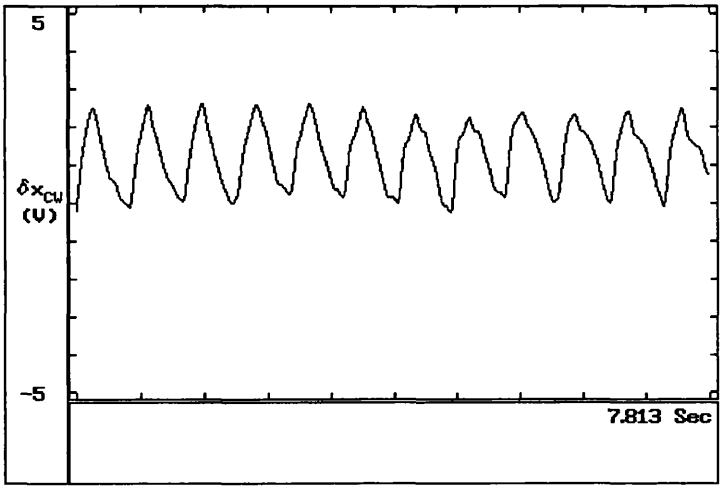
The phase spectrum has an underlying trend (Fig. 6.3b) where the phase increases linearly with frequency. However, the phase does not exhibit any significant change near the resonant frequency, as might be expected in a second-order linear system. This suggests that the underlying system dynamics exhibit complex non-linear behaviour which interferes with the phase relationship between the transfer function variables.



(a) Jet ventilator drive pressure.

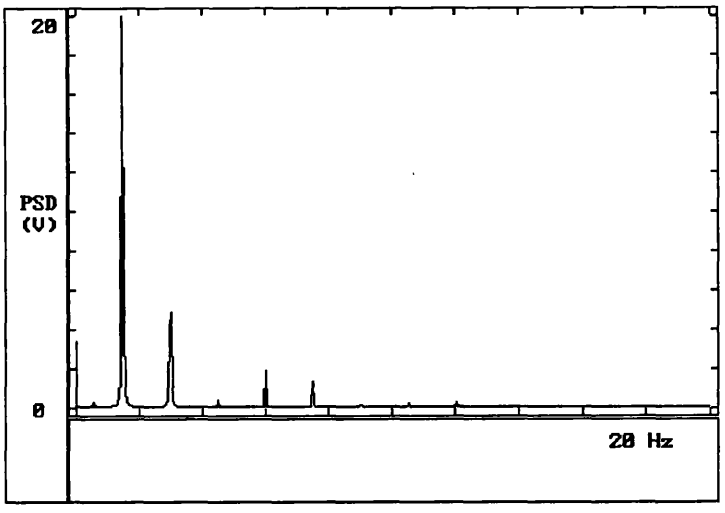


(b) Patient airways pressure.

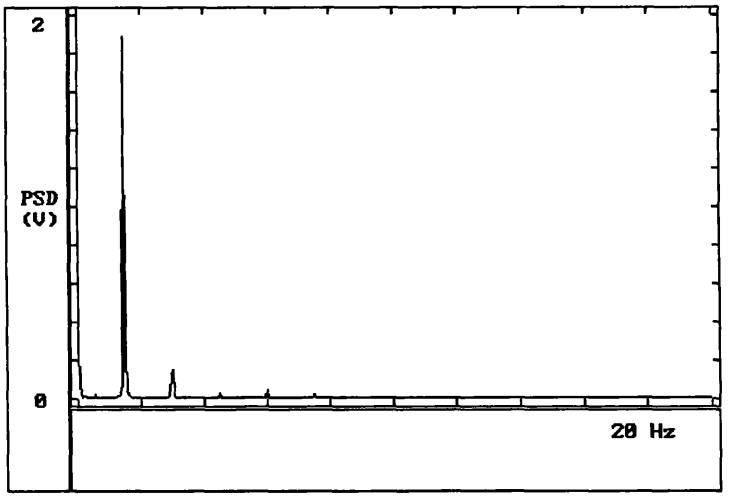


(c) Chest-wall displacement.

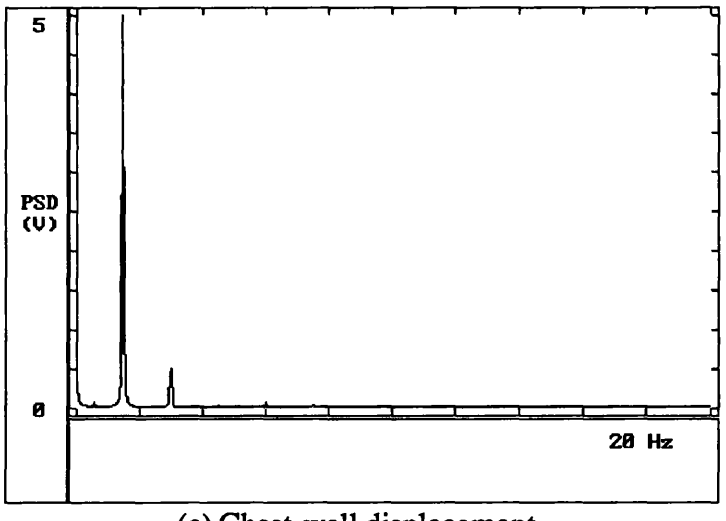
Fig. 6.4 Voltage-time measurements for P02.



(a) Jet drive pressure.

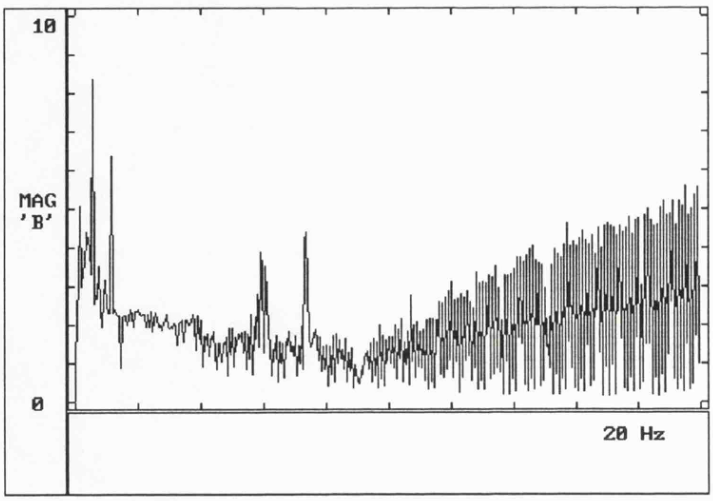


(b) Patient airways pressure.

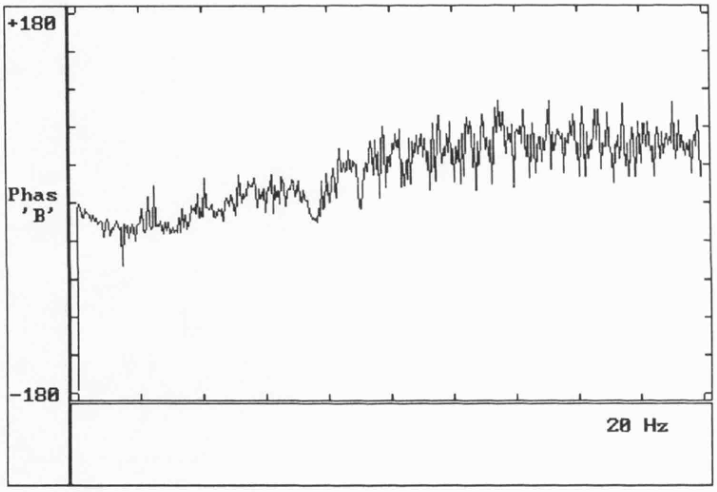


(c) Chest-wall displacement.

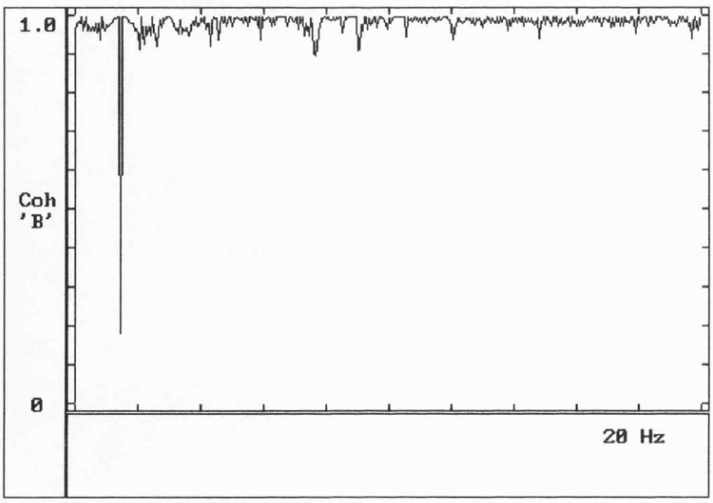
Fig. 6.5 Power-spectrum calculations for P02.



(a) Modulus spectrum.



(b) Phase spectrum.



(c) Coherence spectrum.

Fig. 6.6 Transfer function relationships for P02.

6.4 Case Study P02

Past Medical History

Age: 32 *Sex:* Female *File:* RITU1

This patient has no history of cardiovascular or respiratory problems - as confirmed by the chest X-rays. The patient had a drug overdose and required IPPV. Some muscle rigidity was diagnosed.

IPPV

- Conventional IPPV was applied @ 12 BPMx780 ml/breath (V_T=780 ml/breath);
- Fraction of inspired oxygen, F_iO_2 = 0.36 (or 36% oxygen concentration);
- Peak airway pressure = 20 cmH_2O, $PaCO_2$ = 3.25 kPa, PaO_2 = 17.8 kPa;
- Chest X-ray was clear.

HFJV

Following IPPV the patient was ventilated with HFJV in mode c. During this time, the real-time analyser was used to capture and calculate the results shown in Figs. 6.4-6.6.

P_{DP} = 25 psi @ 90 BPM and I:T = 30% (F_iO_2 = 0.4)

ABGs: PCO_2 = 3.8 kPa, PO_2 = 14.6 kPa.

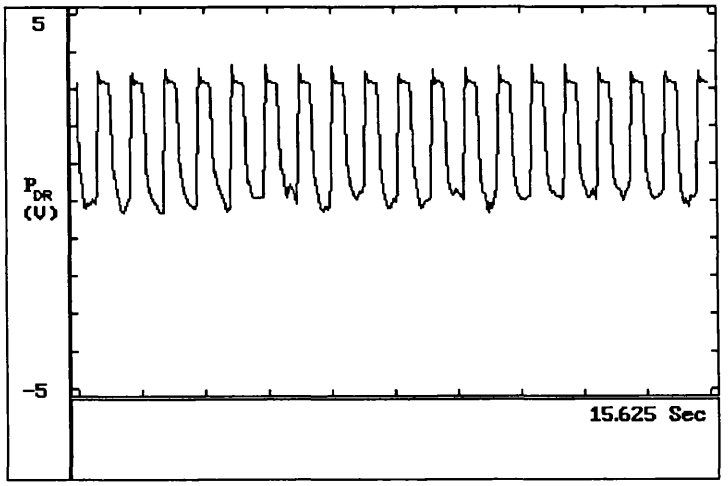
Results

Figs. 6.4-6.5 shows the voltage-time plots and the corresponding power spectra of measured respiratory pressures and displacement. Since the patient was ventilated at approximately 90 BPM we can see a significant peak at 1.5 Hz in all the spectra (Fig. 6.5). The bandwidth of measurements was selected over 0-20 Hz for this patient.

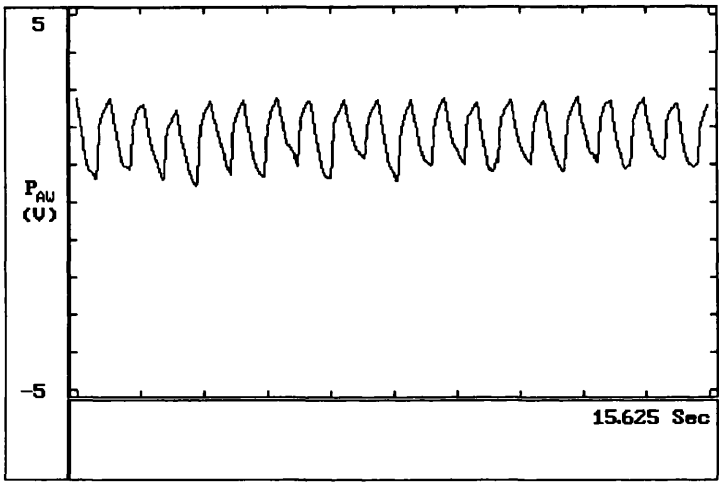
The modulus spectrum (Fig. 6.6a) shows a peak at about 1.5 Hz corresponding to the ventilation frequency and another peak at 0.6 Hz corresponding to the heart rate. More significantly, we can see two resonant peaks at 6.00 Hz and 7.40 Hz. The transfer function spectra are seen to be more noisier compared to the previous patient because of the smaller number of averages that were performed.

The phase spectrum (Fig. 6.3b) is seen to start off at negative and gradually increases to remain positive at higher frequencies. The phase spectrum is seen to cross the zero line somewhere between the two resonant peaks at 6.00 Hz and 7.40 Hz, there is also a significant dip in phase at 7.40 Hz.

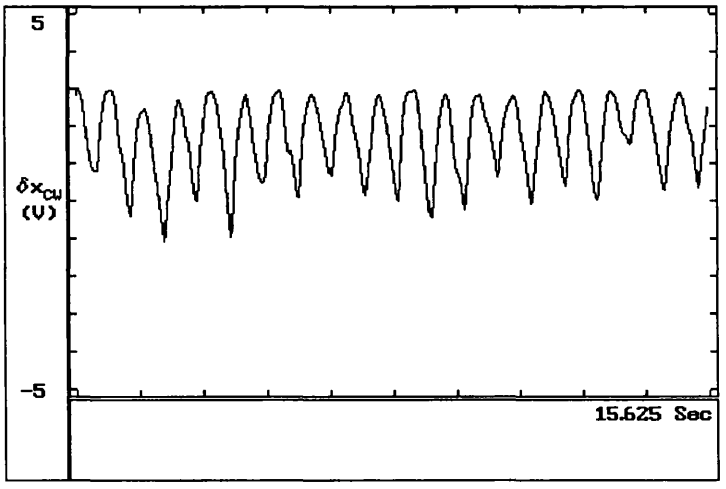
It is possible that the two peaks seen here are the result of a time-varying resonance, where a single resonant peak exists but has gradually shifted with time according to the physiological status of the patient.



(a) Jet ventilator drive pressure.

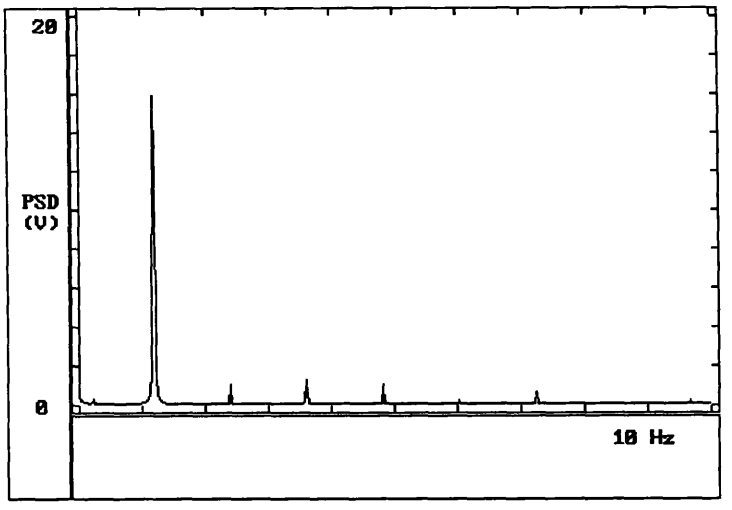


(b) Patient airways pressure.

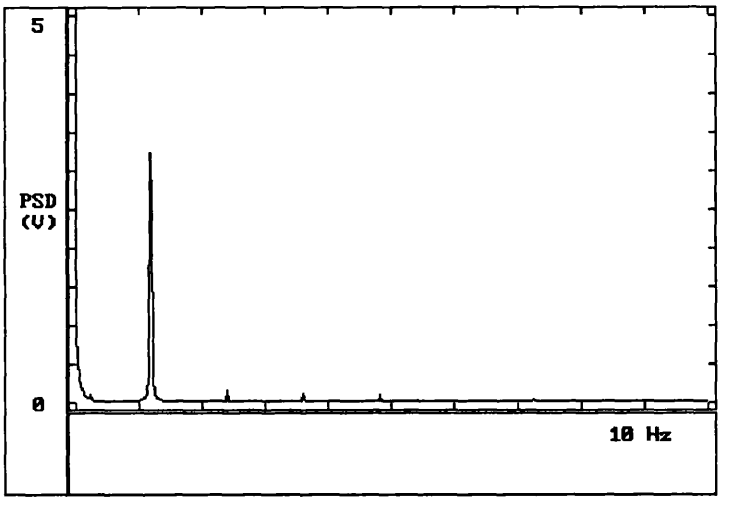


(c) Chest-wall displacement.

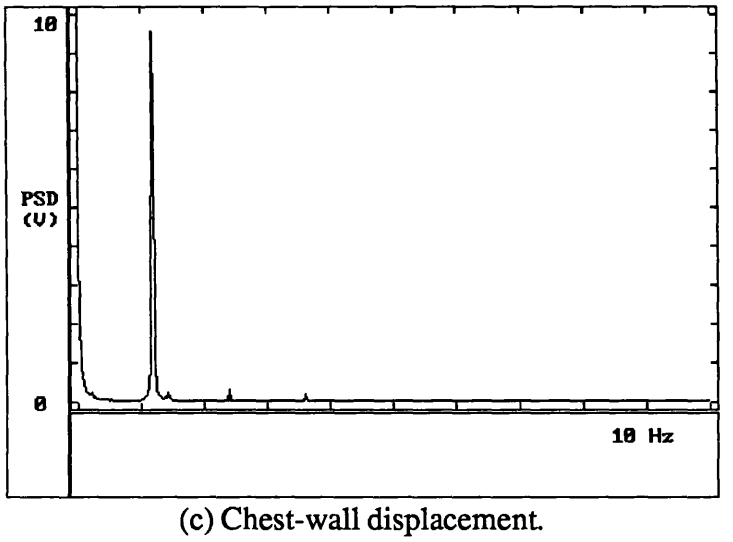
Fig. 6.7 Voltage-time measurements for P03.



(a) Jet drive pressure.

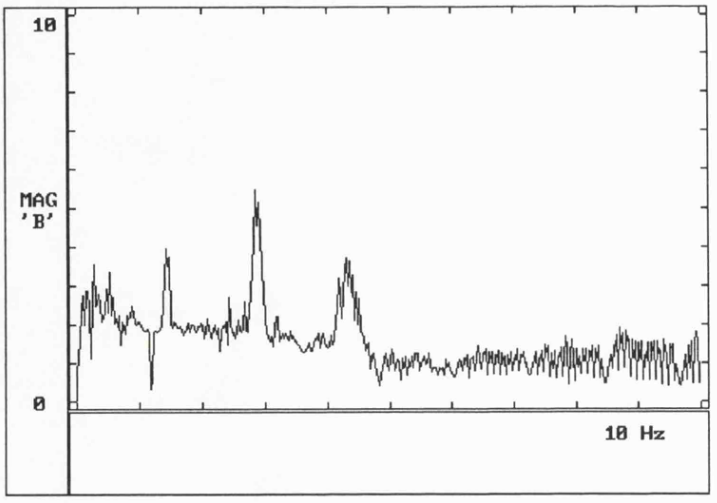


(b) Patient airways pressure.

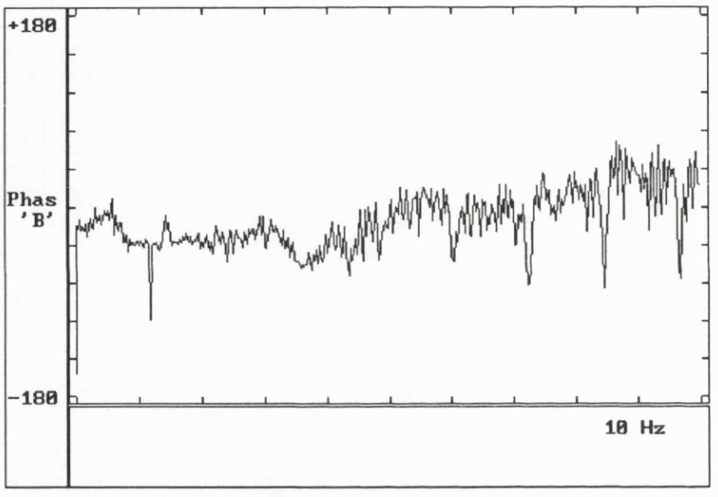


(c) Chest-wall displacement.

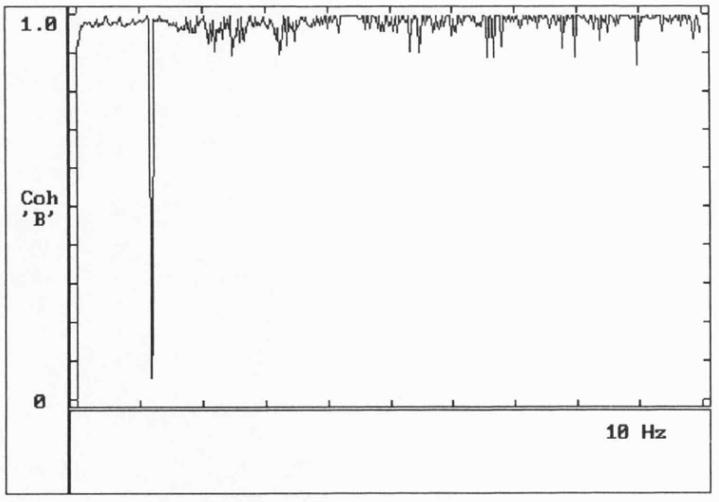
Fig. 6.8 Power-spectrum calculations for P03.



(a) Modulus spectrum.



(b) Phase spectrum.



(c) Coherence spectrum.

Fig. 6.9 Transfer function relationships for P03.

6.5 Case Study P03

Past Medical History

Age: 44 *Sex:* Male *File:* ITUPT2

Alcoholic and asthmatic was admitted after a heavy drinking binge which resulted in respiratory arrest. This condition is caused by muscle fatigue which eventually circumvents ventilation. An acute chronic chest infection was also diagnosed and some degree of cerebral anoxia (i.e. lack of oxygen in the brain).

No history of cardiovascular or respiratory problems - as confirmed by the chest X-rays.

IPPV

- Conventional IPPV was applied @ 12 BPMx750 ml/breath (V_T=750 ml/breath);
- Fraction of inspired oxygen, F_iO_2 = 0.36 (or 36% oxygen concentration);
- Peak airway pressure = 34 cmH_2O, $PaCO_2$ = 7-8 kPa, PaO_2 = 11.4 kPa;
- Chest X-ray was clear.

HFJV

Following IPPV the patient was ventilated with HFJV in mode c. During this time, the real-time analyser was used to capture and calculate the results shown in Figs. 6.7-6.9.

P_{DP} = 24 psi @ 70 BPM and I:T = 40% (F_iO_2 = 0.5)

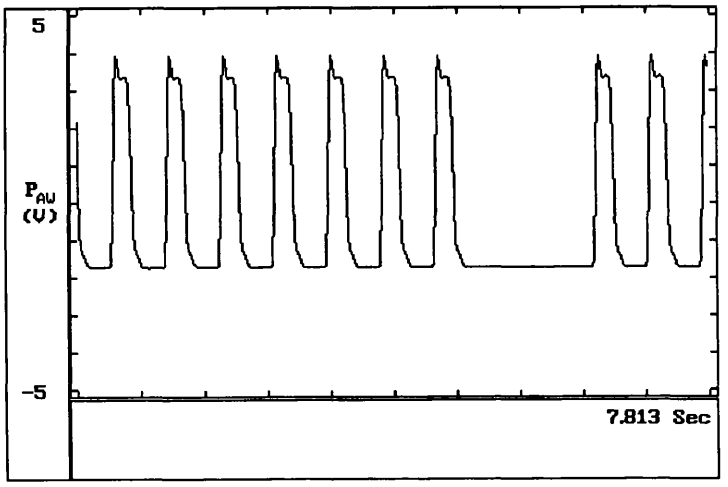
ABGs: PCO_2 = 5-6 kPa, PO_2 = 9-10 kPa.

Mean Airway Pressure = 14 cmH_2O + 5 cmH_2O PEEP.

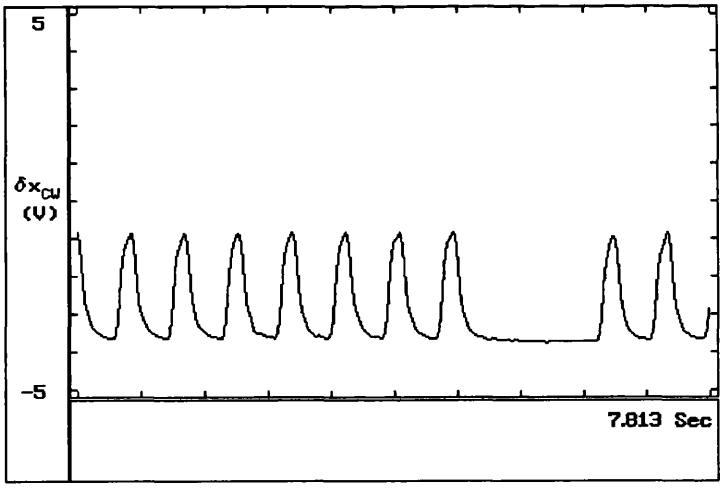
Results

Figs. 6.7-6.8 shows the voltage-time plots and the corresponding power spectra of measured respiratory pressures and displacement. Since the patient was ventilated at approximately 70 BPM we can see a significant peak at 1.2 Hz in all the spectra (Fig. 6.8). The bandwidth of measurements was selected over 0-10 Hz for this patient.

The modulus spectrum (Fig. 6.9a) shows two resonant peaks at 2.875 Hz and 4.325 Hz. A small number of spectral averages were performed on this patient and the spectra seen to be noisy at higher frequencies. The phase spectrum (Fig. 6.9b) is seen to remain negative for most of the frequency range and gradually crosses to remain positive for frequencies greater than about 8.0 Hz.

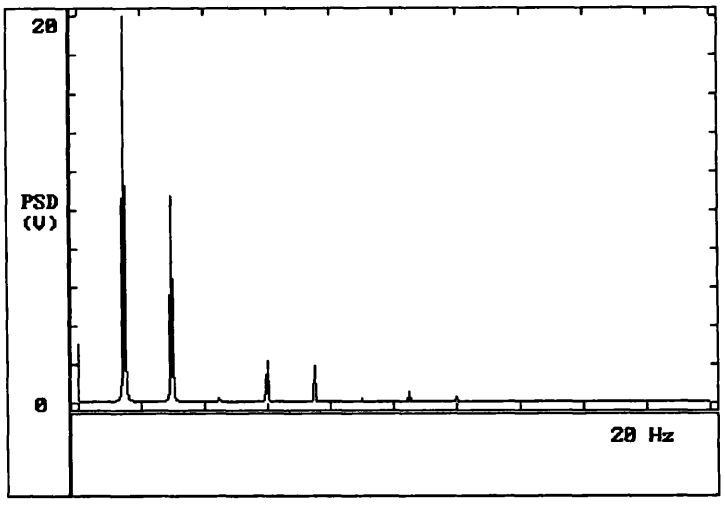


(b) Patient airways pressure.

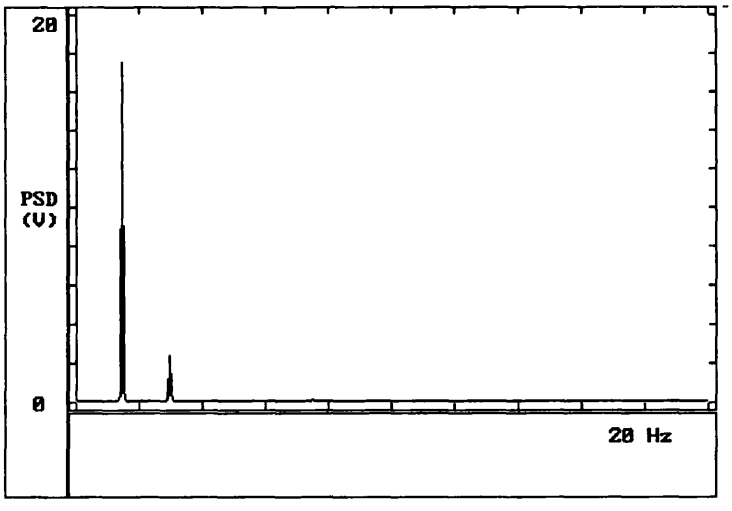


(c) Chest-wall displacement.

Fig. 6.10 Voltage-time measurements for P04.

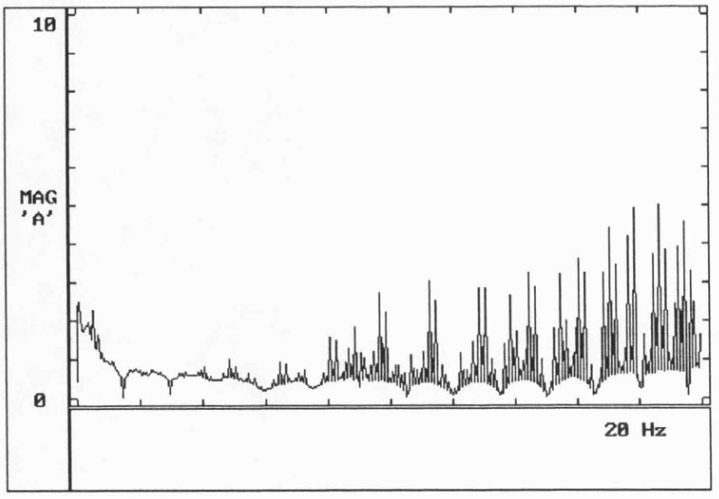


(b) Patient airways pressure.

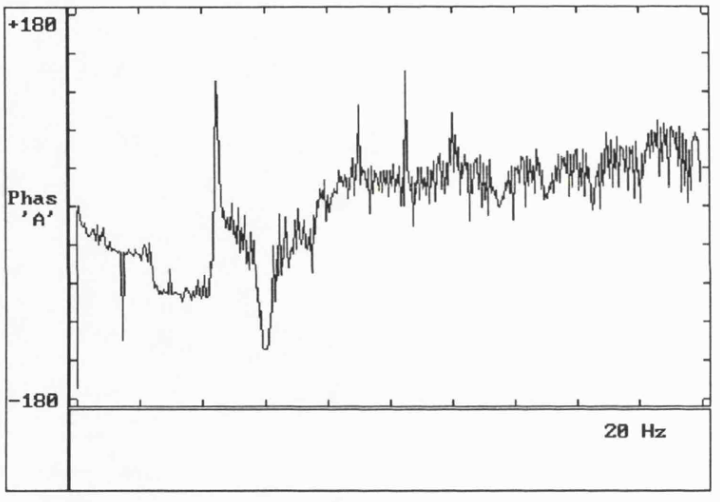


(c) Chest-wall displacement.

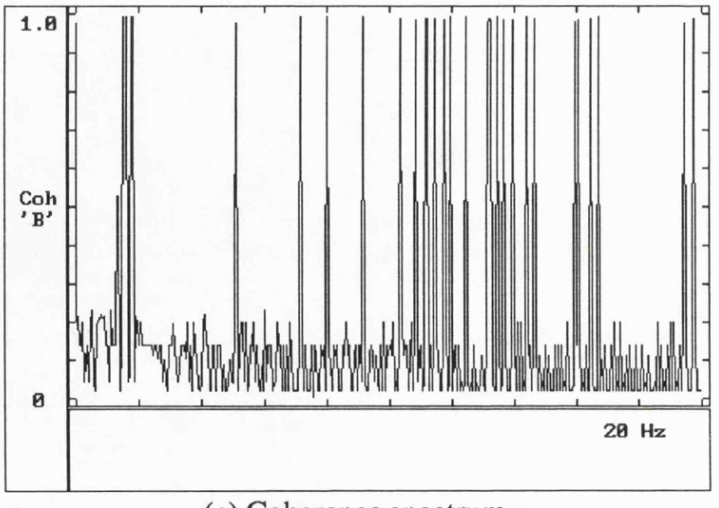
Fig. 6.11 Power-spectrum calculations for P04.



(a) Modulus spectrum.



(b) Phase spectrum.



(c) Coherence spectrum.

Fig. 6.12 Transfer function relationships for P04.

6.6 Case Study P04

Past Medical History

Age: 84 *Sex:* Male *File:* PT03

This patient had a flail chest after falling. Some ribs had been broken and this was affect chest wall movement and hence the efficiency of spontaneous ventilation. A degree of left-ventricular failure led to the loss of pulmonary efficiency and eventually to respiratory failure. HFJV was used applied the outset but was unsuccessful. It was decided that HFJV should be administered through a mini-tracheostomy for extubating the trachea before *weaning* (e.g. withdrawing the patient from artificial ventilation).

IPPV

- Conventional IPPV was applied @ 12 BPMx550 ml/breath (V_T=550 ml/breath);
- Fraction of inspired oxygen, F_iO_2 = 0.4 (or 40% oxygen concentration);
- Peak Airway Pressure = 35 cmH_2O + 7 cmH_2O PEEP;
- PCO_2 = 6.5 kPa, PO_2 = 12.6 kPa.

HFJV

Following IPPV the patient was weaned with HFJV in mode c. During this time, the real-time analyser was used to capture and calculate the results shown in Figs. 6.10-6.12.

P_{DP} = 25 psi @ 100 BPM and I:T = 30% (F_iO_2 = 0.5)

ABGs: PCO_2 = 8-14 kPa, PO_2 = 6-7 kPa.

Results

Figs. 6.10-6.11 shows the voltage-time plots and the corresponding power spectra of measured respiratory pressures and displacement. Since the patient was ventilated at approximately 90 BPM we can see a significant peak at about 1.50 Hz in all the spectra (Fig. 6.11).

The transfer function spectra for this patient (Fig. 6.12) shows no significant peaks at any frequency. The results display poor coherence (Fig. 6.12c) and we cannot attach any significance to these measurements. This is due to the poor condition of the patient and also the mini-tracheostomy tube used for weaning has reduced the overall volume of gas flowing into the patient.

6.7 Discussion

This chapter has presented the results of clinical studies on critically-ill patients undergoing HFJV as part of their therapeutic regimen. The instrumentation developed in Chapter 5 was used to ventilate the patients so that system identification could be performed to determine the dynamic response of their respiratory airways. The main objectives of the clinical studies was to:

- investigate the performance of the computer-based instrumentation, modified high-frequency jet ventilator and fibre-optic displacement transducer in the clinical environment (Chapter 5);
- test the hypothesis that the respiratory airways behaved as an acoustic resonant circuit (Sec. 4.4);
- identify any characteristic features which may need to be considered for implementing automatically controlled patient ventilation.

A study population of four critically-ill patients were jet ventilated and the transfer function relationship between chest-wall displacements (δx_{CW}) to patient airways pressure (P_{AW}) calculated. Spectral averaging was performed to improve the SNR of measurements and the coherency spectrum calculated to provide a measure of confidence in the spectral estimates. The experiments were carried out during routine application of HFJV with minor changes to standard clinical protocol. The results for each patient are presented as case studies P01, P02, P03 and P04.

In all cases, except case study P04 (Sec. 6.6), a prominent resonant peak could be clearly identified in the transfer function magnitude spectra. In cases P02 and P03 there were two resonant peaks (P02: 6.00 Hz and 7.40 Hz, P03: 2.875 Hz and 4.325 Hz) with an inter-peak separation of approximately 1.40 Hz. These resonances may be interpreted as being:

- an accurate representation of the acoustic properties of these patients; or
- artefact created by a single time-varying resonant peak; due to the averaging process, the single resonant peak appears as two separate peaks.

These points can only be clarified following a more substantial clinical study involving a larger population group. If we assume that the latter is true, then we must take this into consideration when designing the next generation of ventilator devices - which should facilitate automatic and optimal ventilation of patients.

This study has clearly validated the hypothesis that the respiratory airways exhibit characteristics similar to an acoustic resonant circuit. These results are reinforced by a strong coherency spectrum throughout the frequency span of measurements.

The performance of the instrumentation was found to significantly improve the speed and accuracy with which data could be captured and analysed compared to previous techniques. The advantages of the system included:

- the computer-system provided real-time signal processing and graphic display of respiratory measurements within the clinical environment. The measurements were successfully performed by the anaesthetist who had been given only nominal training with the instrument (Chapter 5);
- the modified jet ventilator was operated in various modes (Sec. 5.3.4) to obtain stimulus-response data for performing the identification studies presented here;
- the fibre-optic displacement transducer overcame most of the problems associated with previous methods of chest-wall displacement measurement (Sec. 5.7) - providing linear, wide-bandwidth measurements of chest-wall displacement with minimal baseline drift.

Based on the results from the case-studies presented in Secs. 6.3-6.6, the following chapter describes the design and simulation of a second order model-reference adaptive control system for automatic patient ventilation. This is intended to solve the problem associated with a time-varying resonance (Sec. 6.4-6.5). For example, assuming that optimal alveolar ventilation is achieved by ventilating a patient at a frequency corresponding to the resonant frequency of his airways, then a self-tuning ventilator could be specified whose objective would be to ensure that the ventilation frequency tracks the patients' resonant frequency with time. Chapter 7 presents a control scheme which can be used to implement optimal ventilation under conditions of time-varying patient parameters, and hence a time-varying resonance.

This study has made no attempt to correlate the resonance of the respiratory airways with alveolar ventilation. This was mainly due to the limitations imposed by the transducers that were available for continuous monitoring of alveolar ventilation. The transducers were in the form of invasive electrode devices (Clarke or Severinghaus) which were used to measure the partial-pressure of blood-gases. Due to their limited bandwidth they were not suitable for continuous measurement of alveolar ventilation over the range of frequencies covered by HFJV.

Chapter 7
Design and Simulation of a Second Order MRAC System for Artificial Ventilation

7.1 Introduction

The prime task of artificial ventilation is to oxygenate patients incapable of performing spontaneous breathing, by transfer of oxygen from the inspired air to the blood stream via the alveoli, and at the same time remove carbon dioxide from venous blood and exhaust it out into the environment. This method of ventilation is routinely used in respiratory and intensive therapy units on patients with healthy lungs as well as in catastrophic lung disease. However, since artificial ventilation deviates considerably from the normal physiological mechanism of respiration, certain adverse effects can result when the method is used improperly. General acceptance of controlled ventilation is limited by:

- the harmful effects caused by cardiovascular depression due to interference with the venous return to the heart and of disturbances in the distribution of blood and gases through the lungs during ventilation;
- an understanding of the bio-physiological basis of the effectiveness of controlled ventilation and the mechanisms underlying the harmful effects which may arise;
- the diversity of ventilator machines and the optimal criteria used for setting ventilator parameters and control of their performance.

Only with a greater understanding of these points can optimal use be made of the precise control parameters provided by controlled ventilation.

The instrumentation described in Chapter 5 presented the design and implementation of a real-time system for characterising the respiratory dynamics of patients receiving high frequency jet ventilation. The next phase in the instrument development is to introduce

closed-loop controlled automatic ventilation. The most important design criterion is that the control system remain stable at all times. Also, the control algorithm must be robust enough to cope with the fluctuations in physiological parameters of the patient with time. Thus, identification of the patient dynamics, by periodically updating the patient-model parameters on the basis of monitoring and analysis of state informations, becomes a prerequisite to the solution of the optimisation problem.

This chapter presents a simulation study of a model reference adaptive control scheme for automatic management of ventilator parameters to optimise gas exchange in patients receiving anaesthesia. A second order lumped-parameter patient-model is used which takes into account the parameters relevant to ventilation and at the same time can be updated from analysis based on respiratory measurements. The self-tuning ventilation scheme can automatically adjust its parameters in response to changes in the patient's respiratory state (i.e., lung and chest wall compliance, airways impedance and partial pressure of alveolar gases). The control algorithm is intended for implementation on the system described in Chapter 5.

7.2 Considerations of Respiratory Physiology and Control

Artificial ventilation is contrary to the normal physiological mode of respiration in which air is transported to the lungs. This section outlines the physiological factors affecting gas exchange during artificial ventilation and specifies criteria for selecting control variables for implementing automatically controlled ventilation.

7.2.1 Spontaneous Ventilation

During spontaneous breathing, the inspiratory muscles enlarge the size of the thoracic cavity, the volume of the gas within it increases, and the pressure within the thorax falls. The pressure gradient between the intra-pleural and alveolar pressures overcomes the inertial and elastic properties of the lungs and chest wall. A passive expiratory cycle takes place when the elastic energy stored in the chest wall during inspiration is used to expel inspired gases. Active expiration is possible when certain muscles become active during expiration or by applying a negative pressure phase during controlled ventilation.

7.2.2 Adverse Effects of Controlled Ventilation

Any side-effect of controlled ventilation is the result of abnormally increased intra-thoracic pressures from those of spontaneous breathing levels. This may have some adverse affect on the circulatory system of some patients. The earlier studies on the harmful effects of controlled ventilation were reported by Werko (1947), Cournand et al. (1948), Braunwald et al. (1957), and Morgan et al. (1966) respectively.

Morgan et al. (1966) have shown that an increased intra-thoracic pressure results in an immediate decrease in pulmonary arterial blood flow and a corresponding drop in aortic blood flow. Their measurements indicate that the average intra-thoracic pressure over a complete respiratory cycle may lower cardiac output and reduce the systemic circulating blood volume. However, applying higher pressures for shorter periods of time has a less significant affect on cardiac output. Cournand et al. (1948) showed that the depression of cardiac output is reduced for lower intra-thoracic pressures.

Interference of normal pulmonary blood flow and volume can cause blood to back up in the right side of the heart (Wald et al., 1968). The myocardium must then do extra work to expel this excess blood against the added resistance of capillaries restricted by the increased alveolar pressure. This may lead to pulmonary oedema where fluid leaks from the capillaries into the alveoli. In severe cases, the extra burden on the myocardium may precipitate right heart failure.

Another effect of controlled ventilation is a disturbance of the normal balance between ventilation of the lungs and the perfusion of the lungs with blood (Mushin et al., 1980). This can lead to:

- the perfusion of insufficiently ventilated parts of the lungs with venous blood, and, therefore, insufficient oxygenation of this blood;
- the ventilation of insufficiently perfused parts of the lung, which results in an increase in the physiological dead space since such ventilation is wasted.

The disturbance of ventilation/perfusion ratio may be attributed to localised changes in the mechanical properties of smaller airways as found in cases of asthma, chronic bronchitis and emphysema. The acid-base balance of blood is also affected by changes in alveolar ventilation. For example, acidaemia due to underventilation increases the sensitivity of the heart and other organs to hypoxia. Conversely, alkalaemia due to overventilation causes cerebral vasoconstriction and produces changes in tissue reactivity; however, good oxygenation is maintained and carbon dioxide eliminated.

7.2.3 Specification of Ventilator Parameters

In specifying the optimal ventilator, we must consider all parameters of controlled ventilation likely to have a significant affect on the overall safety and efficiency of the method. This section presents ventilator specification criteria based on considerations of the input pressure waveform and functions representing deleterious effects.

A. Input Pressure Waveform

A theoretical study by Wald et al. (1968) of several waveforms and their physiological significance found the optimal input pressure waveform to be a negative ramp input. There are a large number of pressure profiles which may be used. However, there is no very convincing evidence of the superiority of one over the other except that distribution of inspired gas is improved if there is a prolongation of the period during which applied pressure is maximal (Nunn, 1987).

In this study, ventilation is controlled by applying a high frequency jet of gas into the patients airway, which causes inspiratory flow. The rectangular wave has a rapid rise and fall of pressure, with constant value during application. The alveolar pressure constantly increases during inspiration and the peak value of alveolar pressure occurs at the end of the applied pressure pulse. This corresponds to normal physiological activity, where the peak alveolar pressure gradient occurs at the end of inspiration, while maximum intra-pleural pressure occurs a short time before the end of inhalation (Johnson, 1963). The time of occurrence in the inspiratory cycle of the peak pressure was found to be useful (Hildebrant and Young, 1965) in opening smaller airways and alveoli held closed by surface adhesive forces.

The Penlon high frequency jet ventilator used in this study has been modified (Sec. 4.7) to introduce small flow, low pressure white noise perturbations onto the normal binary output. This allows for the systematic characterisation of the subjects respiratory dynamics over all possible inputs without interfering with the normal mode of controlled ventilation (Kabay et al., 1989).

B. Dead Space and Ventilation

An appreciable part of the total inspired volume or tidal volume does not participate in blood-gas exchange. This fraction of the tidal volume is known as the dead space and comprises of an *anatomical* and *physiological* dead space.

The anatomical dead space is that portion of the tidal volume which remains in the upper airways, not entering the alveoli where blood-gas exchange can take place. This deadspace volume is influenced by many factors, some of which are of considerable importance. Radford (1955) showed that there was an approximate correlation between the weight of a subject and the dimensions of the conducting airways. Anatomical dead space tends to increase with age (Fowler, 1950), but this may be due to the increased incidence of chronic bronchitis. Hypoxaemia due to hypoventilation is the result of an imbalance between the rate at which oxygen is removed from the lung by the blood and

the rate at which it is replenished by alveolar ventilation (West, 1977). This state results in a marked reduction of the anatomical dead space, which in turn limits the fall of alveolar ventilation resulting from small tidal volumes.

The physiological dead space is that part of the tidal volume which does not participate in gas exchange and is dependent on the relationship between gas in the alveoli, blood supply to the alveoli, and the transport mechanism of blood-gas exchange. Many factors influence the physiological dead space, including the subjects' age, body size, posture and smoking habits. A useful metric is the ratio of physiological dead space (V_D) to tidal volume (V_T), since it tends to remain fairly constant while the actual value for V_D may vary widely with tidal volume (Enghoff, 1931). The relationship between tidal volume and dead space is crucial to the success of high frequency ventilation, where the tidal volume is less than the anatomical dead space. In practice, an increase in tidal volume without a corresponding increase in pulmonary blood supply will result in an effective increase in physiological dead space. Other effects such as atelectasis and shunting may also cause an effective increase in physiological dead space (Radford, et al., 1954).

C. Work of Breathing

The mechanisms of spontaneous breathing are purely passive and may be produced either by use of the respiratory muscles or by the development of a pressure gradient between the airways and the space surrounding the chest. Inspiration is actively controlled by contraction of the inspiratory muscles. Expiration is normally passive and its energy source is the elastic energy stored in the chest wall and lungs during inspiration. Hence, the work of spontaneous breathing is performed entirely by the inspiratory muscles. During controlled ventilation, the ventilator must perform the work of inspiration and move gas into the patient.

The main sources of impedance which must be overcome are the elastic recoil of the lungs and chest wall, frictional impedance to gas flow of the airways, frictional impedance of tissue deformation, and the inertance of tissue and gases which must be overcome during flow reversal associated with transition between the inspiratory and expiratory phases.

A relatively small amount of power is consumed by the respiratory muscles of a healthy resting subject. Almost 90% of this power is dissipated as heat in the muscles and only 10% is available for moving gas against frictional impedance of the airway and the tissues. The efficiency is reduced even further in cases of respiratory disease. Otis (1954) showed that during spontaneous breathing in healthy patients, increased ventilation beyond a maximum will be entirely consumed by the respiratory muscles.

In designing a ventilator, we must take into account the extra work required in transporting gas through external apparatus and into the patient. Wald et al. (1968) have developed mathematical formulae for quantifying the amount of work that the ventilator must perform.

7.2.4 Automatically Controlled Ventilation

In using controlled ventilation, special care must be taken to ensure that the patient's ventilatory requirement is adequately met whilst satisfying the constraints imposed by harmful effects. The ventilator parameters should be so adjusted that both of these conditions are satisfied. Since ventilation may be required over prolonged periods of time, manual adjustments are not satisfactory and an automatic control scheme is required. However, the complexity of the physiological system is such that it often presents problems for obtaining control information. This section gives a brief overview of the application of closed-loop control to artificial ventilation and looks into the considerations necessary for specifying physiological control variables.

Jain and Guha (1972) have implemented an adaptive controller based on optimisation of a performance index. The control system was developed on an analogue computer using a first order linear RC lung model. The system was tested for changes in process parameters for a constant controller gain (K) setting. In this case, the error rapidly damped out to zero. However, the main drawback of this system is that the gain K must be manually adjusted. Hence for large swings in respiratory parameters, the stability of the control system can be compromised. This problem can be overcome by using the model reference scheme described in Sec. 7.3, which is more robust and assures stability for large changes in system parameters.

The physiological mechanisms of respiratory control are extremely complex and involves the interaction of many different mechanisms. The level of control affecting breathing from any given mechanism will vary according to the circumstances; for example, the mechanism controlling minute volume during exercise is not the same as the mechanism active during the resting state. The respiratory control system regulates ventilation based on neurogenic and chemical factors. Chemoreceptors monitor the chemical composition of the arterial blood and responds to fluctuations to a drop in P_{O_2}, a rise in P_{CO_2} or H^+ concentration, or a fall in their perfusion rate. The central respiratory control areas then stimulate the muscles of respiration (e.g. the diaphragm, intercostal muscles, abdominal muscles and accessory muscles) with stimuli based on the information received from the chemoreceptors.

Artificial ventilation need not be controlled to the same degree of complexity and precision as the physiological system. An effective and efficient control scheme can be implemented provided that we identify the most dominant physiological factor which can be used as the control variable. The following sections will outline some considerations for implementing an artificial ventilator and criteria for selecting control variables.

A. Optimal Ventilator Characteristics

Ideally we require a ventilator which takes into account the factors mentioned in Sec.(7.2.3). In some cases, during prolonged artificial ventilation, the patient may experience metabolic changes. The ventilator control system must respond quickly to such variations in patient parameters and regulate ventilation; in this instance, according to the metabolic rate of the patient.

The main characteristics of an *optimally* controlled ventilator would include the following:

- efficient removal of carbon-dioxide from de-oxygenated blood;
- efficient supply of oxygen to arterial blood for metabolism;
- control ventilation to maintain normal arterial PCO_2 and PO_2 levels;
- average alveolar pressure be minimised to limit any harmful side-effects;
- peak alveolar pressure kept below a maximum safety threshold;
- the work of ventilation be kept to a minimum.

Other important considerations include the stability and robustness of the system. Since the purpose of automatic controlled ventilation is to allow unattended ventilation of patients, it is extremely important that the ventilator control system remain stable at all times. Also, the control algorithm must be robust enough to tolerate *noisy* data.

B. Alveolar Oxygen Pressure

Many workers (Wald et al., 1968; Jain and Guha, 1972) have used alveolar oxygen pressure to control automatic ventilation since it is widely accepted to be indicative of ventilation and respiratory dynamics. The main advantage with this measurement is that the average alveolar pressure can be used as an index of circulatory interference due to ventilation. Increasing the alveolar pressure increases harmful side-effects. Decreased alveolar pressure results in a reduced gas volume, and hence blood-gas exchange will be compromised.

The practical implementation of a control scheme based on continuous monitoring of alveolar pressure is an attractive concept, but is difficult to achieve. The main restriction is the availability of pressure transducers which can be placed in the alveolar lung regions. However, with the development of fibre-optic pressure transducers, continuous

measurement of alveolar pressure may soon become attainable. An alternative approach to overcoming this measurement problem is to estimate continuous alveolar pressure levels indirectly through an on-line respiratory parameters identification scheme. For example, an adaptive identification scheme can be developed which provides an instantaneous estimate of alveolar pressure based on continuous measurements of tidal volume or some other observable parameter. Another approach may be to implement a non-invasive adaptive identification scheme using abdominal and chest-wall displacements to estimate alveolar ventilation.

C. Carbon Dioxide Pressure

Carbon dioxide is the end-product of metabolism which is measured as a partial pressure gradient PCO_2 from the mitochondria through the cytoplasm, the venous blood, the alveolar gas and by way of expired air to the atmosphere. Two important respiratory measurements of carbon dioxide tension are:

Alveolar PCO_2

The alveolar PCO_2 rises steadily during expiration, as carbon dioxide passes into the alveolar gas from the pulmonary capillaries. This expiratory pressure component may be determined from the carbon dioxide concentration at the mouth.

Arterial PCO_2

The PCO_2 of arterial blood leaving the pulmonary capillary follows similar fluctuations to alveolar PCO_2. However, regional variations exist depending on the ventilation-perfusion ratio of the different parts of the lung. Mixed arterial PCO_2 is monitored by taking samples over several seconds. Continuous monitoring is made difficult by the averaging associated with the mixing of blood from different lung regions.

Mitamura et al. (1971) have developed a respirator which controls ventilation by monitoring the patient's metabolic rate from measurements of end-expiratory carbon dioxide tension. This control scheme is useful during long-term ventilation of patients, where the CO_2 output may change. The performance of the system was found to vary depending on the mismatch in the ventilation-perfusion ratio. For example, under constant ventilation, arterial PCO_2 was found to increase for an induced change in metabolic rate. However, under similar conditions, the control system maintained the PCO_2 at a constant level even though CO_2 output increased.

When the ventilation-perfusion ratio is matched, alveolar levels of CO_2 are nearly equal to arterial levels at the end of expiration. Hence, controlled ventilation based on the CO_2 fraction of end-tidal expired gas will keep arterial levels of CO_2 constant. However, where there is a mismatch in the ventilation-perfusion ratio, ventilatory regulation by this method cannot keep PCO_2 constant because of the alveolar-arterial PCO_2 difference.

Many factors influence the end-expiratory PCO_2. The most significant effects are that:

- PCO_2 decreases with respiratory rate;
- at low tidal volumes, the end-tidal PCO_2 and mean alveolar PCO_2 may differ due to contamination of the end-tidal sample by dead-space air;
- the end-expiratory gas is not always representative of mean alveolar gas in cases where all the alveoli do not deflate together during expiration (i.e. in patients with chronic bronchitis, emphysema, and pulmonary fibrosis).

Radford (1955) investigated the relationship between carbon dioxide production and dead space. The results led to a protocol for ventilating normal patients. These standards cannot be applied to patients with respiratory insufficiency associated with serious problems. This discrepancy may be caused by an increase in the physiological deadspace or by an increase CO_2 production.

D. Respiratory Frequency

The commonest controls which are provided on high frequency ventilators include respiratory frequency, inspiratory:expiratory ratio, and the durations of inspiration and expiration. Mikami et al. (1966) have investigated the relationship between the respiratory frequency and alveolar ventilation under conditions of normal dead space, airway resistance and compliance. However, this relationship varied according to the parameter values. Mitamura et al. (1971) used this approach to develop their optimally controlled respirator.

Smith and Lin (1989) have proposed that the respiratory frequency be selected to coincide with the resonant frequency of the respiratory airways. The proposed hypothesis states that gas mixing will be enhanced and gas exchange optimised for ventilation at the resonant frequency. Preliminary results (Kabay et al., 1989) have shown that resonance of airways exists. The instrumentation and techniques developed to perform these experiments was discussed in detail in Chapter 5.

Investigations are currently in progress to determine the acoustic resonance of a population of patients (Chapter 6). The final aim is to perform a correlation analysis between respiratory frequency and its affect on alveolar ventilation. The remainder of this chapter is devoted to the design and simulation of a control system for implementing a self-tuning ventilator based on the mechanical and acoustic properties of the patient.

7.3 A Model-Reference Adaptive Control System

One approach to automatic control of the jet ventilator is to use a model-reference adaptive control (MRAC) scheme. The basic principle of an MRAC system (Fig. 7.1) is that it expresses a formulation of both a set of plant equations for the *patient*-model (true physiological system) and a set of reference equations which represent the desired performance characteristics of the patient in terms of a *reference*-model for a given command stimulus. The state variables of the MRAC system must be selected to be indicative of respiratory dynamics since they form the basis of the control law that will eventually drive the ventilator.

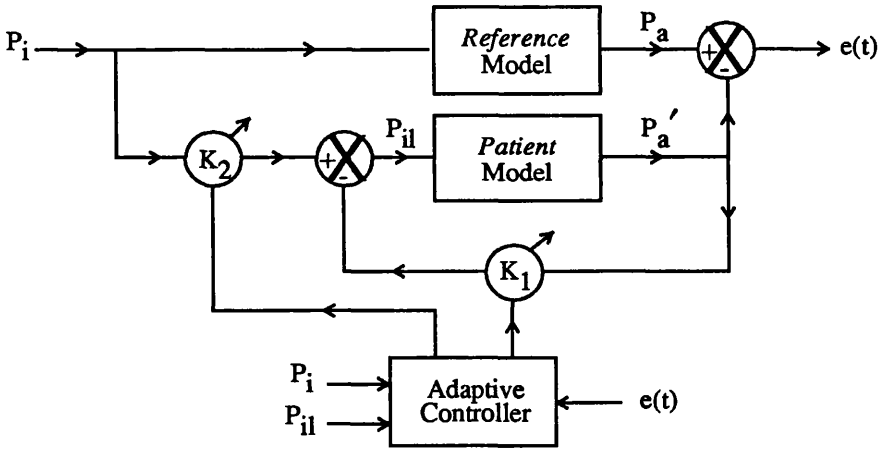


Fig. 7.1 Model-reference adaptive control scheme.

Initially, the fixed-parameter reference-model can be estimated from previous measurements of patient respiratory parameters. The model parameters can then be fine-tuned by using an on-line system identifications scheme. The key elements of respiratory system identification (Chapter 5) includes selection of model structure, experiment design, parameter estimation and validation (Ljung, 1987; Soderstrom and Stoica, 1989). This study has adopted the transfer function approach to system identification since it significantly simplifies the task of automatic and continuous measurement of system characteristics. The mathematical description of the reference-model is in terms of the input pressure P_i and output alveolar pressure P_a.

The patient-model is a variable parameter system which is not amenable to direct control. The patient-model is identical in form to the reference-model, however, the parameter values are time-variant. This assumption is a realistic representation of a true physiological system where its characteristics change with time. The patient-model has an input pressure P_{il} derived from the ventilator drive pressure and output alveolar pressure P_a'.

The error e(t) represents the difference between the outputs of the patient and reference-model. The adaptive controller has two parameters (i.e. the adaptive gain terms K_1 and K_2) that are changed according to the error and other inputs. Parameter adjustment based on the Lyapunov stability criterion guarantees that the system will remain stable and convergent provided certain conditions are satisfied (Astrom and Wittenmark, 1989; Narendra, 1989). The following section describes the dynamic model used to construct the MRAC system of Sec. 7.5.

7.4 Dynamic Modelling of the Respiratory System

Due to the complexity of the respiratory system, a detailed patient-model is difficult to achieve. The mechanical parameters concerned with ventilation are airway resistance, lung-tissue and chest-wall resistance, lung and chest-wall compliance, inertia of lung and chest-walls, and nonlinearities of the lungs. However, a second-order electrical RLC lung model provides an adequate representation of respiratory system dynamics and introduces greater complexity than the first-order RC model adopted by the majority of workers in this subject (Drazen et al., 1984). This study requires at least a second-order representation (Smith and Lin, 1989; Kabay et al., 1989) to take into account the acoustic resonances of the respiratory system (Sec. 4.4). In any case, the divergence of the selected model from the reality is exemplified by the assumption that the circuit elements are linear, lumped, deterministic and time-invariant. Also, no consideration is given to non-linearities due to turbulent flow, differences between inspiration and expiration, hysteresis and long-term effects, stress adaptation effects, a nonlinear threshold of alveoli opening, and a statistical distribution of the properties of some 300 million alveoli.

The first stage in developing the MRAC algorithm is to specify a mathematical model for the dynamic system we are attempting to control. Both the fixed parameter reference-model and the variable parameter patient-model are described in terms of the second order resistance-inertance-capacitance (RLC) electrical circuit shown below (Fig. 7.2).

The analogy between the physiological system and the electrical circuit is interpreted with pressure to voltage, flow to current, and volume to electric charge. Flow resistance is analogous to electrical resistance, compliance to electrical capacitance, and gas inertial effect to electrical inductance.

The dynamics of ventilation starts when a pressure is applied to the airway forcing gas flow into the lungs. As the lung and chest wall expand a counter-pressure is developed, because of their elastic properties, which tends to expel the inspired gas. If this counter-pressure equals that of the applied pressure, no more gas will enter into the lungs. If the counter-pressure exceeds the applied pressure, gas will flow out of the lungs as expiration commences.

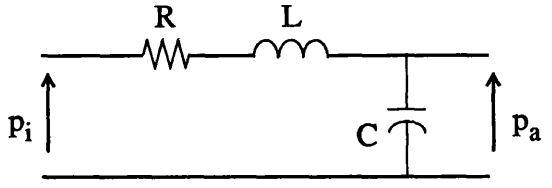


Fig. 7.2 Second order electrical analogue of the patient respiratory system.

The parameters of the electrical model include the total airway resistances R ($cmH_2O\ s\ l^{-1}$), airways and chest wall inertance L ($cmH_2O\ l^{-1}s^2$), lung and chest wall compliances C ($l\ cmH_2O^{-1}$). The ventilator output pressure P_i is the stimulus and the patient alveolar pressure P_a is the response of the electrical circuit. The alveolar pressure is indicative of respiratory dynamics, and is used as a control parameter for the design of a self-tuning ventilator.

The differential equation for the alveolar pressure in the reference-model is:

$$(LC)\,\ddot{p}_a(t) + (RC)\,\dot{p}_a(t) + p_a(t) = p_i(t) \qquad (7.1)$$

A second order system of this form will exhibit resonance depending on the Q-factor of the circuit. This model provides a realistic description of respiratory dynamics as it possesses the essential characteristic that the respiratory system behaves like an acoustic resonator.

Expressing this in state variable form with $\tau_1 = \frac{R}{L}$ and $\tau_2 = \frac{1}{LC}$:

$$\dot{p}_a = p_{2a} \qquad (7.2)$$

and,

$$\dot{p}_{2a} = \tau_2(-p_a + p_i) - \tau_1 p_{2a} \qquad (7.3)$$

Similarly, the alveolar pressure in the patient-model can be written as:

$$\dot{p}'_a = p'_{2a} \tag{7.4}$$

and,

$$\dot{p}'_{2a} = \tau'_2 \ (-p'_a + p_{il}) - \tau'_1 p'_{2a} \tag{7.5}$$

where $p_{il}(t)$ is the input pressure to the patient, which is set according to the adaptive controller equation:

$$p_{il}(t) = K_2(t)\, p_i(t) - K_1(t)\, p'_a(t) \tag{7.6}$$

where $K_1(t)$ and $K_2(t)$ are the adaptive controller gains. Substituting equation (7.6) for $p_{il}(t)$ in equation (7.5) we get:

$$\dot{p}'_{2a} = \tau'_2 [\ K_2(t)\, p_i(t) - (1 + K_1(t))\, p'_a(t)\ ] - \tau'_1\, p'_{2a} \tag{7.7}$$

Comparing with equation (7.3), we see that the patient-model differential equation is controlled by the adaptive gain terms.

7.5 Design of the Adaptive Control Algorithm

The adaptive controller is based on the Lyapunov redesign of the MRAC system (Parks, 1966; Phillipson, 1967). The main advantage of the method is that the asymptotic stability and convergence is guaranteed. Hang and Parks (1973) give a brief survey of the various design rules that can be used to implement model-following control systems. Kaufman et al. (1984) have verified the success of MRAC for regulating the infusion rate of drug in order to maintain desired blood pressure. However, their system is based on a first-order model. This section will develop a second-order MRAC for use in automatic ventilation.

Following from the definition of the dynamic equations of Sec. 7.4, an error differential equation can be defined as:

$$\ddot{e} = \dot{p}_{2a} - \dot{p}'_{2a} \tag{7.8}$$

where, $\dot{e} = p_{2a} - p'_{2a}$ in terms of the instantaneous alveolar pressure difference between the reference- and patient-models.

Substituting for $\dot{p}_{2a}$ and $\dot{p}'_{2a}$ from equations (7.3) and (7.7) respectively gives:

$$\ddot{e} = \tau_2(-p_a + p_i) - \tau_1\dot{p}_a - \tau'_2[K_2 p_i(t) - (1 + K_1(t))\, p'_a(t)] + \tau'_1\dot{p}'_a$$

$$= (-\tau_1\dot{e} - \tau_2 e + (\tau'_2 - \tau_2 + K_1)\, p'_a + (\tau'_1 - \tau_1)\dot{p}_a + (\tau_2 - \tau'_2 K_2)\, p_i)$$

Let: $$\alpha = [\tau'_2 - \tau_2 + K_1(t)]$$

and, $$\beta = [\tau_2 - \tau'_2 K_2(t)] \qquad (7.9)$$

then we have: $$\ddot{e} = -\tau_1\dot{e} - \tau_2 e + \alpha p'_a + \beta p_i + (\tau'_1 - \tau_1)\dot{p}'_a \qquad (7.10)$$

Making the tentative choice of Lyapunov function V in quadratic form:

$$V = \frac{1}{2}\left((k\tau_1 + k_d\tau_2)\, e^2 + 2ke\dot{e} + k_d\dot{e}^2 + \frac{1}{\gamma}(\alpha + \phi(ke + k_d\dot{e})\, p'_a)^2 + \frac{1}{\lambda}(\beta + \theta(ke + k_d\dot{e})\, p_i)^2\right) \qquad (7.11)$$

where the constants $\gamma>0$, $\lambda>0$, $\phi\geq 0$, and $\theta\geq 0$. This choice of Lyapunov function uses proportional plus derivative control on the error signal to control the dynamic response of the error signal when high adaptive gains γ and λ are required for closed-loop stability. In order for the function V in equation (7.11) to be a true Lyapunov function, the derivative of V must be negative semi-definite. Differentiating equation (7.11) leads to:

$$\dot{V} = (k\tau_1 + k_d\tau_2)\, e\dot{e} + k\dot{e}^2 + ke\ddot{e} + k_d\dot{e}\,\ddot{e} + \frac{1}{\gamma}(\alpha + \phi\varepsilon p'_a)\left(\dot{\alpha} + \phi\frac{d}{dt}(\varepsilon p'_a)\right) + \frac{1}{\lambda}(\beta + \theta\varepsilon p_i)\left(\beta + \theta\frac{d}{dt}(\varepsilon p_i)\right) \qquad (7.12)$$

where, $$\varepsilon = ke + k_d\dot{e} \qquad (7.13)$$

The choice of a proportional plus derivative form for the error function ε is significant, since it allows the transient error fluctuations to be controlled to provide a given dynamic response. This form of the error function complicates the form of the Lyapunov function and care should be taken in selecting more complex error functions.

Substituting for ë, equation (7.10), into equation (7.12) and selecting the adaptive laws:

$$\dot{\alpha} = -\gamma \varepsilon p_a' - \phi \frac{d}{dt}(\varepsilon p_a')$$

and,

$$\dot{\beta} = -\lambda \varepsilon p_i - \theta \frac{d}{dt}(\varepsilon p_i) \qquad (7.14)$$

which leads to a $\dot{V}$ described by:

$$\dot{V} = -(k_d \tau_1 - k)\dot{e}^2 - k\tau_2 e^2 - \phi(\varepsilon p_a')^2 - \theta(\varepsilon p_i)^2 - (\tau_1 - \tau_1')\varepsilon \dot{p}_a'$$

where $\dot{V}$ is always <0 for $k_d\tau_1 > k$, $\tau_1 > \tau_1'$ and $|e| \neq 0$. This condition enables V to satisfy the requirements of a Lyapunov function, and the asymptotic stability of the error e(t) is also ensured.

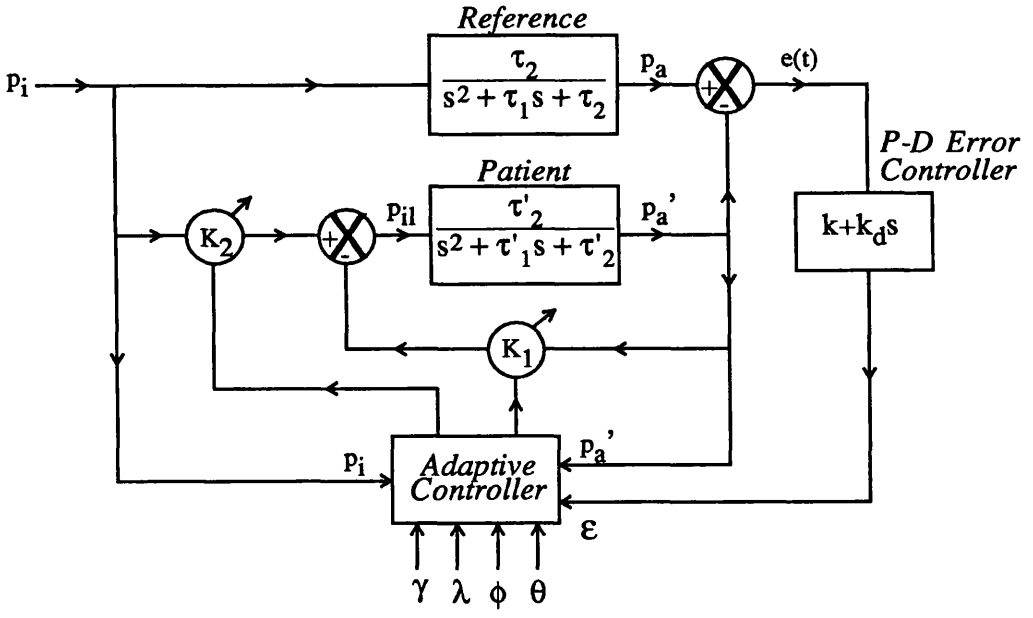


Fig. 7.3 Second order model-reference adaptive controller.

From the definitions of α and β, equation (7.9), the adaptive controller gains are given by:

$$\dot{K}_1(t) = -\gamma \varepsilon p_a' - \phi \frac{d}{dt}(\varepsilon p_a')$$

$$\dot{K}_2(t) = \frac{\lambda}{\tau_2'} \varepsilon p_i + \frac{\theta}{\tau_2'} \frac{d}{dt}(\varepsilon p_i) \qquad (7.15)$$

In differentiating equation (7.9), the rate of change of the patient-model parameters τ_1' and τ_2' are assumed to be negligible in comparison with the rate of change of the adaptive controller gains $K_1(t)$ and $K_2(t)$. The block diagram of the complete system is shown in Fig. 7.3.

7.6 Software Simulation

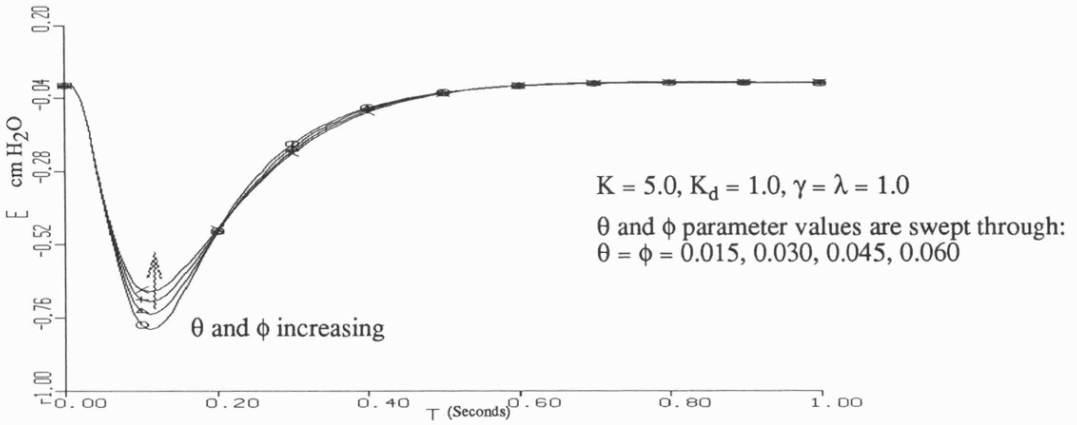
This section describes a simulation study of the second order MRAC system described above. The advanced continuous simulation language (ACSL), with a fourth-order Runge-Kutta numerical integration algorithm, was used to solve the non-linear integral and differential equations of the MRAC system (Appendix 6). The simulation investigates the performance of the self-tuning ventilator algorithm in supporting a hypothetical patient-model, without involving a real patient. This approach to system validation was adopted since ACSL offers a powerful high-level programming environment for rapid prototyping and fine-tuning of system parameters under varying operating conditions. Based on the results of this study, the control algorithm can be ported with confidence onto the target system described in Chapter 5, where low-level implementation details must be considered.

The parameters of the patient-model, described by equations (7.4-7.6), was set from previously published data (Young, 1989). In this way, the system performance can be tested for many groups of patients with varying physiology. The patient-model used in this study represents an adult with normal respiratory function. The total airway resistances, airways and chest wall inertance, and lung and chest wall compliance were fixed at R=10 $cmH_2O\ s\ l^{-1}$, L=0.153 $cmH_2O\ l^{-1}s^2$ and C=0.05 $l\ cmH_2O^{-1}$, respectively.

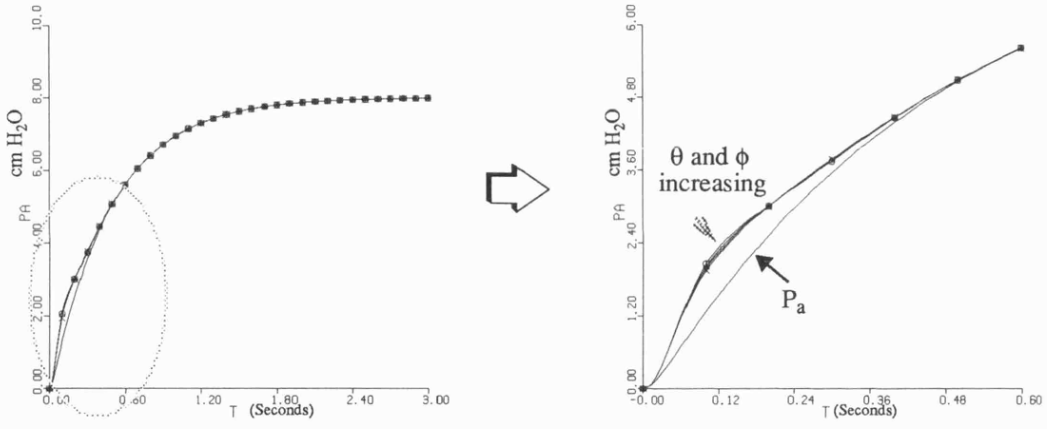
7.6.1 Methods

The initial conditions of the control system was calculated from the assumption that the patient- and reference-models are matched at the start of simulation. In this case, at t=0 the error e(t)=0 which implies that the input pressure to both models are identical $p_{il}(t) = p_i(t)$ and the adaptive gains must be set to $K_1=1.0$ and $K_2=0$.

The simulation was initiated with a step input pressure pulse of 8 cmH_2O. The time-varying characteristics of the patient can be simulated by programming a step change in patient model parameters. For this study, the patient compliance C′ was halved at the start of simulation from 0.050 $l\ cmH_2O^{-1}$ to 0.025 $l\ cmH_2O^{-1}$, whilst the patient resistance (R′) and inertance (L′) remained constant. However, the simulation program and algorithm can be run for any combination of parameter variations without discrimination.

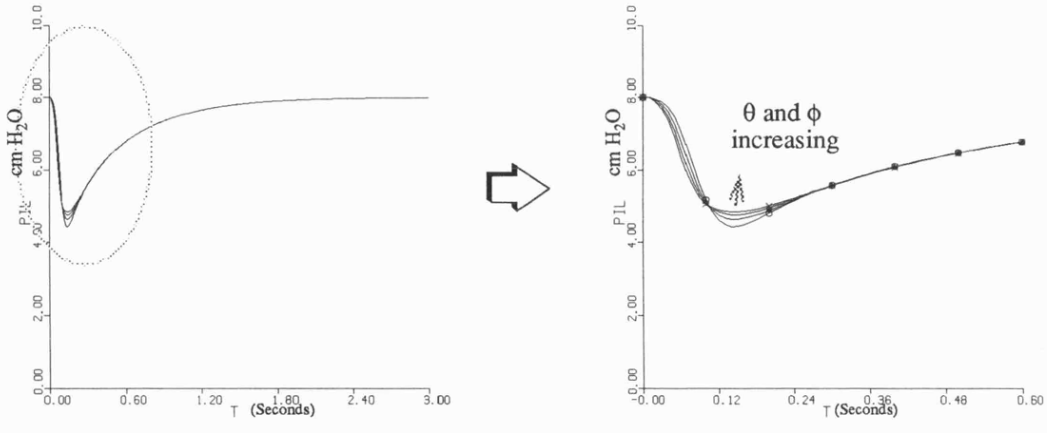


(a) The error (E) signal between P_a and P_a' vs time (T).



(b) Alveolar pressure P_a and P_a' vs time.

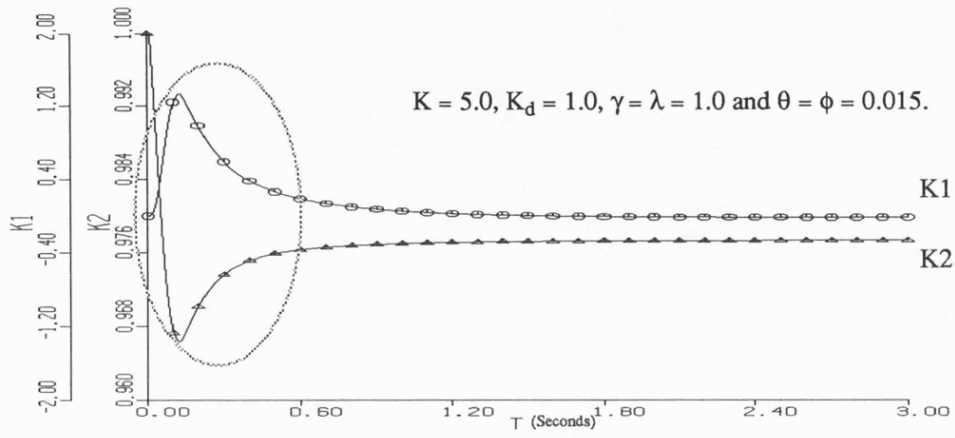
(c) Zoomed version of (b).



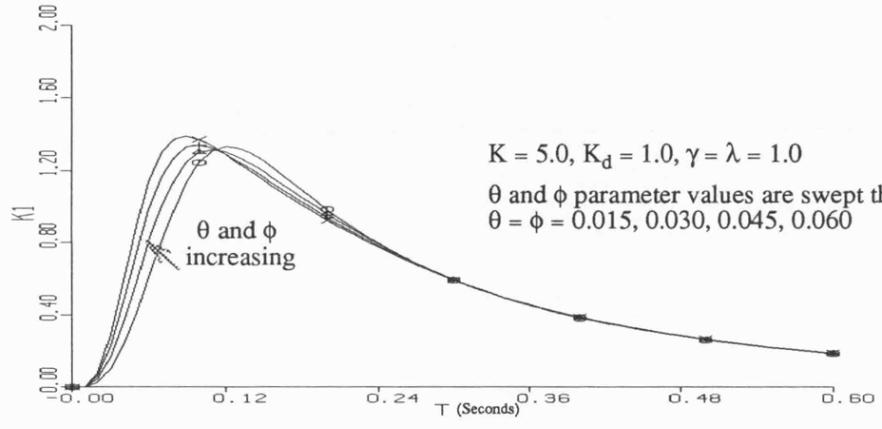
(d) Patient input pressure P_{il} vs time.

(e) Zoomed version of (d).

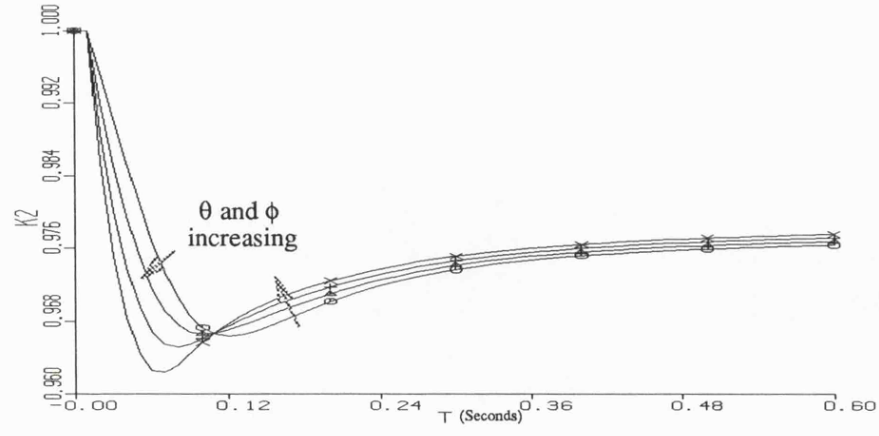
Fig. 7.4 MRAC system dynamic response for θ-, ϕ-sweep. The total resistance R=10 cmH_2O s l^{-1}, airways and chest wall inertance L=0.153 cmH_2O $l^{-1}s^2$, and lung and chest wall compliance C=0.05 l cmH_2O^{-1}. The simulation was started with a step-input pressure of P_i=8 cmH_2O and a step change in the patient-model compliance C=0.025 l cmH_2O^{-1}.



(a) The variation in adaptive controller gains K_1 and K_2 vs time.



(b) The variation in adaptive gain K_1 vs time.



(c) The variation in adaptive gain K_2 vs time.

Fig. 7.5 MRAC system dynamic response for conditions described in Fig. 7.4; shows the variation in the adaptive controller gains K_1 and K_2 for various θ and ϕ parameters.

The MRAC system (Fig. 7.3) comprises the patient-model, reference-model, and a proportional+derivative (P+D) error controller which feeds the adaptive controller. The adaptive controller inputs include the reference-model input pressure $p_i(t)$, the patient alveolar pressure $p_a'(t)$, the P+D error signal $\varepsilon(t)$, and the adaptive gain-constants $(\theta, \phi, \gamma, \lambda)$ of equation (7.11).

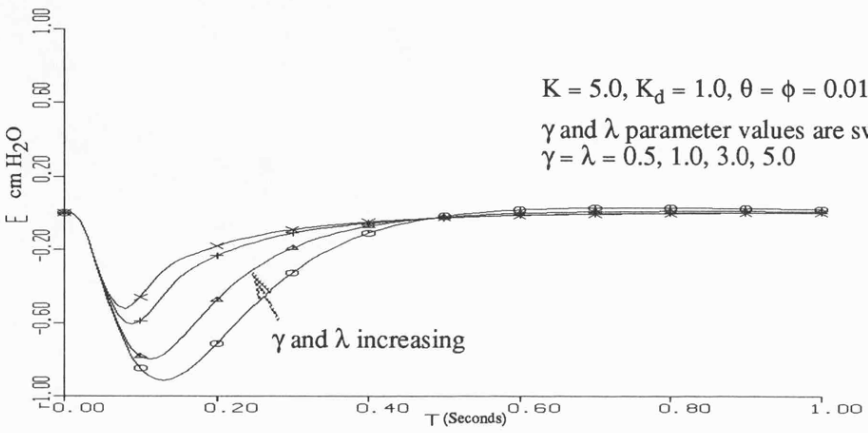
The simulation was run with single-step and periodic square-wave pressure inputs and the system performance monitored for various parameter settings for the adaptive controller and P+D-error controller. The default values of all parameters used during the simulation are set to $K=5.0$, $K_d=1.0$, $\theta=\phi=0.015$ and $\gamma=\lambda=1.0$, unless where otherwise stated.

The results presented in the following sections adopt a common presentation format where MRAC state-variables are plotted against time for different parameter values. In particular, Figs. 7.4, 7.6, 7.8 and 7.10 (a)-(e) plot the reference-to-patient alveolar pressure error signal e(t), the alveolar pressure of the reference-model $p_a(t)$ for varying patient-model pressure $p_a'(t)$, and the input pressure to the patient-model $p_{il}(t)$, respectively. Figs. 7.5, 7.7, 7.9 and 7.11 plot fluctuations in adaptive gains K_1 and K_2 for the range of parameter values with time. Some plots have a dashed ellipse to highlight regions of a curve which are subsequently *zoomed-in* on to enhance details.

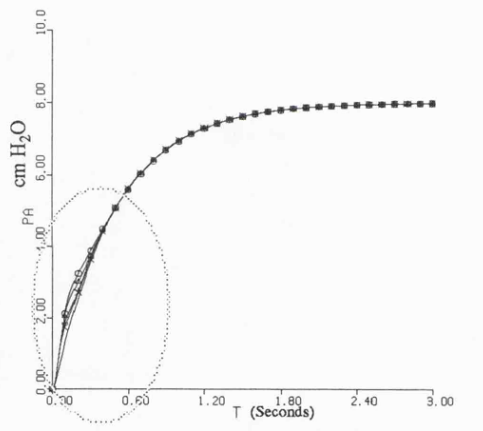
7.6.2 Adaptive-Controller Parameter Variations

A. θ- and ϕ-Parameters

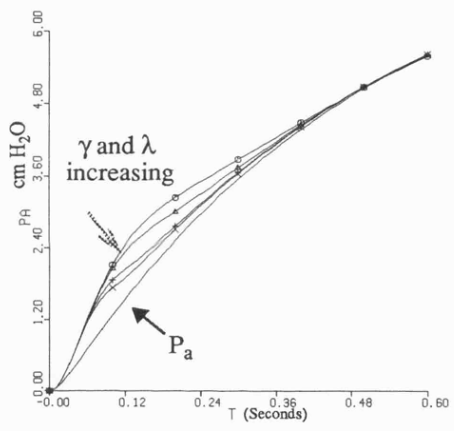
Fig. 7.4-7.5 show the results of the simulation for θ– and ϕ–parameter sweeps ($\theta = \phi = 0.015, 0.030, 0.045, 0.060$). Fig. 7.4b-c shows the deviation between the patient- and reference-model alveolar pressures due to the change in patient compliance C'. Although the peak error signal (Fig. 7.4a) is seen to decrease with increasing θ, the error is not too sensitive to the variations in θ and the patient alveolar pressure converges rapidly toward the reference output for all parameter values. As expected, the patient alveolar pressure is greater than the reference model output (Fig. 7.4c) due to the reduced patient compliance C'. Fig. 7.4d-e shows how the input pressure to the patient drops to compensate for this increase in patient alveolar pressure. As the adaptive gains K_1 and K_2 stabilise about their equilibrium values (Fig. 7.5a-c), the input pressure to the patient exponentially tends to that of the reference-model ($p_{il}(t) = p_i(t) = 8.0\ cmH_2O$). The rate of change of K_1 and K_2 is seen to increase with increasing θ, but in all cases the MRAC response shows quick adaptation (<0.6s) and asymptotic stability.



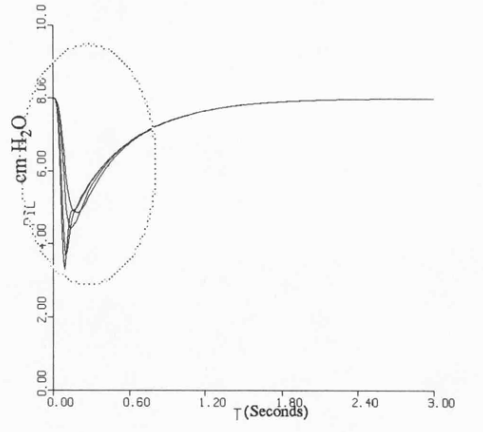
(a) The error (E) signal between P_a and P_a' vs time (T).



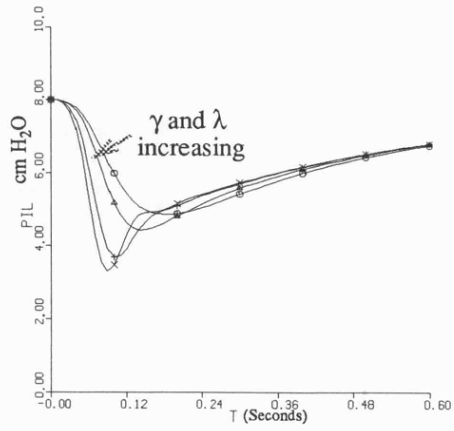
(b) Alveolar pressure P_a and P_a' vs time.



(c) Zoomed version of (b).

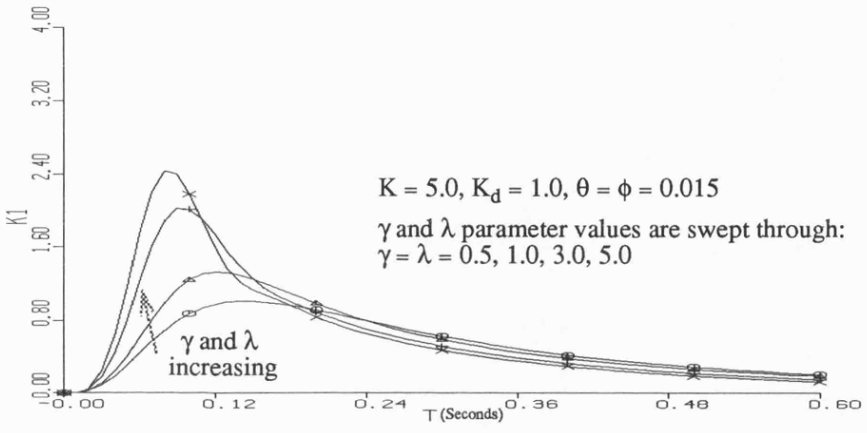


(d) Patient input pressure P_{il} vs time.

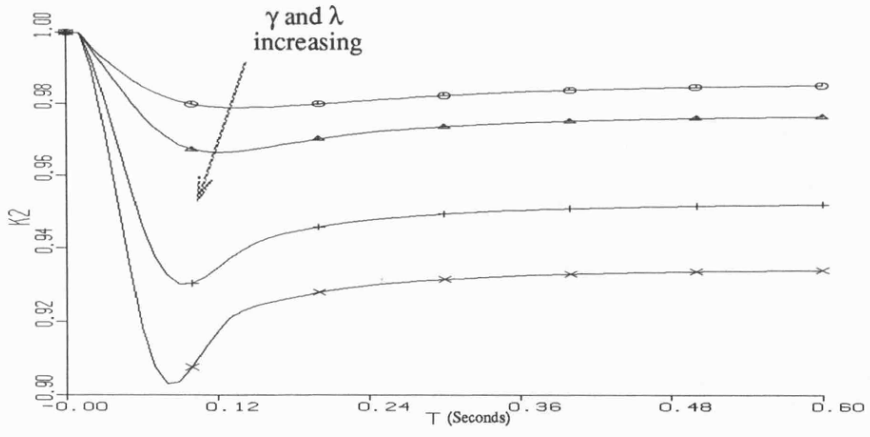


(e) Zoomed version of (d).

Fig. 7.6 MRAC system dynamic response for γ-, λ-sweep. The total resistance R=10 cmH_2O s l^{-1}, airways and chest wall inertance L=0.153 cmH_2O $l^{-1}s^2$, and lung and chest wall compliance C=0.05 l cmH_2O^{-1}. The simulation was started with a step-input pressure of P_i=8 cmH_2O and a step change in the patient-model compliance C=0.025 l cmH_2O^{-1}.

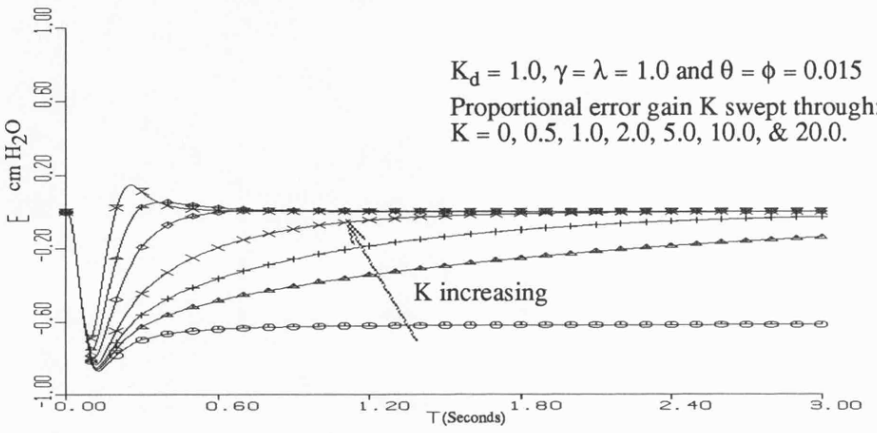


(a) The variation in adaptive gain K_1 vs time.

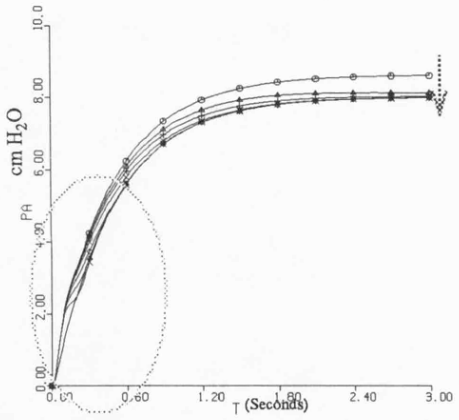


(b) The variation in adaptive gain K_2 vs time.

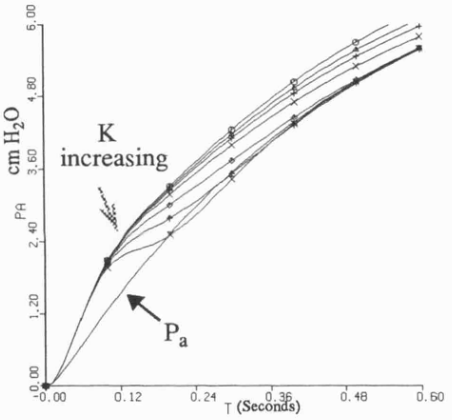
Fig. 7.7 MRAC system dynamic response for conditions described in Fig. 7.6; shows the variation in the adaptive controller gains K_1 and K_2 for various γ and λ parameters.



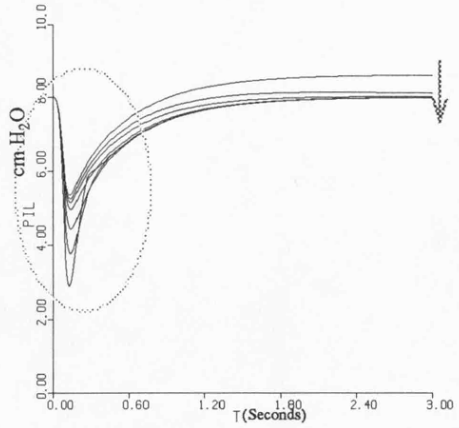
(a) The error (E) signal between P_a and P_a' vs time (T).



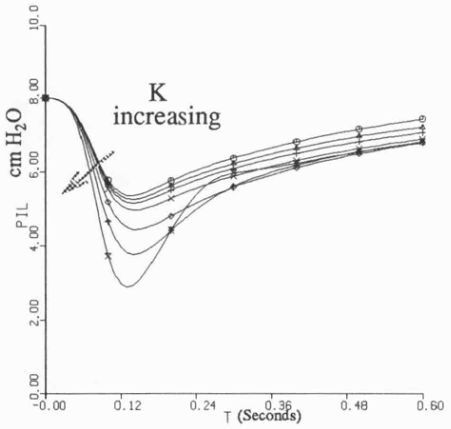
(b) Alveolar pressure P_a and P_a' vs time.



(c) Zoomed version of (b).

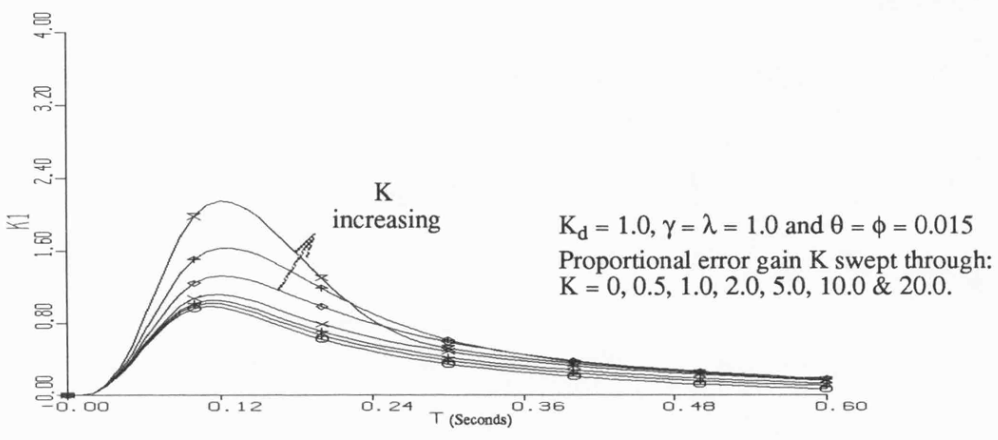


(d) Patient input pressure P_{il} vs time.

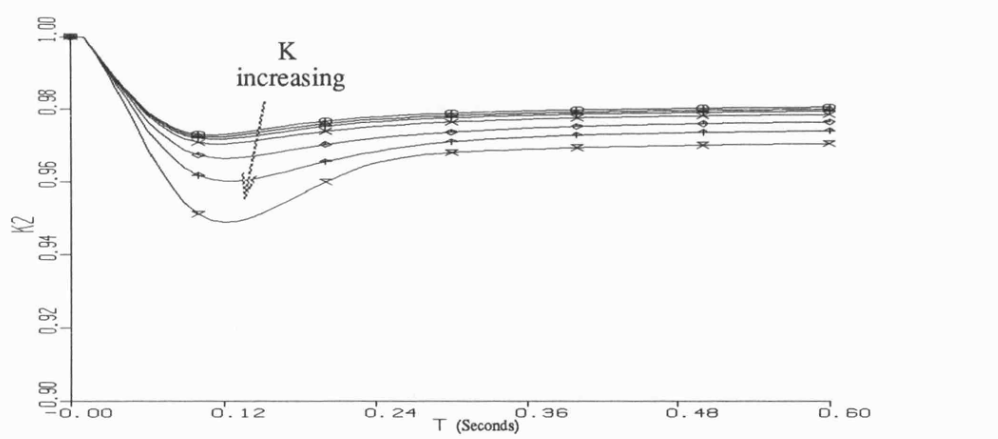


(e) Zoomed version of (d).

Fig. 7.8 MRAC system dynamic response for K-sweep. The total resistance R=10 cmH_2O s l^{-1}, airways and chest wall inertance L=0.153 cmH_2O $l^{-1}s^2$, and lung and chest wall compliance C=0.05 l cmH_2O^{-1}. The simulation was started with a step-input pressure of P_i=8 cmH_2O and a step change in the patient-model compliance C=0.025 l cmH_2O^{-1}.



(a) The variation in adaptive gain K_1 vs time.



(b) The variation in adaptive gain K_2 vs time.

Fig. 7.9 MRAC system dynamic response for conditions described in Fig. 7.8; shows the variation in the adaptive controller gains K_1 and K_2 for various K-values.

B. γ- and λ-Parameters

Fig. 7.6-7.7 show the results of the simulation for γ– and λ–parameter sweeps ($\gamma = \lambda = 0.5, 1.0, 3.0, 5.0$). Fig. 7.6b-c shows the deviation between the patient- and reference-model alveolar pressures resulting from the step change in patient compliance C'. The patient alveolar pressure is again seen to converge rapidly toward the reference output for all parameter values. However, the error signal (Fig. 7.6a) shows a marked reduction in peak error for increasing γ - the error reduces by almost 50% across the range γ=0.5 to γ=5.0. The rate of decay of the error is also seen to be more rapid with increasing γ which results in improved convergence between the model outputs.

Again the patient alveolar pressure is greater than the reference model (Fig. 7.6c) due to the reduced patient compliance C' and this is reflected in Fig. 7.6d-e where the adaptive controller reduces the input pressure to the patient to compensate. The adaptive gains K_1 and K_2 are adjusted as shown in Fig. 7.7a-b. Increasing γ produces a sharper and steeper fall in the patient input pressure 7.6e, however, it also recovers more rapidly than the case for θ– and ϕ–parameter sweeps (Sec. 7.6.2A).

7.6.3 Proportional+Derivative-Controller Gain Variations

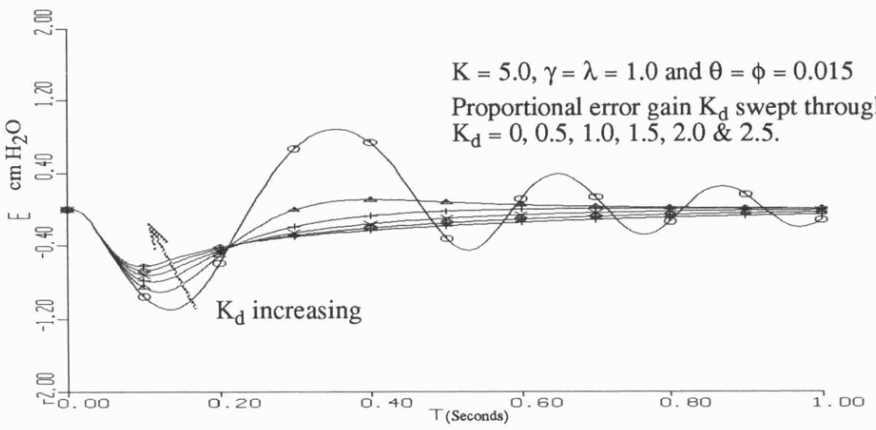
A two-term proportional+derivative (P+D) error controller was used to improve the dynamic characteristics and convergence of the adaptive system. The MRAC performance and its variation with P+D gain is described below.

A. Proportional K-Parameter

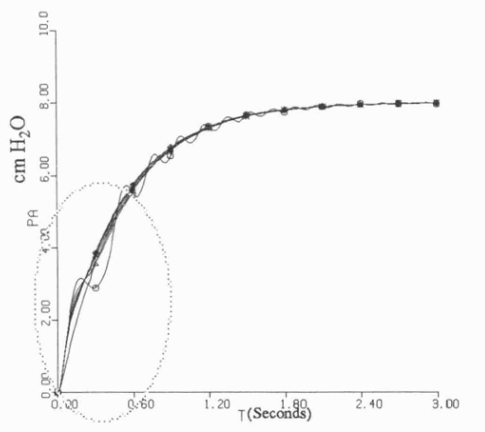
Fig. 7.8-7.9 show the results of the simulation for K–parameter sweeps (K = 0, 0.5, 1.0, 2.0, 5.0, 10.0, 20.0). With K=0, the error signal is subject to pure derivative action. Fig. 7.8a shows how the over-damped error response gradually becomes more oscillatory with increasing proportional gain (K). This is expected since an increased K will produce a faster change in response for a given magnitude of error. Too large a K will lead to increased oscillations and instability. The MRAC system is seen to adapt more rapidly with increasing K (Fig. 7.8a-c). The adaptive gains K_1 and K_2 are adjusted as shown in Fig. 7.9.

B. Derivative K_d-Parameter

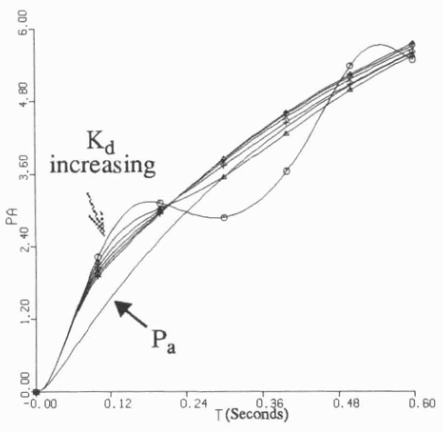
Fig. 7.10-7.11 show the results of the simulation for K_d–parameter sweeps (K_d = 0, 0.5, 1.0, 1.5, 2.0, 2.5). With K_d=0, the error signal is subject to pure proportional action. Fig. 7.10a shows how the under-damped oscillatory error response gradually gets damped with increasing derivative gain (K_d). The derivative term contributes a predictive type of control action, where the output of the controller is modified when the error is changing



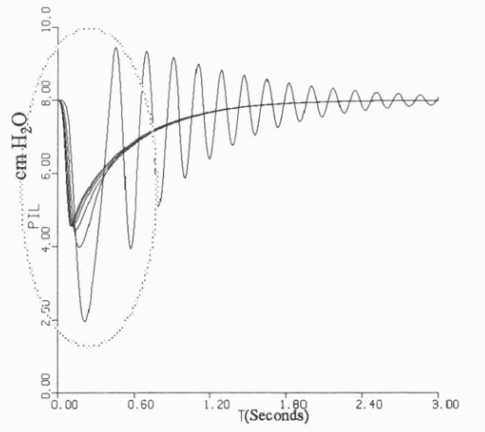
(a) The error (E) signal between P_a and P_a' vs time (T).



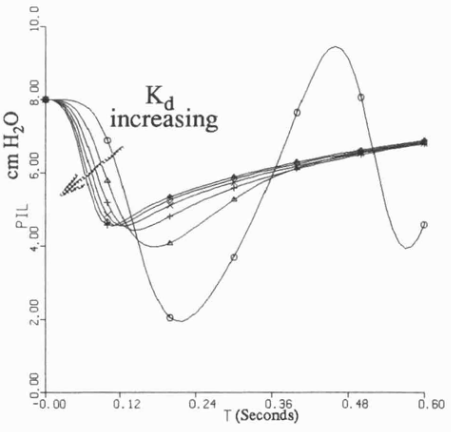
(b) Alveolar pressure P_a and P_a' vs time.



(c) Zoomed version of (b).

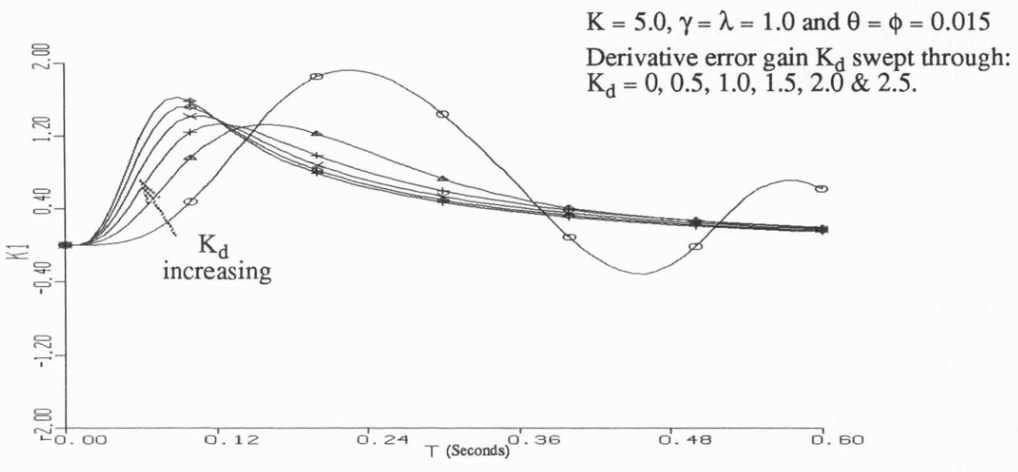


(d) Patient input pressure P_{il} vs time.

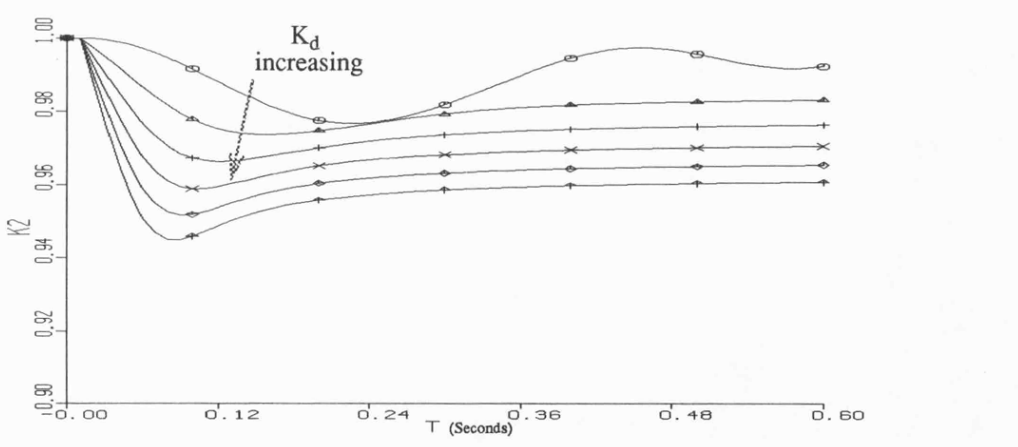


(e) Zoomed version of (d).

Fig. 7.10 MRAC system dynamic response for K_d-sweep. The total resistance R=10 cmH_2O s l^{-1}, airways and chest wall inertance L=0.153 cmH_2O $l^{-1}s^2$, and lung and chest wall compliance C=0.05 l cmH_2O^{-1}. The simulation was started with a step-input pressure of P_i=8 cmH_2O and a step change in the patient-model compliance C=0.025 l cmH_2O^{-1}.

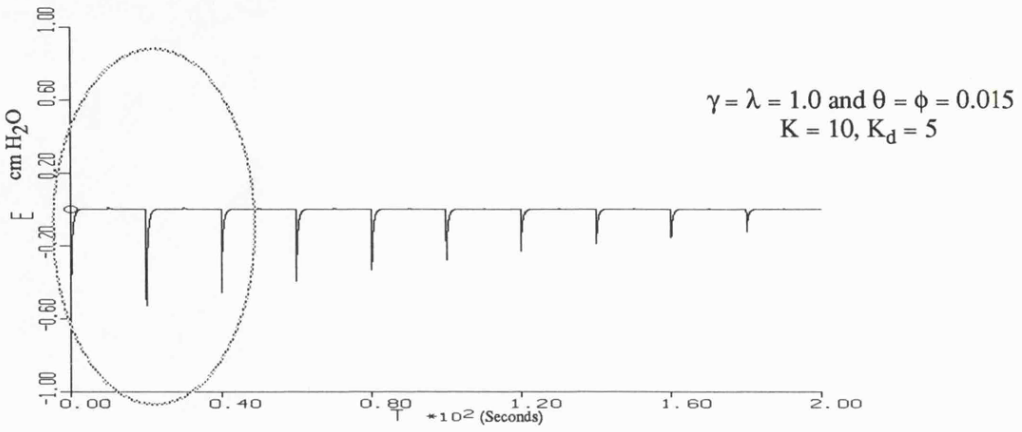


(a) The variation in adaptive gain K_1 vs time.

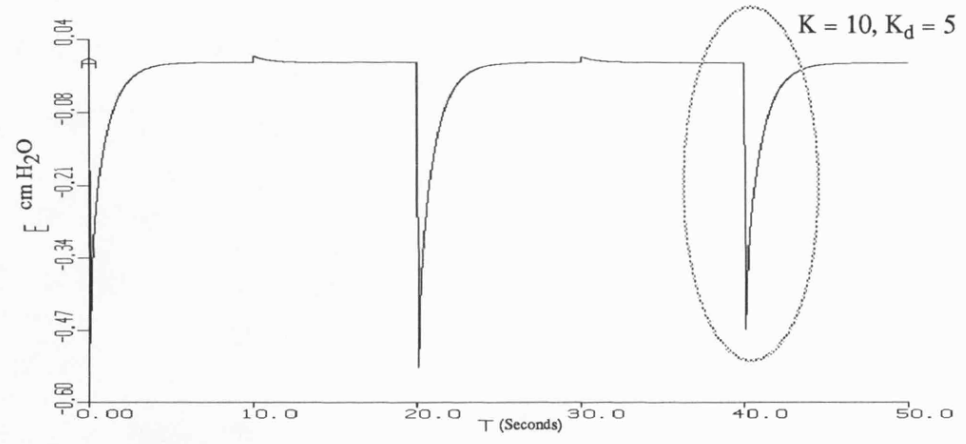


(b) The variation in adaptive gain K_2 vs time.

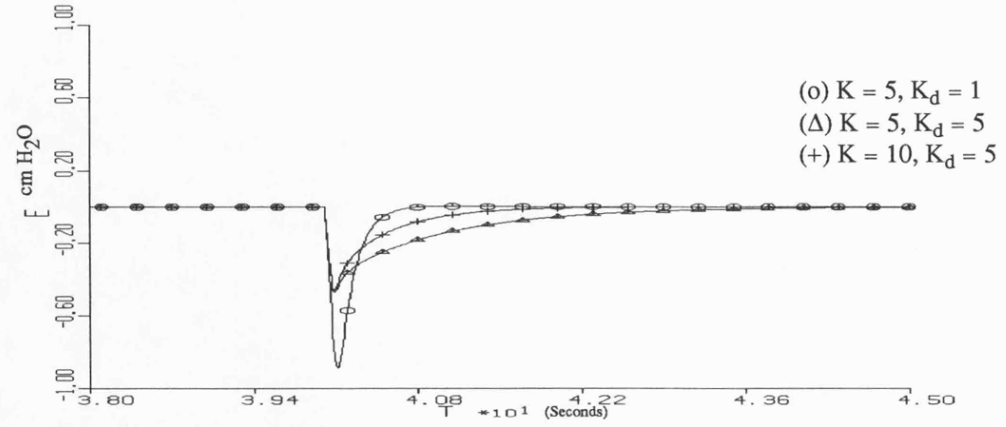
Fig. 7.11 MRAC system dynamic response for conditions described in Fig. 7.10; shows the variation in the adaptive controller gains K_1 and K_2 for various K_d-values.



(a) The attenuation of the error (E) between P_a and P_a' vs time (T).



(b) Zooming on the error (E) signal of (a).



(c) Zooming on the error (E) signal of (b) for different combinations of K and K_d.

Fig. 7.12 MRAC system dynamic response shows the relationship between E and the PD-error controller gains K-K_d.

rapidly, thus anticipating a large overshoot and making some corrective action before it occurs. When the system is moving towards a state of zero error (Fig. 7.10a-c), then e and $\dot{e}(t)$ have opposite signs; hence K_d reduces the magnitude of the overall error $\varepsilon(t)$ (Fig. 7.3) and thus reduces the signal that is accelerating $p_a'(t)$ towards the zero error state. When $p_a'(t)$ has overshot or undershot and is moving away e=0, then e and $\dot{e}(t)$ have the same sign and K_d augments the decelerating signal. As the system settles, $\dot{e}(t)$ tends to zero and K_d has no further influence.

7.6.4 Periodic Pressure Wave Excitation

The MRAC system was also tested for a periodic square wave pressure input (0-8 cmH_2O swing, period = 20s and an inspiratory:expiratory ratio = 1:1). The resulting error signal between patient- and reference-models was calculated over a 200 s time interval. Fig. 7.12a-b shows how the error signal attenuates with time. The effect of the P+D error controller gains $K-K_d$ on the rate of attenuation of the error signal can be seen in Fig. 7.12c.

The attenuation of the error signal can be adjusted to decay even more rapidly by varying the θ-, ϕ-, γ-, and λ-parameters of the adaptive controller. In any case, the characteristics shown in Fig. 7.12 are acceptable since patients receiving anaesthesia are normally ventilated for several hours.

7.7 Discussion

This chapter has investigated the physiological factors affecting gas exchange during artificial ventilation and specified criteria for selecting control variables for implementing automatically controlled ventilation. Several control variables have been considered for this task and their relative merits summarised.

A second order model reference adaptive control (MRAC) scheme has been designed and simulated for implementing self-tuning automatic control of high-frequency ventilation. The feasibility of the adaptive ventilator circuit has been validated by a simulation study investigating its performance under various conditions. The results indicate the stability and convergence of this adaptive scheme for a time-varying patient-model - in agreement with theoretical predictions.

The MRAC technique described in this chapter is based on the assumption that the patient alveolar pressure can be monitored. Although this assumption is impractical at present, it is proposed that techniques described in other parts of this thesis may be used to establish

alternative measures of ventilation which could be used to estimate the patient alveolar pressure. For example, abdominal-wall displacements may prove to be a good estimator of the patient alveolar pressure (Chapters 5 and 6).

The practical realisation of an MRAC scheme for automatic ventilation and monitoring of the patient's respiratory state seems feasible. The first stage of this scheme would require a method for determining the respiratory dynamics of a patient so that the reference-model parameters can be initialised. The instrumentation described in Chapter 5 is well-suited to performing on-line *reference-model* identification. The real-time computational power of the instrument can also be exploited to implement a numerical technique for computing the MRAC system equations.

Clinical use of the MRAC ventilator should significantly improve the effectiveness and safety of artificial ventilation. Typically, the instrument would enter a learning phase where system identification of the patient respiratory dynamics and airways resonant frequency are performed using the techniques discussed in Chapter 5. Once the MRAC reference-model is initialised, the ventilator can be set to operate at the resonant frequency of the patient airways - corresponding to optimal ventilation. In this case, the input-output variables of the patient-model in Fig. 7.3 would then be replaced by *real* measurements from a *true* patient. The MRAC algorithm can then be implemented to perform in a manner similar to that shown in the simulation study (Sec. 7.6).

Application of the ventilator in a clinical environment would require certain safe-guards against excessive build-up of pressure in the patient airways. In cases where pressure levels exceed pre-set alarm levels it may be necessary to re-evaluate the reference-model so that it is more representative of the current patient physiology. For example, the MRAC scheme can be used to stabilise patient ventilation for small fluctuations about an equilibrium state. When the patient dynamics diverge significantly from this equilibrium, the instrument can be programmed to either re-adjust the reference-model and continue ventilation or set off an alarm to alert clinical staff of the danger.

7.8 References

Braunwald, E., Binion, J.H., Morgan, W.L. and Sarnoff, S.J. (1957): "Alterations in central blood volume and cardiac output induced by positive pressure breathing and corrected by metaraminol (Aramine)". *Circulation Res.*, **5**, 670-675.

Cournand, A., Motley, H.L., Werko, L. and Richards, D.W. (1948): "Physiological studies of the effects of intermittent positive pressure breathing on cardiac output in man". *Am. J. Physiol.*, **152**, 162-174.

Drazen, J.M., Kamm, R.D. and Slutsky, A.S. (1984): "High frequency ventilation", *Physiol. Revs.*, **64**, 505-543.

Enghoff, H. (1931): "Zur frage des schadlichen raumes bei der atmung". *Skand. Arch. Physiol*, **63**, 15.

Fowler, W.S. (1950): "Lung function studies. V: Respiratory dead space in old age and in pulmonary emphysema". *J. Clin. Invest.*, **29**, 1439.

Hang, C-C. and Parks, P.C. (1973): "Comparative studies of model reference adaptive control systems". *IEEE Trans. Automat. Contr.*, **AC-18**, n5, 419-428.

Jain, V.K. and Guha, S.K. (1972): "A control system for long-term ventilation of the lungs". *IEEE Trans. Biomed. Eng.*, **BME-19**, n1, 47-53.

Kabay, S., Jones, N.B. and Smith, G. (1989): "A system for real-time measurement and control in high frequency jet ventilation", In: *IFAC-BME, Decision support for patient management: measurement, modelling and control*, 151-160.

Kaufman, H., Roy, R. and Xu, X. (1984): "Model reference adaptive control of drug infusion rate". *Automatica*, **20**, n2, 205-209.

Smith, B.E. and Lin, E.S. (1989): "Resonance in the mechanical response of the respiratory system to high frequency jet ventilation", *Acta Anaesthesiol. Scand.*, **Supp. 90**, 65-69.

Lindorff, D.P. and Carroll, R.L. (1973): "Survey of adaptive control using Liapunov design". *Int. J. Control*, **18**, n5, 897-914.

Ljung, L. (1987): "*System identification: theory for the user*", Prentice Hall, New Jersey.

Mitamura, Y., Mikami, T., Sugawara, H. and Yoshimoto, C. (1971): "An optimally controlled respirator". *IEEE Trans. Biomed. Eng.*, **BME-18**, n5, 330-337.

Morgan, B.C., Martin, W.E., Hornbein, T.F., Crawford, E.W. and Guntheroth, W.G. (1966): "Hemodynamic effects of intermittent positive pressure respiration". *Anesthesiology*, **27**, 584-590.

Mushin, W.W., Baker, L.R-, Thompson, P.W. and Mapleson, W.W. (1980): *Automatic ventilation of the lungs*. (3rd edition) Blackwell Scientific Publications.

Narendra, K.S. (1989): *Stable adaptive systems*, Englewood-Cliffs, Prentice-Hall.

Nunn, J.F. (1987): *Applied respiratory physiology*. (3rd edition) Butterworth & Co. Ltd.

Otis, A.B. (1954): "The work of breathing". *Physiol. Rev.*, **34**, 449.

Parks, P.C. (1966): "Liapunov redesign of model reference adaptive control systems". *IEEE Trans. Automat. Contr.*, **AC-11**, n3, 362-367.

Phillipson, P.H. (1967): "Concerning Lyapunov redesign of model reference adaptive control systems". *IEEE Trans. Automat. Contr.*, **AC-12**, n4, 625.

Radford, E.P., Ferris, B.J. and Kriet, B.C. (1954): "Clinical use of nomogram to estimate proper ventilation during artificial respiration". *New Eng. J. Med.*, **251**, 877-884.

Radford, E.P. (1955): "Ventilations standards for use in artificial respiration". *J. Appl. Physiol.*, **7**, 451-460.

Soderstrom, T. and Stoica, P. (1989): "*System identification*", Prentice Hall, New Jersey.

Wald, A.A., Murphy, T.W. and Mazzia, V.D.B. (1968): "A theoretical study of controlled ventilation". *IEEE Trans. Biomed. Eng.*, **BME-15**, n4, 237-248.

Werko, L. (1947): "The influence of positive pressure breathing on the circulation in man". *Acta. Med. Scan. Suppl.*, **193**, 1-125.

West, J.B. (1977): *Ventilation/blood flow and gas exchange.* (3rd edition) Blackwell Scientific Publications.

Wittenmark, B. and Astrom, K.J. (1984): "Practical issues in the implementation of self-tuning control". *Automatica*, **20**, n5, 595-605.

Woo, J. and Rootenberg, J. (1972): "Digital simulation of an adaptive system for long-term ventilation of the lungs". *IEEE Trans. Biomed. Eng.*, **BME-20**, n6, 475-477.

Young, J.D. (1989): "Gas movement during high-frequency ventilation", *Acta Anaesthesiol. Scand.*, **Supp. 90**, 70-71.

Chapter 8
Discussion and Conclusions

INTRODUCTION

The research work described in this thesis has been concerned with the design and development of intelligent computer-based instrumentation that can be easily adapted for measurement and control in specific domains. The particular area selected for application of this instrumentation was in anaesthesia for the measurement and control of high-frequency jet ventilation (HFJV). The analytical methods and experimental procedures required for measurement and processing of data in this domain are also applicable to many other areas throughout engineering and biomedicine. Based on the results obtained from clinical studies using this instrumentation, the author has validated the hypothesis that the human respiratory airways exhibit characteristics similar to an acoustic resonant circuit by using a modified jet ventilator device. Simulation studies have also been performed to investigate the feasibility of providing automatically controlled patient ventilation.

A prototype general-purpose signal processing computer (SPC), described in Chapter 3, encompasses many new design concepts to provide a flexible and user-friendly system. The SPC philosophy was based on low technology devices arranged to provide a novel interface to standard hardware under software control. The personal contribution of the author has taken the SPC from a simulation tool to a fully functional multi-processing computer system. The main contributions have been to develop the SPC signal acquisition processor (SAP), host-to-SAP communications interface, signal acquisition card and

programmable anti-aliasing filters. These units were integrated with the host processor and dual plane memory to provide a flexible and extensible real-time instrumentation system.

A generic application program interface (API) was also written to configure the SPC into a general-purpose data-acquisition and signal processing system for use in different application areas. Following preliminary clinical trials of the instrument in HFJV, several disadvantages of the system were identified which compromised its practical use. However, these problems are related to the bandwidth limitations of the SPC hardware and can be overcome by using powerful, modern and cost-effective technology. These problems do not detract from the underlying design concepts of the SPC system architecture.

Since its initial conception, the hardware and software used to implement the SPC has been superseded with new development tools that offer greater functionality, higher bandwidth and faster development times - all at lower cost. Using personal computers with commercially-available add-on cards (e.g. for data-acquisition, digital signal processing and graphics output) and high-level software development tools, one can rapidly prototype flexible and user-friendly real-time instrumentation for data-acquisition and signal processing based around the SPC design.

HFJV INSTRUMENTATION

HFJV is a form of mechanical ventilatory support that differs from conventional modes of ventilatory support (IPPV) in both its relative tidal volume and respiratory rate. Despite its success over IPPV in many clinical applications, widespread acceptance of HFJV has been inhibited by a lack of:

- understanding of the underlying fluid dynamics;
- practical guidelines for clinicians on when to apply the technique; and
- criteria for adjusting ventilator settings such that alveolar gas exchange is optimised.

One objective of this study was to develop instrumentation and measurement techniques to improve our understanding of the underlying respiratory dynamics of patient airways during HFJV - to determine the efficacy of this mode of ventilation.

Drawing on past experience with the SPC design, the author has developed an IBM-PC based system for real-time measurement, modelling and control of HFJV in anaesthesia. The system uses special hardware and real-time signal processing algorithms implemented around a flexible generic kernel to produce a second generation SPC.

In order to use a general-purpose personal computer as the primary interface to specialised instrumentation, the man-machine interface (MMI) becomes an essential component of the system since it provides an intuitive high-level medium of interaction between user and instrument. A generic MMI was developed for this study which encapsulates the functionality of the system into a graphics-based control-panel that mimics the buttons and displays as seen on a traditional instrument panel. The MMI allows the user to fully manipulate the system without worrying about the advanced underlying architecture of the instrument.

The other component of the measurement system was the high-frequency jet ventilator. In its initial form, the ventilator could only be used to generate binary pressure pulses with variable amplitude, period and mark-to-space ratio. This was a limitation since the objectives of the study was to determine the dynamics of the respiratory system in an efficient and effective manner. For this reason, the ventilator hydraulic and electrical circuits were modified such that band-limited white-noise pressure stimuli could be generated. The ventilator now permits systematic identification of respiratory dynamics with extremely high precision in a fraction of the time taken by previous workers in this field. This is achieved with only minimal changes to existing jet ventilation equipment and procedures. The system is intended to cope with the volume of information that needs to be considered during HFJV and the level of complexity that this method of ventilation entails.

The identification procedure adopted in this study requires continuous monitoring of:

- jet-ventilator drive pressure;
- patient airways pressure; and
- abdominal and thoracic wall displacements.

The first two variables can be easily monitored using existing pressure transducers. However, traditional methods of measuring wall displacement are based either on a strain-gauge transducer or an inductance plethysmograph. Both methods suffer from severe disadvantages which greatly degrades the quality of measurements. For this reason, a new fibre-optic displacement transducer was specifically developed to overcome these problems.

The fibre-optic transducer provides a non-invasive method of measuring chest wall displacements, greater versatility, relative freedom from artefact, high resolution and linearity over a wide bandwidth. The dynamic and static performance of the system was determined and found to be acceptable for the purposes of this study.

CLINICAL STUDIES

Having developed the instrumentation and tools to investigate the physiological and dynamical affects of HFJV on patient respiration, a clinical study was performed on a small population of critically-ill patients to:

- investigate the performance of the newly developed computer-based environment for simultaneous ventilation and identification of patient respiratory dynamics during HFJV;
- test the hypothesis that the respiratory airways behave as an acoustic resonant circuit which exhibits resonance over the range of frequencies covered by HFJV;
- identify any characteristic features of the respiratory system behaviour during HFJV which may need to be considered for implementing automatically controlled patient ventilation.

The performance of the instrumentation was found to significantly improve the speed and accuracy with which data could be captured and analysed compared to previous techniques. The MMI was sufficiently user-friendly for an anaesthetist with some basic computing skills to successfully perform his normal duties and carry out the measurements recorded in the case studies (Chapter 6).

The study clearly validated the hypothesis that the respiratory airways exhibit characteristics similar to an acoustic resonant circuit. In all cases, except case study P04 (Sec. 6.6), a prominent resonant peak could be observed from the transfer function magnitude spectra. Confidence in these results was reinforced by a strong coherency spectrum throughout the frequency span of measurements.

An observation from two of the case studies worth noting is the presence of what appears to be two *unique* resonant peaks which are features of the airways. The high coherency spectrum would support this view. However, an alternative view might assume that the second resonant peak is an artefact that is inadvertently produced by averaging. To elaborate, if we assume that the patient has a single resonant peak whose frequency varies with time according to the physiological and mechanical status of the respiratory airways, then the averaging process can produce the appearance of two resonant peaks. This characteristic has not been validated, however, the possibility that it may exist has been noted for investigation in future studies. If it exists, a time-varying resonant peak may have important consequences where automatically controlled ventilation is being considered.

The results from the clinical studies are too few to be of major significance, however, they provide a preliminary insight into the dynamic behaviour of the respiratory system of patients undergoing HFJV. A more substantial clinical study is required to provide conclusive confirmation of these results.

SIMULATION

During the specification and design phase of the instrumentation described in Chapter 5, consideration was given to the long-term aim of having a computer-controlled high-frequency jet ventilator which optimises patient alveolar ventilation according to some objective function. With this in mind, Chapter 7 presents an investigation of the physiological factors affecting gas exchange during artificial ventilation and specifies criteria for selecting control variables for implementing automatically controlled ventilation. Several control variables were considered for this task and their relative merits summarised. The most suitable control variables were the respiratory ventilation frequency, patient chest-wall displacement, jet-drive pressure and patient alveolar pressure.

In Chapter 7 the concept of using a Lyapunov-based model reference adaptive control (MRAC) scheme for implementing automatic control of high-frequency ventilation was introduced. The MRAC is designed so that the output of the system being controlled (the patient-model) eventually follows the output of a prespecified model (the reference-model) that exhibits desirable characteristics. The Lyapunov redesign method guarantees the asymptotic stability of the overall closed-loop system; so that the patient-model eventually tracks the reference-model with zero error. This type of control scheme appears well-suited to this particular application area. The adaptation process is initiated as a result of a mismatch in parameters between patient- and reference-models. When adaptation is complete, the properties of the patient-model should match the properties of the desired reference-model within the limits of the model structure.

Based on the results of the clinical studies, a simple patient-model was designed around a second-order lumped-parameter RLC circuit to represent the acoustic properties of a real patient. A second order MRAC scheme was then designed to investigate the performance of the closed-loop system in a simulation environment. The objective of the study was to simulate the condition of a time-varying resonant peak - as hypothesised from the clinical data - and study the stability and convergence of the adaptive ventilator under varying conditions. Once some preliminary decisions have been made about the initial value of certain constants, the performance of the MRAC system demonstrated rapid adaptation and convergence of the patient-model to match the properties of the reference-model for a step change in patient-model parameters.

The MRAC method was chosen since it has the advantage that an implicit proof of closed-loop stability exists. In fact, the adaptive laws are chosen specifically to guarantee that the patient will always follow the model assuming enough frequency richness in the input

signal. Also, the MRAC algorithm is robust in the sense that it does not require a different design for different reference inputs. However, the stability proof is not guaranteed if the model of the system does not match the actual system.

The practical realisation of an MRAC scheme for automatic optimal control of patient alveolar ventilation is feasible with the existing instrumentation. The first stage of this scheme would require a method for determining the respiratory dynamics of a patient so that the reference-model parameters can be initialised. The instrumentation described in Chapter 5 is well-suited to performing this type of identification. The real-time computational power of the instrument could then be exploited to implement a numerical technique for computing the MRAC system equations.

The clinical use of an MRAC-based ventilator would require certain safe-guards against excessive build-up of pressure in the patient airways. For instance, where the patient dynamics diverge significantly from the reference-model it is possible that the adaptive transients generate input pressures to the patient which may have harmful side-effects. In this case, it may be necessary to re-evaluate the reference-model such that it is more representative of the current physiological state of the patient thereby reducing the adaptive transients to a safer level. Thus, the MRAC scheme can be used to stabilise patient ventilation for small fluctuations about an equilibrium state. As the patient dynamics diverge significantly from this equilibrium, the instrument can be programmed to either re-adjust the reference-model and continue ventilation or set off an alarm to alert clinical staff of the danger.

FUTURE WORK

During clinical studies no attempt was made to correlate the observed airways resonances with alveolar ventilation. This was mainly because of the limitations imposed by existing transducers for continuous measurement of alveolar ventilation. It is hoped that a future study will perform these measurements to determine whether alveolar ventilation is enhanced as result of ventilating a patient at the resonant frequency of his airways. If a strong correlation can be established between acoustic resonance and alveolar ventilation, then it follows that ventilation can be non-invasively measured from chest-wall oscillations. This method of monitoring offers significant advantages over traditional, invasive techniques that directly measure average blood gas pressures. Also, the chest-wall dynamics are such that the delay associated with blood gas stabilisation is not incurred.

If we assume, for now, that optimal alveolar ventilation can be achieved by ventilating a patient at a frequency corresponding to the resonant frequency of his airways, then an adaptive ventilator could be specified whose objective would be to ensure that the ventilation frequency tracks the patients' resonant frequency with time.

The MRAC ventilator described in Chapter 7 is based on the assumption that the patient alveolar pressure can be monitored. Although this assumption is impractical at present, it is proposed that chest-wall oscillations may be used as an alternative measure of alveolar ventilation and hence as a control variable of the MRAC system.

The type of processing considered throughout this study has been centred around numerical and procedural methods. A more powerful approach is based around the use of artificial intelligence (AI) techniques. Two particular sub-branches of AI - *expert systems* ES and *knowledge-based systems* - are particularly suitable for application in some areas of this study. For example, whilst the identification of respiratory dynamics is best performed using a procedural numeric-intensive algorithm, the task of providing practical guidelines to clinicians on *when* and *how* to apply HFJV and *what* initial parameters are suited for a particular patient are best solved using the expert systems approach. The value of the ES approach lies in its ability to perform numerical and symbolic processing. In cases where numerical solutions are not applicable, the ES can draw upon *heuristics* to provide a valid solution.

The conceptual hardware and software framework for implementing such expert systems and support environments can be found in a recent paper (Appendix 4). This paper outlines current research and development work (partly derived as a direct result of the research work presented in this thesis) to implement around an IBM-PC based computer a blackboard expert system model to enable real-time distributed knowledge-based signal processing and control.

Appendix 1

Published Paper on the Signal Processing Computer

Hailstone, J.G., Jones, N.B., Parekh, A., Sehmi, A.S., Watson, J.D. and Kabay, S. (1986): "Smart instrument for flexible digital signal processing", *Med. & Biol. Eng. & Comput.*, **24**, 301-304.

Technical note

Smart instrument for flexible digital signal processing

J. G. Hailstone N. B. Jones A. Parekh
A. S. Sehmi J. D. Watson S. Kabay

Graduate Division of Biomedical Engineering, University of Sussex Centre for Medical Research, Falmer, Brighton, E. Sussex BN1 9QT, UK

Keywords—*Interactive signal processing, EMG analysis, Vascular impedance measurement*

Med. & Biol. Eng. & Comput., 1986, **24**, 301–304

ntroduction

ignificant problem with most digital signal processing ems so far introduced is the rigidity of the user inter- e format adopted. As a result of this an inordinate ount of time is required to consult the system. This ible signal processing instrument has been designed for either by nonexperts or in situations where the user is ble to pay much attention to the instrument (for mple in an operating theatre). The signal processing puter (SPC) is a friendly menu-driven system present- the user with a page of options on the screen, allowing y of analysis parameters, patient data etc. in a conve- t manner through depression of the appropriate soft . The relabelling of these buttons and the use of ching menus allows the user to progress through a l acquisition, processing and display of results with prior knowledge that all options have been considered iously in the laboratory during the program writing e. Furthermore, the instrument exists as a single trans- able unit, containing the display with its soft keys, a disk drive and various data acquisition and pro- ing cards. The small number of buttons used to control instrument dispenses with the need for an alphanu- ic keyboard, although this can be connected if ired.

nstrument requirement and specification

he first field of instrument application was chosen to medical signal acquisition and processing, particularly lood flow and pressure during surgery. In this applica- derived from BUTLER *et al.* (1980) and LAW *et al.* 3), the surgeon could be presented with flow, pressure ECG displays, or vascular impedance modulus or phase acteristics. An application area considered later is for re use in an electromography outpatient clinic. This ication is illustrated by Fig. 5 and is derived from LAGO and JONES (1983). Consequently, the initial requirement placed upon the instrument was a capability to sample up to three channels of signal input (namely flow, pressure and ECG) at sampling frequencies of up to 2 kHz. To provide for the electromyographic application area and many other medical, biological and industrial applications, the present system facilitates acquisition of up to four multiplexed channels (8 or 12 bits) at a nominal sampling frequency of 3·3 kHz, the maximum for a single channel being 10 kHz. Apart from ease of use and safety in a clinical environment, the instrument should be portable, inexpensive and produce a visual display of computed results rather than numeric tabulation.

These requirements were derived from the current and proposed research work at the universities of Sussex and Leicester. Bearing in mind that other uses for the machine are envisaged besides that already mentioned, the instrument is amenable to expansion using commercial boards configured to the standard S-100 bus. It was also specified that programming of the machine could be done locally or remotely in a high-level language such as Fortran or in assembler via any CP/M compatible development facility. The full instrument specification is listed in the Appendix.

3 Hardware organisation

The specifications outlined have been translated into the hardware shown in Fig. 1, which portrays the front panel

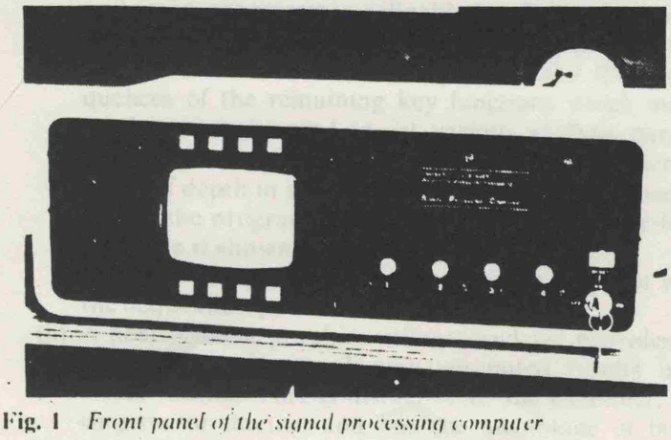


Fig. 1 *Front panel of the signal processing computer*

received 26th March and in final form 15th July 1985.

J. Hailstone is now with West Glamorgan Health Authority, N. B. es, A. S. Sehmi and S. Kabay are now with the University of ester, and J. D. Watson is now with Eurotherm International.

espondence should be addressed to Prof N. B. Jones, Depart- t of Engineering, The University, Leicester LE1 7RH, UK.

MBE: 1986

of the instrument. Its principal features are four input channels, for example, flow, pressure, ECG or other data, the floppy disk drive (FDC) for transferring programs and/or data to and from the remote research computers in the laboratory, a monitor and eight button keys. Fig. 2 is a schematic diagram of the system as a whole.

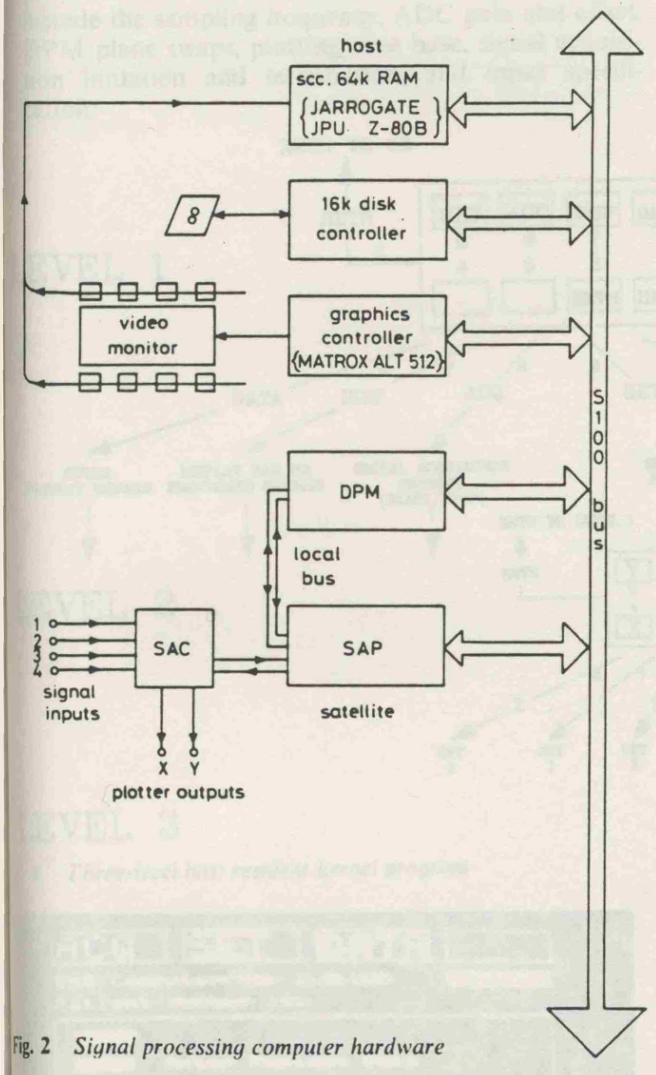


Fig. 2 *Signal processing computer hardware*

The host processor is a standard single-card computer incorporating a 6 MHz Z80B microprocessor, 64 kbytes of RAM and three each of serial and parallel ports and timers. A Matrox ALT-512 graphics board provides 256 × 256 resolution on a 5½ in display monitor. An RS232 connector allows a VDU and terminal to be connected if required for local program developments and debug facilities on the machine itself. It is not proposed that this should be used in clinical situations. The 16 kbyte FDC controls disk I/O. The front end or slave processor board is made up of 6 MHz Z80B microprocessor, 24 kbytes of RAM and 8 kbytes of PROM three parallel ports and three timers. This card controls signal acquisition of up to four channels of input data and *xy* plotter outputs for hard copy of results. Signal acquisition is performed using a software programmable analogue/digital convertor (ADC) with automatic offset and gain facility. This dual processor arrangement allows for signal acquisition on the front end and signal processing on the host. To facilitate the direct transfer of acquired/processed data to/from the host from/to the slave processor, the dual plane memory (DPM) is used. This feature provides data throughput at much faster rates than are obtainable through direct standard data bus utilisation techniques such as I/O and direct memory access (DMA). The dual plane memory is mapped into both the host's and the front-end processor's memory spaces, occupying a 16 kbyte block in each. This arrangement allows data being acquired by the slave to be loaded into its half of the DPM through its local bus while the host concurrently performs data manipulation on its half of the DPM across the S-100 bus. Thus parallel processing can be achieved without bus contention problems. 'Phantoming' is used to switch out the pre-existed memory overlaid in the host computer. The dual-plane memory provides 32 kbytes of store in two 16 kbyte blocks which may be swapped between the memory maps of the two processors by means of one I/O instruction from the host. The transition time is approximately 1·7 s, thus enabling acquisition and processing of data requiring more than a 16 kbyte buffer space to be implemented in real time. DMA techniques having the same memory bandwidth would take typically 7–8 ms. Fig. 3 illustrates the idea presented above.

Programs to be run on the slave processor can be downloaded via disk to the host DPM memory space and then transferred to the slave by issuing a memory plane swap under program control.

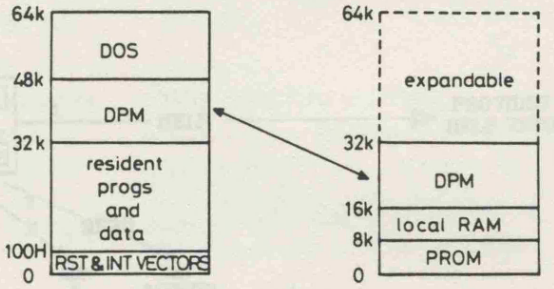


Fig. 3 *Dual plane memory*

4 Software organisation

Today instrumentation in almost every field is becoming increasingly smart or intelligent. This is primarily because the incorporation of a microprocessor, and its associated software, allows more complex configuration and analysis to be performed than in an instrument designed conventionally. There is little doubt that, through the use of suitable software and the associated interface hardware, instrument ergonomics can be high in terms of user friendliness.

The software driving the SPC can be divided into three major modules:

(i) Host resident kernel program: this suite of macros and dedicated subroutines provides the nucleus of user-configurable programs to run on the SPC for the purpose desired. The kernel provides any number of pages in a menu. With each page is a reserved RETN key function used to access the previous menu page and seven programmer definable key functions, one of which provides up to three pages of HELP information. These are used to describe in detail the consequences of the remaining key functions which might be, for example, used to set various analysis parameters. This branching structure allows for numerous levels of depth in the tree just by specifying the macros during the program writing stage. A simple three-level structure is shown in Fig. 4.

The kernel program also handles the protocol with the acquisition processor.

(ii) A host resident graphics utilities package provides an adequate facility to display computed results in a visual format. This is attractive to the examiner, e.g. surgeon or doctor, and has the advantage of being

able to convey trends or patterns in computed results more readily than a list of figures.

The monitoring program existing in the front end processor provides protocol handling with the host through which various functions may be set or slave processor status determined. The settable functions include the sampling frequency, ADC gain and offset, DPM plane swaps, plotting time base, signal acquisition initiation and termination, and input specification.

By combining these modules as an integrated whole during the program writing stage, most medical and biological signal acquisition and analysis packages can be assembled with ease. This will provide a complete menu driven system that is adequately user friendly and flexible. Programs used to provide key functions in the kernel can be written in a high level language for double precision number crunching in FFTs and matrix operations. Where speed is important, assembler programs can be used.

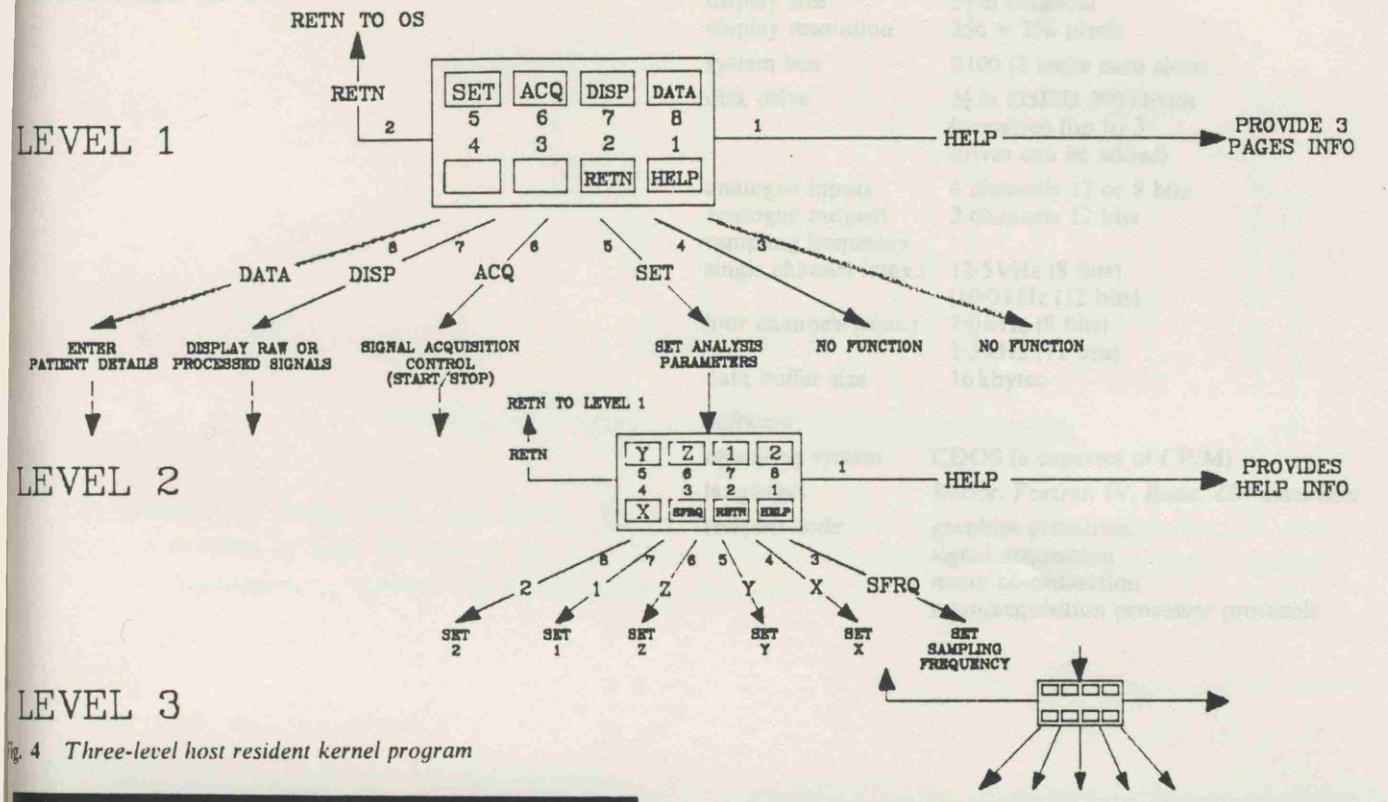
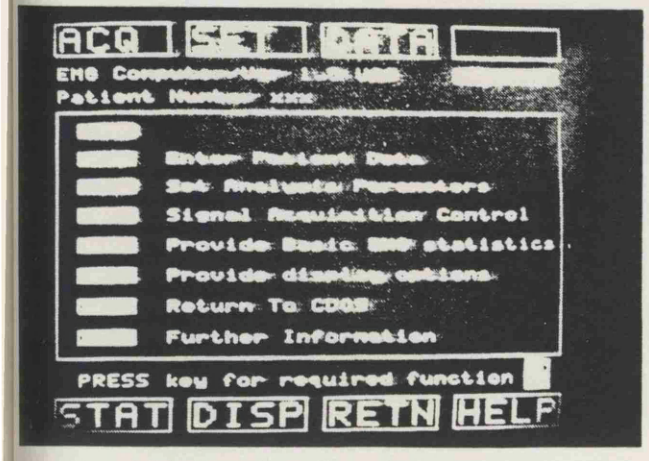
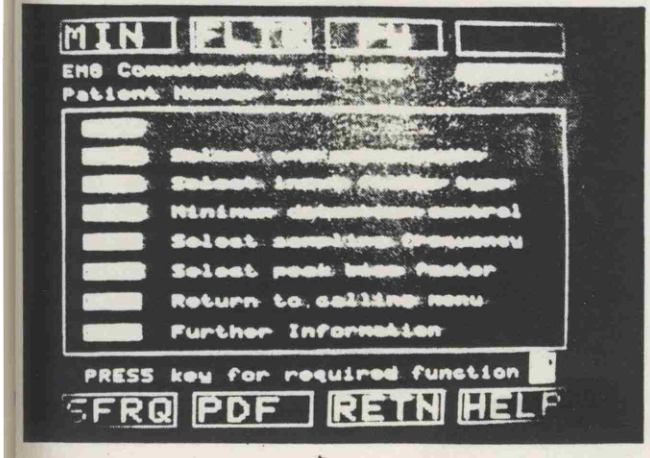


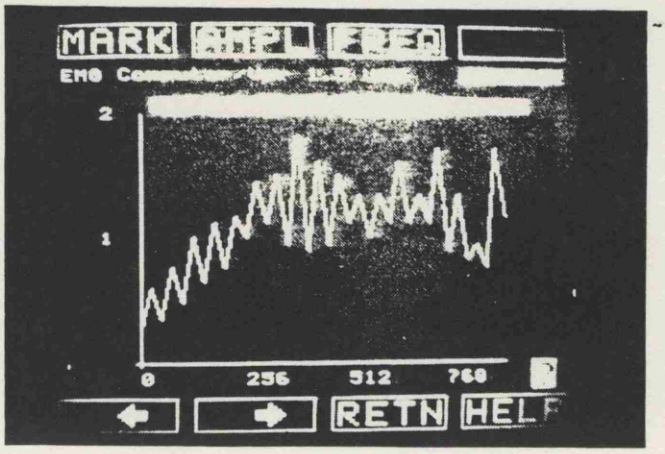
Fig. 4 *Three-level host resident kernel program*



a



b



c

Fig. 5 *Examples of video display formats*

Example of the video display formats that can be achieved are shown in Figs. 5*a*, *b* and *c*.

References

UTLER, P. A., JONES, N. B., STRACHAN, C. and SOMERVILLE, P. (1980) 'Per-operative measurement of hydraulic vascular impedance'. Final Report, South East Thames Regional Health Authority 77/4 June 1980.

AGO, P. J. A. and JONES, N. B. (1983) 'Turning points spectral analysis of the interference myoelectric activity'. *Med. & Biol. Eng. & Comput.*, **21**, 333–342.

AW, Y. F., GRAHAM, J. C., COTTON, L. T. and ROBERTS, V. C. (1983) 'Per-operative haemodynamic assessment of lower limb arterial surgery—Part 1: hydraulic impedance measurement'. *J. Biomed. Eng.*, **5**, 185–193.

Appendix

University of Sussex signal processing computer specification

Hardware:

	main processor	*acquisition processor*
processor	Z80B	Z80B
memory	64 kbytes	32 kbytes
peripheral interfaces	3 × serial 3 × 8 bit parallel	3 × 8 bit parallel
timers	3 × 16 bit	3 × 16 bit
display size	$5\frac{1}{2}$ in diagonal	
display resolution	256 × 256 pixels	
system bus	S100 (2 spare card slots)	
disk drive	$5\frac{1}{4}$ in DSDD 390 kbytes formatted (up to 3 drives can be added)	
analogue inputs	4 channels 12 or 8 bits	
analogue outputs	2 channels 12 bits	
sampling frequency		
single channel (max.)	12·5 kHz (8 bits) (10·0 kHz (12 bits)	
four channels (max.)	7·0 kHz (8 bits) 3·3 kHz (12 bits)	
data buffer size	16 kbytes	

Software:

operating system	CDOS (a superset of CP/M)
languages	Ratfor, Fortran IV, Basic, Z80 assembler
resident code	graphics primitives signal acquisition menu co-ordination main/acquisition processor protocols

Appendix 2

Published Paper on HFJV Instrumentation

Kabay, S., Jones, N.B. and Smith, G. (1989): "A system for real-time measurement and control in high frequency jet ventilation", In: *IFAC-BME, Decision support for patient management: measurement, modelling and control*, 151-160.

A System for Real-Time Measurement and Control In High Frequency Jet Ventilation.

S Kabay, N.B. Jones and G. Smith*

Department of Engineering, The University, Leicester LE1 7RH, UK.

*Department of Anaesthesia, Leicester Royal Infirmary, Leicester LE1 5WW, UK.

Abstract

This paper presents computer-aided instrumentation using the latest technology hardware for real-time measurement and modelling of high frequency jet ventilation in the area of Critical Care Medicine. The system has been developed initially to test the hypothesis that acoustic resonance of the respiratory airways represents an optimal state for alveolar gas exchange. Real-time signal processing algorithms have been implemented around a user-friendly shell to aid in the identification of transfer function relationships between respiratory data. This analysis is intended as a first step towards the development of a self-tuning algorithm for automatic control of high frequency jet ventilation.

Key Words: Real-Time Signal Processing, Respiratory Dynamics Testing, Optimal Ventilation.

Introduction

High Frequency Jet Ventilation (HFJV) is a recognised form of mechanical ventilatory support that is used in both anaesthesia and critical care medicine. The technique differs from conventional modes of ventilatory support in both its relative tidal volume and respiratory rate. Several studies have shown that HFJV is capable of maintaining adequate gas exchange in cases where conventional methods have either failed or proved to be impractical. The main advantages of HFJV include lower peak and mean airway pressures (Heijman et al., 1972), a reduction in pulmonary barotrauma (Keszler et al., 1982) and less disturbance to cardiovascular function (Sjostrand, 1980). A more detailed consideration of the subject may be found in several recent reviews (Smith, 1982; Drazen et al., 1984; Kolton, 1984).

General acceptance of HFJV has been inhibited by lack of understanding of the mechanism by which HFJV maintains gas exchange, and also by a lack of practical guidelines for

clinicians on when to apply HFJV and what ventilator parameters to set for optimum gas exchange.

One approach to the better understanding of gas exchange during HFJV treats the patient's respiratory system as an acoustic resonator whose characteristics vary markedly over the range of HFJV frequencies (Lin and Smith, 1987). Preliminary results based on an animal model have supported the hypothesis that the respiratory system behaves as an acoustic resonator (Smith and Lin, 1989).

This paper presents a computer aided instrumentation system for real-time measurement of respiratoy data using a modified high frequency jet ventilator. The system will eventually be used to provide automatic closed-loop control and management of ventilator parameters to optimise gas exchange in patients receiving critical care medicine.

The instrumentation is designed to provide real-time analysis of multi-channel respiratory data within the clinical environment. Respiratory dynamics may be examined with extremely high resolution in a fraction of the time taken by previous workers with only minimal changes to existing jet ventilation procedures.

The instrument will initially undergo clinical trials at the intensive therapy unit (ITU) of the Leicester Royal Infirmary. The aim is to test and validate the hypothesis that acoustic resonances exist within the respiratory system and that optimal gas exchange will be attained when the ventilator setting is close to this resonant frequency.

Experimental Investigations

Current clinical pratice requires trained personnel to alter and monitor ventilator settings. Present day ventilators are purpose-designed devices which have relevant monitoring, are fail-safe and have alarm systems.

A Penlon Bear-Jet ventilator used in this study is a commercially available device based on flow interruption using a jet drive valve. Gas from a high pressure source is delivered into the patient's respiratory tract via a jet cannula. The flow is controlled by changing the inspiratory-to-expiratory (I:E) ratio of the solenoid drive signal. The original system is described elsewhere (Smith, 1985).

This system provides the patient with binary pressure and flow pulses. Such stimuli are unsuitable for measuring respiratory dynamics because of damping in the system. These signals can, at best, be approximated to sine waves, and swept sine-wave analysis performed to determine the system's response. This method is extremely tedious, time-consuming, and particularly cumbersome in the clinical environment. Such analysis cannot be applied to a system which adapts to the applied signal or shows fatigue. Also, the amplitude of each "sinusoid" must be adjusted so that it remains within the linear range of the system at a particular frequency. Respiratory dynamics testing using traditional methods have already been used in animal studies by Smith and Lin (1989). In this case, however, the output variables measured also included blood gases and thus required long settling times before the system was in steady-state.

The present study initially involves measurement of abdominal and thoracic wall displacements as output variables, in order to test the hypothesis that the respiratory system behaves as an acoustic resonator over the range of frequencies attainable by HFJV. These variables have relatively short time-constants and will not require the delay associated with blood gas stabilisation.

An alternative approach is to replace the solenoid valve with a proportional controller valve and driver electronics, such that randomly varying stimuli, in addition to the existing jet drive signal, can be applied. Band-limited white-noise inputs are more efficient than sinusoidal inputs since it has a flat power spectrum over the frequency range of interest and is equivalent to applying a range of sinusoids simultaneously, see Fig.[1]. This technique inherently provides a method for rapidly determining respiratory dynamics over an expanded frequency range with a frequency resolution greater than is practicable using conventional techniques.

Instrument Specification

The method of analysing generated data involves sampling of the stimuli and response signals and performing spectral analysis using the Fast Fourier Transform (FFT). From these spectra the frequency response function and measures of the linearity of the system can be determined (Bendat and Piersol, 1971).

The sampling interval determines the maximum frequency F_{max} which can be analysed correctly before aliasing becomes a significant problem and is defined according to the Nyquist sampling theorem. In practice, analogue low-pass filters with sharp cut-off close to F_{max} are used to minimise the undesirable energy content of signals prior to digitisation.

A maximum of four channels of data is logged including upper airways pressure and flow signals, abdominal and thoracic wall displacements. At least two sets of transfer function calculations are required.

The maximum bandwidth of HFJV measurements varies between 30-300 breaths per minute (BPM) or 0.5-5 Hz. Assuming that all significant spectral energy is contained within 10 times the fundamental frequency, this sets the maximum bandwidth per channel to be 50 Hz. Thus, to satisfy the Nyquist sampling criterion, the data acquisition module should be capable of digitising analogue data at a minimum rate of 400 Hz.

Several analytical tools are required in order to assess the relationships between stimuli-response curves. These include the ability to compute and display power spectral density functions, transfer function gain and phase spectra and coherency spectra from the raw time-domain signal in real-time.

The instrumentation is finally intended for use by nonexperts. Hence the man-machine interface should be an environment which presents the functional power of the system in a clear and concise manner. A friendly menu-driven system based on the type of system architecture described by Hailstone et al. (1986) is desirable.

Hardware

The hardware implementation adopted is that of a host IBM-AT compatible microcomputer and a front-end, real-time digital signal processor (DSP). This dual processor configuration was selected since it separated the overall task into two functional sub-units and then allocated the work amongst the two processors according to their relative strengths.

The host machine is an IBM PC-AT compatible microcomputer, based on the the iAPX80286 CPU running at 12.5 MHz, a 80287-8 coprocessor is mounted to increase the speed of floating point computations. The system graphics uses the new IBM colour VGA standard 640x480 pixels with 256 KBytes of video RAM. The system contains 640 KByte of RAM expandable to 4.6 MBytes, a 40 MByte (28 ms access) fixed disk, a $5^1/_4$" and $3^1/_2$" floppy disk drive and a 40 MByte streaming tape backup unit.

The Loughborough Sound Images (LSI) coprocessor board is a high speed DSP card which occupies a single AT-slot on the host motherboard. The board comprises a TMS320C25 running at 40 MHz with 64-KWords of combined program and daand data memory (zero-wait states). The dual ported memory of the DSP can be read by the host at any time, and even modified whilst the DSP is working on other tasks. The on-board timer can be used to generate vectored-interrupts allowing complex applications to be developed which make full use of the DSP bandwidth.

An analogue interface card, also from LSI, comprising 4-input and 2-output channels connects directly to the DSP via a 50-way ribbon connector. Each input channel contains an anti-allasing filter and sample-and-hold amplifier. This system permits the simultaneous sampling of 4 channels of data which are multiplexed out to a single 3 µs conversion 12-bit analogue-to-digital converter. This system permits 4-channel data acquisition of up to 60 K samples per second. Two 12-bit digital-to-analogue converters with a typical settling time of 3 µs are also available for transmitting analogue signals to external devices.

Software

The host is responsible for supervision of the user-interface and redirection of data flow depending on the requests of the user. This complex series of tasks is coordinated through an event-handler, which interrupts the host from its current background activity in response to user requests. In some cases, where data transfer between processors is taking place, a user request may have to be put on the event queue until the current activity has been completed. All executable code and user-selected data acquisition parameters are down-loaded to the DSP board via the host. The transmission of data from the DSP to the host is initiated by a vectored interrupt call to the host processor.

The spectral analysis of data, based on the FFT, is usually the first operation to be performed and is often a preliminary to further processing. However, the FFT is a very computation intensive operation and is the major bottleneck hindering the widespread

development of real-time spectrum analysers. The DSP offers several features such as bit-reversed memory addressing to provide hardware level decimation of input/output FFT data and specialised instructions to perform single cycle mutilply-and-accumulate operations. Such provisions make it ideal for computation intensive signal processing applications.

The DSP handles all data acquisition, storing sampled data in a circular buffer. A record length of 1024 data points is analysed per channel with a 75% window overlap. Hence the DSP must have performed all processing on the current set of data and transferred the results to the host before the next frame of 256 samples per channel is updated and transferred to the main calculation buffer. With this configuartion, data acquisition memory is optimised to be 256 samples per channel. The overlapped FFT scheme is used to increase the accuracy of the analysis (Rabiner and Gold, 1975).

The spectral data transferred to the host inciudes the 4-channels of raw data and power spectra and two sets of transfer function gain, phase and coherency spectra. These are all performed in real-time, up to a spectral bandwidth of 50 Hz. The bandwidth is user selectable over the range 1-50 Hz, providing spectral resolutions of the order 0.001-0.050 Hz. This data is then displayed to the user, in real-time.

Man-Machine Interface

The man-machine interface is probably one of the most significant factors determining whether a given instrument will be accepted as a valuable tool for everyday use in the clinical environment. This phase of instrument design requires considerable planning and forethought. A brief description of the menu system is given below.

Fig.[1] also shows the menu interface presented to the user. The display is organised as a central viewport on which all signals are plotted. Either side of this canvas are two panels containing labelled buttons describing the available functions. The panels are functionally different. The panel to the right of the viewport is used to activate functions related to data acquisition and signal processing. Also included is a status display area presenting averaging information and system transfer function configuration.

The second panel activates functions related to system configuration (e.g. channel labelling, number of averages to be performed, etc) and archive management. Functions that require further information activate a pop-up window with a message indicating the response required from the user. Once the correct parameter has been set, the window vanishes, restoring the screen to its original state. This scheme avoids major re-definition of the screen and thereby avoids a complex appearance to the system.

Fig.[2] shows the pop-up associated with the 'Archive' button. This function is selected when data needs to be saved to disk or for recalling previously saved patient data. The upper half of the pop-up contains personal information specific to the patient. Each of these fields can be edited at any time. The lower half of the pop-up displays existing archive files. A highlighted cursor can be used to scroll through the archive directory to select a given file. At the bottom of the pop-up is a panel of buttons describing the operations which can be performed.

Conclusions

The use of HFJV as a form of respiratory support is known to have certain advantages over conventional methods of ventilation. However, a general acceptance of the technique has been inhibited by lack of understanding of the underlying fluid dynamics and a lack of practical guidelines for clinicians on when to apply HFJV and the ventilator settings that should be used for optimal gas exchange.

A computer orientated approach offers an attractive solution to these problems, as it can cope with the volume of information that needs to considered in such an application, and with the level of complexity that the HFJV technique entails.

A major difficulty encountered in the use of automatic control of HFJV is to obtain real-time response to ventilator parameters, on the basis of continuously monitored respiratory data, to maintain an adequate level of ventilation for patients receiving intensive care.

Computer-controlled instrumentation has been developed which is a significant first step towards the development of a self-tuning controller for use in HFJV. Coupled with the advances that have been made in non-invasive monitoring of the arterial blood gases and

transducers for measurement of airways pressure and abdominal/thoracic wall displacements, it is believed that such instrumentation will further advance the knowledge of respiratory physiology during HFJV.

The computer system is currently undergoing trials on patients with relatively normal respiratory systems who require automatic ventilation as part of their therapeutic regimen (e.g. severe head injury patients). It is hoped that a large population of patient data can be accumulated to provide the basis for a knowledge-based system which clinicians can consult on how best to employ HFJV in individual cases.

The instrumentation should eventually evolve into a full computer-aided decision support system which also takes into account seemingly trivial factors such as stiffness, length, diameter and shape of connector tubes which are critical to the performance of individual HFV machines.

References

- Bendat, J.S. & Piersol, A.G. (1971): "Random data: analysis and measurement procedures". Wiley, New York, USA.
- Drazen, J.M., Kamm, R.D., and Slutsky, A.S. (1984): "High frequency ventilation". Physiol. Rev., v64, p505.
- Hailstone, J.G., Jones, N.B., Parekh, A., Sehmi, A.S., Watson, J.D. and Kabay, S. (1986): "Smart instrument for flexible digital signal processing". Med. & Biol. Eng. & Comput., v24, p301-304.
- Heijman, K., Heijman, L., Jonzon, A., Sedin, G., Sjostrand, U., and Widman, B. (1972): "High frequency positive pressure ventilation during anaesthesia and routine surgery in man". Acta Anaesthesiologica Scandinavica, v16, p176-187.
- Kolton, M. (1984): "A review of high frequency oscillation". Can. Anaesth. Soc. J., v31, p416.
- Lin, E.S. and Smith, B.E. (1987): "An acoustic model of the patient undergoing ventilation". Br. J. Anaesth., v59, p256-264.
- Rabiner, L.R., and Gold, B. (1975): "Theory and application of digital signal processing". Prentice-Hall, p386-388.
- Sjostrand, U.H. (1980): "High frequency positive pressure ventilation (HFPPV)". Critical Care Medicine, v8, p345-364.
- Smith, B.E. (1985): "The Penlon Bromsgrove high frequency jet ventilator for adult and paediatric use". Anaesthesia, v40, p700-796.
- Smith, B.E. and Lin, E.S. (1989): "Resonance in the mechanical response of the respiratory system to HFJV". Acta Anaesthesiologica Scandinavica, v33, p65-69.
- Smith, R.B. (1982): "Ventilation at high frequencies". Anaesthesia, v37, p1011.

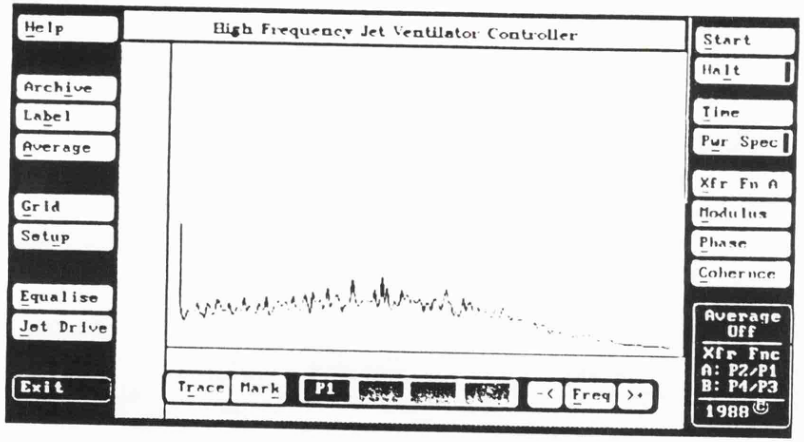


Fig.[1] Power spectrum of bandlimited white noise as seen on the system viewport.

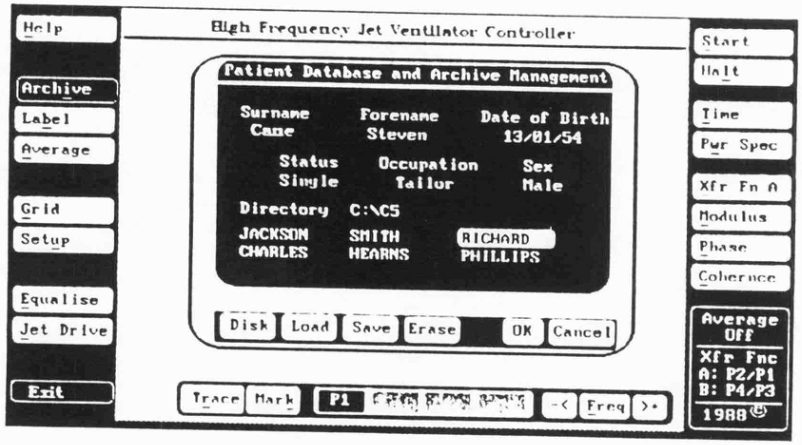


Fig.[2] Pop-up window of Archive menu.

P. J. A. Lago

Grupo de Matemática Aplicada, Faculdade de Ciências da Universidade do Porto
Rua das Taipas, 135, 4000 Porto-Portugal.
fax 351-2-698736, telex 28109 FCUP-P

Abstract

One of the applications of pharmacokinetic and pharmacodynamic models is the design of drug dosing regimens. In its simplest form only average pharmacokinetic and pharmacodynamic parameters are used. The incorporation of additional information regarding the probability distribution of the model parameters leads to alternative strategies based in some form of stochastic control. Although appealing, a major difficulty of this type of methods results from its complexity, heavy computational requirements ant the need for *a priori* information on the probability distribution of the pharmacokinetic parameters, which together strongly restrain its applicability. In this paper we propose a much simpler approach to the open-loop stochastic control of pharmacokinetic systems based on a first order approximation. The application of the method is illustrated with pharmacokinetic data available from the literature.

1 Introduction

Assuming linear and time-invariant pharmacokinetics the description of the concentration, $c(t)$, resulting from the administration of a drug, $u(t)$, can be done in terms of the input-output description,

$$c(t) = \int_0^t g(t-\tau)u(\tau)\,d\tau, \tag{1}$$

with

$$g(t) = \sum_{i=1}^{n} a_i \exp(-\lambda_i t), \tag{2}$$

or in terms of a compartmental model [1]. Since both descriptions are in essence equivalent we will consider only the input-output description. We note that the use of the input-output model does not introduce any loss in generality and avoids the identifiability problems associated with compartmental models [1,2]. In (1) the impulse response $g(t)$ may include the linear dynamic part of the pharmacodynamic model. Therefore $c(t)$ can either be interpreted as a plasma concentration, or as the concentration in the effect compartment of the pharmacodynamic related model.

It is well known that the parameter vector $\Theta = [a_1, \lambda_1, ..., a_n, \lambda_n]$ has large interindividual variations and, consequently, for the same administration regimen the induced individual

Appendix 3

Abstract of Paper on Second Order MRAC System Submitted to *6th Int. Conf. Biomed. Eng.*, Singapore 1990.

Kabay, S. and Jones, N.B. (1990): "A 2nd order model reference adaptive control system for artificial ventilation", In: *6th International Conference on Biomedical Engineering*, (Submitted).

A 2nd Order Model Reference Adaptive Control System for Artificial Ventilation

S. Kabay and N.B. Jones

Abstract

The prime task of artificial ventilation is to oxygenate patients incapable of performing spontaneous breathing, by transfer of oxygen from the inspired air to the blood stream via the alveoli, and at the same time remove carbon dioxide from venous blood and exhaust it out into the environment. This method of ventilation is routinely used in respiratory and intensive therapy units on patients with healthy lungs as well as in catastrophic lung disease. However, since artificial ventilation deviates considerably from the normal physiological mechanism of respiration, certain adverse effects can result when the method is used improperly. General acceptance of controlled ventilation is limited by:

- the harmful effects caused by cardiovascular depression due to interference with the venous return to the heart and of disturbances in the distribution of blood and gases through the lungs during ventilation
- an understanding of the bio-physiological basis of the effectiveness of controlled ventilation and the mechanisms underlying the harmful effects which may arise
- the diversity of ventilator machines and the optimal criteria used for setting ventilator parameters and control of their performance

Only with a greater understanding of these points can optimal use be made of the precise control parameters provided by controlled ventilation.

The instrumentation described in a previous paper† presented the design and implementation of a real-time system for characterising the respiratory dynamics of patients receiving high frequency jet ventilation. This paper introduces the next phase of the instrument development into an automatically controlled self-tuning ventilator. The most important design criterion is that the control system remain stable at all times. Also, the control algorithm must be robust enough to cope with the fluctuations in physiological parameters of the patient with time. Thus, identification of the patient dynamics, by periodically updating the patient model parameters on the basis of monitoring and analysis of state informations, becomes a prerequisite to the solution of the optimisation problem.

This paper presents a simulation study of a model reference adaptive control system for automatic management of ventilator parameters to optimise gas exchange in patients receiving anaesthesia. A second order lumped-parameter patient model is used which takes into account the parameters relevant to ventilation and at the same time can be updated from analysis based on respiratory measurements. The self-tuning ventilation scheme can automatically adjust its parameters in response to changes in the patient's respiratory state (i.e., lung and chest wall compliance, airways impedance and partial pressure of alveolar gases) whilst ensuring that the system remains stable at all times.

---

†. Kabay, S., Jones, N.B. and Smith, G. (1989): "A system for real-time measurement and control in high frequency jet ventilation", In: *IFAC-BME, Decision support for patient management: measure ment, modelling and control*, 151-160.

Appendix 4

Abstract of Paper on Knowledge-Based Systems Submitted to IEE

Sehmi, A.S., Loudon, G.L., Jones, N.B., and Kabay, S. (1990): "Knowledge-based systems in neuroelectric signal processing and interpretation", Submitted for publication in: *IEE Proc. Appl. of AI in Signal Processing.*

Knowledge-Based Systems in Neuroelectric Signal Processing and Interpretation

A.S. Sehmi, G.H. Loudon, N.B. Jones and S. Kabay
Department of Engineering, University of Leicester, Leicester, LE1 7RH, UK

(Invited paper submitted for publication in special issue of IEE proceedings (1990 -)
on "The Application of Artificial Intelligence Techniques to Signal Processing")

Abstract

This paper describes expert systems for decision support in the interpretation of brainstem auditory evoked potentials (BAEP), decomposition of interference pattern electromyograms (EMG) and analysis of sleep electroencephalographs (EEG). The former two systems are the work of the authors' and the latter system due to Jansen and Dawant (1989) is reviewed here for completeness and to help appreciate the wide application of knowledge-based signal processing techniques in neuroelectric signal analysis and interpretation. These systems are characterised by a significant amount of coupling between numerical and symbolic processing techniques. The BAEP and EMG expert systems incorporate rule-based inference mechanisms with a high degree of uncertain inference using fuzzy logic. Both these systems are coded in the C language for numerical processing and tokenisation of raw data and in Prolog for intermediate symbolic processing stages. The EEG expert system uses an object-oriented approach to capture high level stereotypes of spatio-temporal concepts in multi-channel EEG signals. These stereotypes can trigger lower-level numerical procedures in an opportunistic manner to extract contextual numerical information. This system uses a limited form of uncertain inference and is coded in the KEE knowledge engineering environmemt and in Lisp.

A conceptual hardware and software framework for implementing such expert systems and related support environments will also be described briefly. Research and development work on the feasibility of this framework is underway and is based upon modern digital and graphics signal processing devices and a blackboard expert system model to enable real-time distributed knowledge-based signal processing.

Appendix 5

Source Code Listings For HFJV System Described in Chapter 5.

VPORT4.C

```
/*=============================================================================

File:   Vport4.c
Version: 4.0
Release: 1.0
Authors: Salih Kabay
Date: 16 April 1989
Purpose: PC-Based DSP Workstation using TMS320C25
Usage:  vport4
Arg Description: n/a
Returns: " Maximum Happiness !!! "
Import List: n/a

=============================================================================

Copyright (C) Salih Kabay, The University, Department of Engineering,
Leicester LE1 7RH.  April 1989. All rights reserved.
No part of this program may be reprinted, reproduced or utilised in any
form or by any electronic, mechanical or other means, now known or
hereafter invented, including photocopying or recording, or in any
information storage or retrieval system, without permission in writing
from the author.

=============================================================================*/

#include "c:\c5\vport.h"

/*=============================================================================*/

main(argc,argv)
 int argc;
 char *argv[];
{
   unsigned int timer;
   int done,errsl,j,n,x,retsig;
   char intext[80];
   image *palmap;

   rect scrnR1,scrnR2,tR;
   rect VPanel_Wave,SelPanel_Wave;
   rect WaveLabel_Info,SpanKey_Info,Status_Info;
   rect Exit_Pop,Set_Pop,Average_Pop,Label_Pop,Archive_Pop;
   rect XfrA_Box,XfrB_Box,Average_Box;

   rect SpanInc_Key,SpanDec_Key;

   static rect Start_Key,Halt_Key,Time_Key,Spectra_Key;
   static rect Mod_Key,Phase_Key,Coh_Key,Equ_Key,Help_Key;
   static rect Label_Key,Archive_Key,Avge_Key;
   static rect Set_Key,Jet_Key,Exit_Key;

   static rect *mainKey[]=
    {&Time_Key,&Spectra_Key,&Mod_Key,&Phase_Key,&Coh_Key,&Halt_Key,&Start_Key};

   mainPort Wave_rcd;

   Time_Axis=True;
   Spectra_Axis=TranF_Axis=Coh_Axis=Delta_Axis=False;

   GrQuery(argc,argv);
   GrafIxCard=EGA640x350; /*128K EGA 16-colour*/
   InitGrafIx(-GrafIxCard);

   BackColor(Black);
   PenColor(Grey);
   ScreenRect(&scrnR1);
   SetDisplay(GrafPg0);
   FrameRect(&scrnR1);
   FillRect(&scrnR1,3);
   GetPort(&Init_Prt);

   Wave_Prt=&Wave_rcd;
   InitPort(&Wave_rcd);

   SetPort(Wave_Prt);
   SetLocal();
   SetOrigin(OrgX_Wave,OrgY_Wave);
   PortSizeXY(Lim_Wave,YLim_Wave);
   MovePenTo(XSet_Wave,YSet_Wave);
   BackColor(Blue);
   PenColor(LGrey);
   ScreenRect(&scrnR2);
   EraseRect(&scrnR2);
   FillRect(&scrnR2,4);

 /*Draw the label display area*/

   BackColor(Blue);
   PenColor(LWhite);
   SetRect(&WaveLabel_Info,5,2,455,19);
   FillRect(&WaveLabel_Info,0);
   KeyDrw(&WaveLabel_Info,menuTitle,-1);

   BackColor(Black);
   PenColor(Black);
   SetRect(&Exit_Pop,100,50,400,250);
   SetRect(&Set_Pop,80,40,420,260);
   SetRect(&Average_Pop,80,40,420,260);
   SetRect(&Average_Box,X0_AvgeBox,Y0_AvgeBox,X1_AvgeBox,Y1_AvgeBox);
   SetRect(&XfrA_Box,X0_XfrA,Y0_XfrA,X1_XfrA,Y1_XfrA);
   SetRect(&XfrB_Box,X0_XfrB,Y0_XfrB,X1_XfrB,Y1_XfrB);

   SetRect(&Label_Pop,80,40,420,270);
   SetRect(&LabelP1_Box,95,113,405,128);
   SetRect(&LabelP2_Box,95,141,405,156);
   SetRect(&LabelP3_Box,95,169,405,184);
   SetRect(&LabelP4_Box,95,197,405,212);

   SetRect(&Archive_Pop,X0_ArchPop,Y0_ArchPop,X1_ArchPop,Y1_ArchPop);
   SetRect(&patSurname_Box,X0_patSur,Y0_patSur,X1_patSur,Y1_patSur);
   SetRect(&patFore_Box,X0_patFor,Y0_patFor,X1_patFor,Y1_patFor);
   SetRect(&patSex_Box,X0_patSex,Y0_patSex,X1_patSex,Y1_patSex);
   SetRect(&patBirth_Box,X0_patBir,Y0_patBir,X1_patBir,Y1_patBir);
   SetRect(&patSta_Box,X0_patSta,Y0_patSta,X1_patSta,Y1_patSta);
   SetRect(&patJob_Box,X0_patJob,Y0_patJob,X1_patJob,Y1_patJob);
   SetRect(&patDisk_Box,X0_patDisk,Y0_patDisk,X1_patDisk,Y1_patDisk);

   SetRect(&Plot_Wave,50,25,450,275);
   SetRect(&Axes_Wave,45,20,455,280);
   SetRect(&tR,45,20,455,280);
   FillRect(&tR,1);       /* Clear the window    */
   PenMode(Invert);       /* Outline the window  */
   FrameRect(&tR);

   BackColor(Black);
   SetRect(&SelPanel_Wave,45,282,455,335);
   SetRect(&VPanel_Wave,5,20,44,335);
   FillRect(&SelPanel_Wave,0);
   FillRect(&VPanel_Wave,0);

   PenColor(Red);
   FrameRect(&VPanel_Wave);
   FrameRect(&Axes_Wave);
   TickAxes(&Axes_Wave);

   SetRect(&SpanDec_Key,346,310,373,330);
   SetRect(&SpanKey_Info,376,310,416,330);

   SetRect(&SpanInc_Key,419,310,446,330);
   SetRect(&Trace_Key,X0_TraceKey,Y0_TraceKey,X1_TraceKey,Y1_TraceKey);
   SetRect(&Mark_Key,X0_MarkKey,Y0_MarkKey,X1_MarkKey,Y1_MarkKey);

   BackColor(LBlue);
   PenColor(LWhite);
   KeyDrw(&SpanDec_Key,SpanDec_Tag,-1);
   KeyDrw(&SpanInc_Key,SpanInc_Tag,-1);
   KeyDrw(&SpanKey_Info,SpanKey_Tag,0);
   KeyDrw(&Trace_Key,Trace_Tag,1);
   KeyDrw(&Mark_Key,Mark_Tag,3);

   /* Draw in the Active Port Indicator Panel */
   for(actPort=3;actPort>=0;actPort--)
     PortSel();
   actPort=0;
   PenColor(Red);
   FrameRect(&SelPanel_Wave);

   SetRect(&Start_Key,X0_StrtKey,Y0_StrtKey,X1_StrtKey,Y1_StrtKey);
   SetRect(&Halt_Key,X0_HaltKey,Y0_HaltKey,X1_HaltKey,Y1_HaltKey);
   SetRect(&Time_Key,X0_TimeKey,Y0_TimeKey,X1_TimeKey,Y1_TimeKey);
   SetRect(&Spectra_Key,X0_SpectKey,Y0_SpectKey,X1_SpectKey,Y1_SpectKey);
   SetRect(&TranF_Key,X0_TranFKey,Y0_TranFKey,X1_TranFKey,Y1_TranFKey);
   SetRect(&Mod_Key,X0_ModKey,Y0_ModKey,X1_ModKey,Y1_ModKey);
   SetRect(&Phase_Key,X0_PhaseKey,Y0_PhaseKey,X1_PhaseKey,Y1_PhaseKey);
   SetRect(&Coh_Key,X0_CohKey,Y0_CohKey,X1_CohKey,Y1_CohKey);
   SetRect(&Equ_Key,X0_EquKey,Y0_EquKey,X1_EquKey,Y1_EquKey);
   SetRect(&Help_Key,X0_HelpKey,Y0_HelpKey,X1_HelpKey,Y1_HelpKey);
   SetRect(&Label_Key,X0_LabelKey,Y0_LabelKey,X1_LabelKey,Y1_LabelKey);
   SetRect(&Archive_Key,X0_ArchKey,Y0_ArchKey,X1_ArchKey,Y1_ArchKey);
   SetRect(&Avge_Key,X0_AvgeKey,Y0_AvgeKey,X1_AvgeKey,Y1_AvgeKey);
   SetRect(&Grid_Key,X0_GridKey,Y0_GridKey,X1_GridKey,Y1_GridKey);
   SetRect(&Set_Key,X0_SetKey,Y0_SetKey,X1_SetKey,Y1_SetKey);
   SetRect(&Jet_Key,X0_JetKey,Y0_JetKey,X1_JetKey,Y1_JetKey);
   SetRect(&Exit_Key,X0_ExitKey,Y0_ExitKey,X1_ExitKey,Y1_ExitKey);

   SetPort(Init_Prt);
   BackColor(Blue);
   PenColor(LGrey);
   SetRect(&Status_Info,X0_Status,Y0_Status,X1_Status,Y1_Status);
   FillRoundRect(&Status_Info,8,8,0);
   FrameRoundRect(&Status_Info,8,8);
   MoveTo(X0_Status+5,Y0_Status+32);
   LineTo(X1_Status-5,Y0_Status+32);
```

VPORT4.C

```
MoveTo(X0_Status+5,Y0_Status+78);
LineTo(X1_Status-5,Y0_Status+78);

BackColor(LBlue);
PenColor(LWhite);
KeyDrw(&Start_Key,Start_Tag,0);
KeyDrw(&Halt_Key,Halt_Tag,2);
KeyDrw(&Time_Key,Time_Tag,0);
KeyDrw(&Spectra_Key,Spectra_Tag,1);
KeyDrw(&TranF_Key,TranFA_Tag,0);
KeyDrw(&Mod_Key,Mod_Tag,0);
KeyDrw(&Phase_Key,Phase_Tag,0);
KeyDrw(&Coh_Key,Coh_Tag,0);
KeyDrw(&Eqe_Key,Eqe_Tag,0);
KeyDrw(&Set_Key,Set_Tag,3);
KeyDrw(&Help_Key,Help_Tag,0);
KeyDrw(&Archive_Key,Archive_Tag,4);
KeyDrw(&Label_Key,Label_Tag,2);
KeyDrw(&Avge_Key,Avge_Tag,0);
KeyDrw(&Grid_Key,Grid_Tag,0);
KeyDrw(&Jet_Key,Jet_Tag,0);
BackColor(Red);
KeyDrw(&Exit_Key,Exit_Tag,-1);
SetPort(Wave_Prt);
EventQueue(True);
AvgeInfo();

curbase=SelectBoard(BaC25);
Hold();
ReadStatReg();
LoadObjectFile("pr.rspo");

DrawDetails();
sprintf(msg2,"%3d Hz",SFreq[stl]);
MoveTo(380,300);
BackColor(Black);
PenColor(LRed);
DrawString(msg2);
timer=(0xffff-(3125000.0/(PScal[stl]*SFreq[stl])));
timer++;
PutMem16('d',TIMER,timer);
PutMem16('d',PSCAL,PScal[stl]-1);

PutMem16('d',XPIPA,portDef[0].sig);
PutMem16('d',XPOPA,portDef[1].sig);
PutMem16('d',XPIPB,portDef[2].sig);

PutMem16('d',XPOPB,portDef[3].sig);

done=False;
do {
  if((KeyEvent(False,&Evnt1)==True)) {
if((Evnt1.ASCII==0)&&(Evnt1.ScanCode==F1)) {

/* Exit */SetPort(Init_Prt);
        InvertRoundRect(&Exit_Key,8,8);
        SetPort(Wave_Prt);
        OpenWindow(&pulmage,&Exit_Pop,Exit_PopTag,YesNo,Red);
MoveTo(105,100);
DrawString(L1_Exit);
MoveTo(105,115);
DrawString(L2_Exit);
        MoveTo(105,135);
DrawString(L3_Exit);
        MoveTo(105,175);
DrawString(Exit_Ask);
        cmd=True;
        do {
          if((KeyEvent(False,&Evnt1)==True)) {
          switch(Evnt1.ASCII) {
          case 'y':
            {
            done=True;
            cmd=False;
            break;
            }
          case 'n':
            {
            cmd=False;
            break;
            }
          } /* switch */
          } /* if */
        } while(cmd==True);
        CloseWindow(pulmage,&Exit_Pop);
        SetPort(Init_Prt);
        InvertRoundRect(&Exit_Key,8,8);
        SetPort(Wave_Prt);
      }
      else {
        switch(Evnt1.ASCII) {
          kase('s',Start(&TranF_Key,manKey))

/* Help */   case 'h':

          SetPort(Init_Prt);
          InvertRoundRect(&Help_Key,8,8);
          Delay1();
          InvertRoundRect(&Help_Key,8,8);
          SetPort(Wave_Prt);
          break;

/*Archive*/   case 'l':

          keyOff=keyOn;keyOn=HLT;
          KeyStat(manKey);
          Archive(&Archive_Key,&Archive_Pop,archLabel,MAXARCHLABELS);
          SetPort(Init_Prt);
          BackColor(LBlue);
          PenColor(LWhite);
          (xfrSta==XPERA) ? KeyDrw(&TranF_Key,TranFA_Tag,0) :
                  KeyDrw(&TranF_Key,TranFB_Tag,0);
          SetPort(Wave_Prt);
          (keyOn==HLT) ? (keyOn=keyOff) : NULL;
          KeyStat(manKey);
          break;

/* Label */   kase('b',Label(&Label_Key,&Label_Pop,portLabel,MAXPORTLABELS))

/* Average */   kase('a',Average(&Average_Pop,&Avge_Key,&Average_Box))

/* Grid */   kase('g',Grid())

/* Trace */   kase('r',Trace(manKey))

/* Jet Drive */ case 'j':

          SetPort(Init_Prt);
          InvertRoundRect(&Jet_Key,8,8);
          Delay1();
          InvertRoundRect(&Jet_Key,8,8);
          SetPort(Wave_Prt);
          break;

/* Time */   case 't':

          xfrOn=False;

          keyOff=sigSta;keyOn=sigSta=TIME;
          KeyStat(manKey);
          WhichPlot();
          break;

/* Pwr Spec */ case 'w':

          xfrOn=False;
          keyOff=sigSta;keyOn=sigSta=PSD;
          KeyStat(manKey);
          WhichPlot();
          break;

/* Xfer Fn */ case 'x':

          if(xfrOn==False) {
            xfrOn=True;
            keyOff=sigSta;keyOn=sigSta=MAG;
            KeyStat(manKey);
          }
          SetPort(Init_Prt);
          BackColor(LBlue);
          PenColor(LWhite);
          if(xfrSta==XPERA) {
            xfrSta=XPERB;
            KeyDrw(&TranF_Key,TranFB_Tag,0);
          }
          else {
            xfrSta=XPERA;
            KeyDrw(&TranF_Key,TranFA_Tag,0);
          }
          InvertRoundRect(&TranF_Key,8,8);
          Delay1();
          InvertRoundRect(&TranF_Key,8,8);
          SetPort(Wave_Prt);
          WhichPlot();
          break;

/* Modulus */ case 'm':

          xfrOn=True;
          keyOff=sigSta;keyOn=sigSta=MAG;
          KeyStat(manKey);
          WhichPlot();
          break;
```

VPORT4.C

```
/* Phase */    case 'p':

            xfrOn=True;
            keyOff=algSts;keyOn=algSts=PHAS;
            KeySts(menKey);
            WhichPlot();
            break;

/* Coherence */ case 'c':

            xfrOn=True;
            keyOff=algSts;keyOn=algSts=COH;
            KeySts(menKey);
            WhichPlot();
            break;

/* Equation */  case 'e':

            SetPort(Init_Prt);
            InvertRoundRect(&Equ_Key,8,8);
            Delay1();
            InvertRoundRect(&Equ_Key,8,8);
            SetPort(Wave_Prt);
            break;

/* Setup */    Icase('s',SetUp(&Set_Pop,&Set_Key,&XfrA_Box,&XfrB_Box))

/* Freq */     case 'f':

            BackColor(Black);
            PenColor(LRed);
            InvertRoundRect(&SpanKey_Info,8,8);
            errd=True;
            do {
            if((KeyEvent(False,&Evnt1)==True)) {
             switch(Evnt1.ScanCode) {
/* Decr Freq */     case L_Arr:

              InvertRoundRect(&SpanDec_Key,8,8);
              if(sfi=!)
                sfi--;
              if(SFreq[sfi])
                 sprintf(msg2,"%3d Hz ",SFreq[sfi]);
              else
                 sprintf(msg2,"%3d KHz",SFreq[sfi]/1000);
              MoveTo(380,300);
              DrawString(msg2);
              InvertRoundRect(&SpanDec_Key,8,8);
              break;

/* Incr Freq */    case R_Arr:

              InvertRoundRect(&SpanInc_Key,8,8);
              if(sfi<4)
                 sfi++;
              if(SFreq[sfi])
                 sprintf(msg2,"%3d Hz ",SFreq[sfi]);
              else
                 sprintf(msg2,"%3d KHz",SFreq[sfi]/1000);
              MoveTo(380,300);
              DrawString(msg2);
              InvertRoundRect(&SpanInc_Key,8,8);
              break;

              default:

                if((Evnt1.ASCII ')=='f')
                  errd=False;
                break;

             } /* switch */
             } /* if */
             } while(errd==True);
             timer=(0xffff-(3125000.0/(PScal[sfi]*SFreq[sfi])));
             timer++;
             PutMem16('d',TIMER,timer);   /* Timer value */
             PutMem16('d',PSCAL,PScal[sfi]-1);
             InvertRoundRect(&SpanKey_Info,8,8);
             break;

            } /*Switch*/

            switch(Evnt1.ScanCode) {
              case R_Arr:

                Port5[actPort].status=False;
                PortSel();
                (actPort) ? actPort++ : (actPort=0);
                Port5[actPort].status=True;
                PortSel();
                WhichPlot();
                break;

              case L_Arr:

                Port5[actPort].status=False;
                PortSel();
                (actPort0) ? actPort-- : (actPort=3);
                Port5[actPort].status=True;
                PortSel();
                WhichPlot();
                break;

              case U_Arr:

                ZoomPlot();
                WhichPlot();
                break;

              case D_Arr:

                DeZoomPlot();
                WhichPlot();
                break;

             } /*switch*/
            } /*Else Evnt1.ASCII==0*/
           } /*Else KeyEvent*/
          } while(done==False);
          StopEvent();
          Delay1();
          ClearText();
          SetDisplay(TextPg0);
          i=QueryError();
          printf("QueryError=%d/%d \n\n",i > 7,i & 127);
        } /* End of Main */
```

CAPTURE.C

```
/*****************************************************************

File:   Capture.c
Version: 4.0
Release: 1.1
Author: Salih Kabey
Date:   3rd July 1989
Purpose: PC-Based DSP Workstation using TMS320C25
Usage:  vport4
Arg Description: n/a
Returns: " Maximum Happiness !!! "
Import List: n/a

*****************************************************************

Copyright (C) Salih Kabey, The University, Department of Engineering,
Leicester LE1 7RH.  April 1989. All rights reserved.
No part of this program may be reprinted, reproduced or utilised in any
form or by any electronic, mechanical or other means, now known or
hereafter invented, including photocopying or recording, or in any
information storage or retrieval system, without permission in writing
from the author.

*****************************************************************/

#include "c:\c5\vport.h"

/*****************************************************************
 * Start()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
Start(rect *Key, rect **manKey) {

  extern readit(),Init25();
  rect *R,*S;

  keyOn=STRT;keyOff=HLT;
  KeyStat(manKey);
  keyOn=algSts;keyOff=OFF;
  KeyStat(manKey);

  DrawDetails();

  mmxPlot=True;
  Halt_Flg=False;
  Init25();
  do {
    readit();
    Plot();
    if((KeyEvent(False,&Evnt1)==True)) {
      switch(Evnt1.ASCII' ') {

/* Halt */    case 'I',Halt_Flg=True)

/* Time */    case 't':

            xfrOn=False;
            keyOff=algSts; keyOn=algSts=TIME;
            KeyStat(manKey);
            DrawDetails();
            break;

/* Pwr Spec */ case 'w':

            xfrOn=False;
            keyOff=algSts; keyOn=algSts=PSD;
            KeyStat(manKey);
            DrawDetails();
            break;

/* Xfer Fn */  case 'z':

            if(xfrOn==False) {
              xfrOn=True;
              keyOff=algSts; keyOn=algSts=MAG;
              KeyStat(manKey);
            }

            SetPort(Init_Prt);
            BackColor(LBlue);
            PenColor(LWhite);
            if(xfrSts==XFERA) {
              xfrSts=XFERB;
              KeyDrw(Key,TranPB_Tag,0);
            }
            else {
              xfrSts=XFERA;
              KeyDrw(Key,TranPA_Tag,0);
            }
            InvertRoundRect(Key,8,8);
            Delay1();
            InvertRoundRect(Key,8,8);
            SetPort(Wave_Prt);
            DrawDetails();
            break;

/* Modulus */  case 'm':

            xfrOn=True;
            keyOff=algSts;keyOn=algSts=MAG;
            KeyStat(manKey);
            DrawDetails();
            break;

/* Phase */    case 'p':

            xfrOn=True;
            keyOff=algSts;keyOn=algSts=PHAS;
            KeyStat(manKey);
            DrawDetails();
            break;

/* Coherence */ case 'c':

            xfrOn=True;
            keyOff=algSts;keyOn=algSts=COH;
            KeyStat(manKey);
            DrawDetails();
            break;

        } /* switch */

        switch(Evnt1.ScanCode) {
          case R_Arr:

            Port5[actPort].status=False;
            Port5el();
            (actPort) ? actPort++ : (actPort=0);
            Port5[actPort].status=True;
            Port5el();
            break;


          case L_Arr:

            Port5[actPort].status=False;
            Port5el();
            (actPort0) ? actPort-- : (actPort=3);
            Port5[actPort].status=True;
            Port5el();
            break;

          kass(U_Arr,ZoomPlot())
          kass(D_Arr,DnZoomPlot())

        } /*switch*/
      } /* If to check Halt key */
    } while(Halt_Flg==False);

  keyOn=HLT;keyOff=STRT;
  KeyStat(manKey);
  mmxPlot=False;
  WhichPlot();
  avgeRst=True;
  Hold();

}

/*****************************************************************
 * InitAvg()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */
 void
InitAvg(void) {

  int i;

  for(i=0;i ;i++)
    *(psdbf+2*i)= *(psdbf+2*i+1)= *(gain+i)= *(phase+i)= *(coh+i)= 0.0;
  avgeRst=False;
  avgePrm=0;

} /*InitAvg*/
```

CAPTURE.C

```
/*****************************************************************
 * EvalTime()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
EvalTime(void) {

  int i=0,j;

  for(j=TIMBF;j<BA;j+=2,i++)
    *(timbf+i)=adc2ins(j);

} /*EvalTime*/

/*****************************************************************
 * EvalPSD()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
EvalPSD(void) {

  unsigned int ipA,ipB,opA,opB;
  int i,j,k,m,n;
  float Gxx,Gyy,Gxy,Hxx,Hyy,Hxy,G[4];

  ipA=portDef[0].sig; opA=portDef[1].sig;
  ipB=portDef[2].sig; opB=portDef[3].sig;

  for(j=PWRBF,J=0;j<(PWRBF+NN);j+=4,J++) {

    for(k=0;k<4;k++) {
      m=k*N2; n=k*NN;
      G[k]=(65536*pwr2flt(j+n)+pwr2flt(j+n+2))*1.0e-6;
      if(avgSts==False)
        *(psdbf+i+m)=0.0;
      *(psdbf+i+m)+=G[k];
    }

    Gxx=G[ipA]; Gyy=G[opA];
    Hxx=G[ipB]; Hyy=G[opB];

    *(mag+i)=Gxx*Gyy;
    *(mag+i+N2)=Hxx*Hyy;

  }

  for(j=BFMGA,k=BFMGB,J=0;jHA;j+=4,k+=4,J++) {

    Gxy=(65536*pwr2flt(j)+pwr2flt(j+2))*4096.0e-12;
    Hxy=(65536*pwr2flt(k)+pwr2flt(k+2))*4096.0e-12;

    if(avgSts==False)
      *(gain+i)= *(gain+i+N2)= *(coh+i)= *(coh+i+N2)= 0.0;

    *(gain+i)+=Gxy; *(gain+i+N2)+=Hxy;

    *(coh+i)+=( *(mag+i)==0) ? 0.0 : Gxy/( *(mag+i));
    *(coh+i+N2)+=( *(mag+i+N2)==0) ? 0.0 :  Hxy/( *(mag+i+N2));

    if(avgSts==True) {
      ( *(coh+i)>avgPrm) ? *(coh+i)=avgPrm : NULL;
      ( *(coh+i+N2)>avgPrm) ? *(coh+i+N2)=avgPrm : NULL;
    }
    else {
      ( *(coh+i)<.0) ? *(coh+i)=1.0 : NULL;
      ( *(coh+i+N2)<.0) ? *(coh+i+N2)=1.0 : NULL;
    }
  }

} /*EvalPSD*/

/*****************************************************************
 * EvalPhase()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
EvalPhase(void) {

  int i,j,k;

  for(j=BFPHA,k=BFPHB,J=0;jHA;j+=3,k+=3,J++) {

    (avgSts==False) ? *(phase+i)=0.0 : NULL;
    (avgSts==False) ? *(phase+i+N2)=0.0 : NULL;
    *(phase+i)+=pha2flt(j);
    *(phase+i+N2)+=pha2flt(k);
  }

} /*EvalPhase*/

/*****************************************************************
 * ZoomPlot()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
ZoomPlot(void) {

  switch(sigSts) {

    case MAG:
      if(xfrCalIdx<AXXFRSCALES-1)
        xfrCalIdx++;
      break;

    case PSD:
      if(psdCalIdx<AXPSDSCALES-1)
        psdCalIdx++;
      break;

    case TIME:
      if(timeCalIdx<AXTIMESCALES-1)
        timeCalIdx++;
      break;

  } /*switch*/
  DrawDetails();

}

/*****************************************************************
 * DeZoomPlot()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
DeZoomPlot(void) {

  switch(sigSts) {

    case MAG:
      if(xfrCalIdx>0)
        xfrCalIdx--;
      break;

    case PSD:
      if(psdCalIdx>0)
        psdCalIdx--;
      break;

    case TIME:
      if(timeCalIdx>0)
        timeCalIdx--;
      break;

  } /*switch*/
  DrawDetails();
}

/*****************************************************************
 * pwr2flt()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

float
```

CAPTURE.C

```
pwr2flt(int i) {

  return(( *(buffer+i))+256.0*( *(buffer+i+1)));
}

float
pha2flt(int i) {

  float c1;

  c1=( *(buffer+i))+256.0*( *(buffer+i+1));
  if(c1>32767)
    c1-=65536;
  return(c1/182.0388889);
}

/*****************************************************************
 * adc2int()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

int
adc2int(int i) {

  unsigned char lo,hi1,hi2;
  int adcwrd;

  lo=buffer[i];
  hi1=hi2=buffer[i+1];
  lo=(lo>>4)&0x0f;
  hi1=(hi1<<4)&0xf0;
  hi2=(hi2>>4)&0x0f;
  adcwrd=(lo|hi1)+256*(hi2);
  if(adcwrd>2048)
    adcwrd-=4096;
  return(adcwrd);

} /*adc2int*/
```

PLOTS.C

```
/*****************************************************************

File:    Plots.c
Version: 4.0
Release: 1.0
Authors: Salih Kabay
Date:    1st July 1989
Purpose: PC-Based DSP Workstation using TMS320C25
Usage:   vport4
Arg Description: n/a
Returns: " Maximum Happiness !!! "
Import List: n/a

*****************************************************************

Copyright (C) Salih Kabay, The University, Department of Engineering,
Leicester LE1 7RH.  April 1989. All rights reserved.
No part of this program may be reprinted, reproduced or utilised in any
form or by any electronic, mechanical or other means, now known or
hereafter invented, including photocopying or recording, or in any
information storage or retrieval system, without permission in writing
from the author.

*****************************************************************/

#include "c:\c5\vport.h"

/*****************************************************************
 * WhichPlot()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
WhichPlot(void) {

  switch(sigSts) {

    case(TIME,PltTime())
    case(PSD,PltPSD())
    case(MAG,PltMag())
    case(PHAS,PltPhase())
    case(COH,PltCoh())

  }
  DrawDetails();
  PlotSig();

}

/*****************************************************************
 * Plot()
 *
 * Purpose:  To plot a signal into a window.
 * Parameters: (R): defines the window area.
 * Returns:  None.
 * Action:
 */

 void
Plot(void) {

  if(avgeRst==True)
    InitAvg();

  if(avgeSts==True)
    avgePrm++;

  EvalTime();   /* Calculate Time Waveforms      */
  EvalPSD();    /* Calculate PSD, X& Pn Mag & Coherency */
  EvalPhase();  /* Calculate X& Pn Phase         */

  switch(sigSts) {

    case(TIME,PltTime())   /* Time signal    */
    case(PSD,PltPSD())     /* Power Spectra  */
    case(MAG,PltMag())     /* X& Pn Magnitude */
    case(PHAS,PltPhase())  /* X& Pn Phase    */
    case(COH,PltCoh())     /* X& Pn Coherency */

  } /*switch*/

  AvgeIndx();
  PlotSig();
  if(avgePrm==avgeTot)
    Halt_Plg=True;

} /*Plot*/

/*****************************************************************
 * PltTime()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
PltTime(void) {

  register int i;
  float c1;

  c1=(125.0*5.0)/(timsCal[timsCalIdx]*2048.0);
  for(i=0;i ;i++)
    *(plotbf+i)=ClipPlot(150-( *(timbf+i))*c1);

} /*PltTime*/

/*****************************************************************
 * PltPSD()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
PltPSD(void) {

  register int i;
  float c1;

  c1=249.0/(psdCal[psdCalIdx]);
  (avgeSts==True) ? (c1/=avgePrm) : NULL;

  for(i=0;i ;i++)
    *(plotbf+i)=ClipPlot(275.0-( *(psdbf+i))*c1);

} /*PltPSD*/

/*****************************************************************

 * PltMag()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
PltMag(void) {

  unsigned int ipA,ipB;
  register int i=0;
  float c1,c2;

  ipA=portDat[0].sig;
  ipB=portDat[2].sig;

  while(1 2) {
    c1= *(gain+i); c2= *(gain+i+N2);
    (avgeSts==True) ? (c1*=avgePrm) : NULL;
    *(mag+i)=( *(psdbf+i+ipA*N2)==0) ? 0 : sqrt(c1)( *(psdbf+i+ipA*N2));

    (avgeSts==True) ? (c2*=avgePrm) : NULL;
    *(mag+i+N2)=( *(psdbf+i+ipB*N2)==0) ? 0 : sqrt(c2)( *(psdbf+i+ipB*N2));
    i++;
  }

  i=0; c1=249.0/xfrCal[xfrCalIdx];
  while(1 2) {
    *(plotbf+i )=ClipPlot(275.0-( *(mag+i))*c1);
    *(plotbf+i+N2)=ClipPlot(275.0-( *(mag+i+N2))*c1);
    i++;
  }

} /*PltMag*/

/*****************************************************************
 * PltPhase()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */
```

PLOTS.C

```
void
PltPhase(void) {

  register int i;
  float c1;

  c1=124.0/180.0;
  (avgeStr==True) ? (c1/=avgePrm) : NULL;
  for(i=0;i ;i++)
    *(plotbf+i)=150.0-( *(phase+i))*c1;

} /*PltPhase*/

/*****************************************************************
 * PltCoh()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
PltCoh(void) {

  register int i;
  float c1=249.0,c2;

  (avgeStr==True) ? (c1/=avgePrm) : NULL;
  for(i=0;i ;i++)
    *(plotbf+i)=275.0-( *(coh+i))*c1;

} /*PltCoh*/

/*****************************************************************
 * PlotSig()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
PlotSig(void) {

  register int i,x=50;
  int sampleCnt=1;

  PenMode(Ovrlay);

  i=InitTrakBuf();

  MoveTo(x,( *(plotbf+i)));
  BackColor(Black);
  PenColor(Grn);
  EraseRect(&Plot_Wave);
  if(maxPlot)
    sampleCnt=2;   /*Decimate plot during data acquisition*/
  i+=sampleCnt;
  while(x%=) {
    LineTo(x,( *(plotbf+i)));
    x+=sampleCnt; i+=sampleCnt;
  } /* while inner */
  PenMode(Invert);

} /*PlotSig*/

/*****************************************************************
 * ClipPlot()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 float
ClipPlot(float x) {

  if(x
    x=25.0;
  if(x275.0)
    x=275.0;
  return(x);

}
```

MENUS.C

```
/*****************************************************************

File:    Menus.c
Version: 4.0
Release: 1.0
Authors: Salih Kalay
Date:    1st July 1989
Purpose: PC-Based DSP Workstation using TMS320C25
Usage:   vport4
Arg Description: n/a
Returns: " Maximum Happiness !!! "
Import List: n/a

*****************************************************************

Copyright (C) Salih Kalay, The University, Department of Engineering,
Leicester LE1 7RH.  April 1989. All rights reserved.
No part of this program may be reprinted, reproduced or utilised in any
form or by any electronic, mechanical or other means, now known or
hereafter invented, including photocopying or recording, or in any
information storage or retrieval system, without permission in writing
from the author.

*****************************************************************/

#include "c:\c5\vport.h"

/*****************************************************************
 * SetUp()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
SetUp(rect *Popup, rect *Key, rect *BoxA, rect *BoxB) {

  int errd,nxtsig,port;
  unsigned int ipA,ipB,opA,opB;
  image *pdimage;

  SetPort(Init_Prt);
  InvertRoundRect(Key,8,8);
  SetPort(Wave_Prt);

  OpenWindow(&pdimage,Popup,Set_PopTag,OKAY,Red);

  PenColor(LGrey);
  MoveTo(105,75);
  DrawString(L1_Set);
  MoveTo(105,90);
  DrawString(L2_Set);
  MoveTo(105,105);
  DrawString(L3_Set);

  BackColor(Black);
  PenColor(LWhite);

  MoveTo(BoxA->Xmin-25,BoxA->Ymin+15);
  LineTo(BoxA->Xmax+25,BoxA->Ymin+15);
  FillRect(BoxA,0);
  FrameRect(BoxA);

  MoveTo(BoxB->Xmin-25,BoxB->Ymin+15);
  LineTo(BoxB->Xmax+25,BoxB->Ymin+15);
  FillRect(BoxB,0);
  FrameRect(BoxB);

  PenColor(LGrey);
  MoveTo(BoxA->Xmin+12,BoxA->Ymin+18);
  DrawString(XfA_Tag);
  MoveTo(BoxB->Xmin+12,BoxB->Ymin+18);
  DrawString(XfB_Tag);

  ipA=portDef[0].sig; opA=portDef[1].sig;
  ipB=portDef[2].sig; opB=portDef[3].sig;

  BackColor(Blue);
  for(port=0;port<AXPORT;port++) {
    MoveTo(portDef[port].x,portDef[port].y);
    DrawString(portDef[portDef[port].sig].tag);
  }

  errd=True; port=0; nxtsig=portDef[port].sig;
  BackColor(Red);
  MoveTo(portDef[port].x,portDef[port].y);
  DrawString(portDef[nxtsig].tag);

  do {
    if((KeyEvent(False,&Evnt)==True)) {
      switch(Evnt.ScanCode) {
        case SPACE:
        {
        BackColor(Red);
        (nxtsig<AXPORT) ? (nxtsig++) : (nxtsig=0);
        MoveTo(portDef[port].x,portDef[port].y);
        DrawString(portDef[nxtsig].tag);
        portDef[port].sig=nxtsig;
        AvgeInfo();
        break;
        }
        case RETN:
        {
        BackColor(Blue);
        MoveTo(portDef[port].x,portDef[port].y);
        DrawString(portDef[nxtsig].tag);
        (port<AXPORT) ? (port++) : (port=0);
        BackColor(Red);
        nxtsig=portDef[port].sig;
        MoveTo(portDef[port].x,portDef[port].y);
        DrawString(portDef[nxtsig].tag);
        break;
        }
      } /*Switch*/
      switch(Evnt.ASCII) {
        case 'o':
        {
        errd=False;
        PutMem16('d',XFIPA,portDef[0].sig);
        PutMem16('d',XFOPA,portDef[1].sig);
        PutMem16('d',XFIPB,portDef[2].sig);
        PutMem16('d',XFOPB,portDef[3].sig);
        break;
        }
        case 'c':
        {
        errd=False;
        portDef[0].sig=ipA; portDef[1].sig=opA;
        portDef[2].sig=ipB; portDef[3].sig=opB;
        AvgeInfo();
        break;
        }
      } /*Switch*/
    } /*If*/
  } while(errd==True); /*Do While*/

  CloseWindow(pdimage,Popup);
  SetPort(Init_Prt);
  InvertRoundRect(Key,8,8);
  SetPort(Wave_Prt);

}

/*****************************************************************
 * Average()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
Average(rect *Popup, rect *Key, rect *Box) {

  int errd;
  image *pdimage;

  SetPort(Init_Prt);
  InvertRoundRect(Key,8,8);
  SetPort(Wave_Prt);

  OpenWindow(&pdimage,Popup,Avge_PopTag,OKAY,Red);

  PenColor(LGrey);
  MoveTo(95,85);
  DrawString(L1_Avge);
  MoveTo(95,100);
  DrawString(L2_Avge);
  MoveTo(95,115);
  DrawString(L3_Avge);
  MoveTo(95,140);
  DrawString(L4_Avge);
  MoveTo(95,155);
  DrawString(L5_Avge);

  BackColor(Red);
  PenColor(LWhite);
  FillRect(Box,0);
  FrameRect(Box);
```

MENUS.C

```
  errsl=True;
  avgeBak=avgeTot;
  AvgePrnt();

  do {
    if((KeyEvent(False,&Evnt1)==True)) {
    switch(Evnt1.ScanCode) {
    case SPACE:
      {
      (avgeTot|1024) ? (avgeTot*=2) : (avgeTot=1);
      AvgePrnt();
      break;
      }
    } /*Switch*/
    switch(Evnt1.ASCIII' ') {
      case 'o':
      {
      errsl=False;
      avgeSts = (avgeTot!=1) ? True : False;
      avgePrm = (avgeTot!=1) ? 1 : 0;
      break;
      }
      case 'c':
      {
      errsl=False;
      avgeTot=avgeBak;
      AvgePrnt();
      break;
      }
    } /*Switch*/
    } /*If*/
    } while(errsl==True); /*Do While*/

    CloseWindow(pslmage,Popup);

    SetPort(Init_Prt);
    InvertRoundRect(Key,8,8);
    SetPort(Wave_Prt);

}

/*-----------------------------------------------------------------------
 * AvgePrnt(void)
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
AvgePrnt(void) {

  extern int avgeTot;
  extern char msg2[];

  if(avgeTot!=1) {
     MoveTo(230,192);
     DrawString("    ");
     sprintf(msg2,"%4d",avgeTot);
     MoveTo(234,192);
     DrawString(msg2);
  }
  else {
     MoveTo(230,192);
     DrawString("    ");
     MoveTo(237,192);
     DrawString("Off ");
  }
  AvgeInfo();

} /*AvgePrnt*/

/*-----------------------------------------------------------------------
 * AvgeInfo(void)
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
AvgeInfo(void) {

  extern int avgePrm,avgeTot;

  extern char msg2[];

  SetPort(Init_Prt);
  BackColor(Blue);
  PenColor(LCyan);
  MoveTo(X0_Status+5,Y0_Status+12);
  DrawString("Averaging");
  MoveTo(X0_Status+5,Y0_Status+44);
  DrawString("Xfer Func");
  MoveTo(X0_Status+10,Y0_Status+58);
  DrawString("A:");
  MoveTo(X0_Status+10,Y0_Status+71);
  DrawString("B:");
  PenColor(LGrey);

  if(avgeTot!=1) {
    sprintf(msg2,"%4d/%-4d",avgePrm,avgeTot);
    MoveTo(X0_Status+5,Y0_Status+26);
    DrawString(msg2);
  }
  else {
    MoveTo(X0_Status+5,Y0_Status+26);

    DrawString("  Off  ");
  }

  MoveTo(X0_Status+30,Y0_Status+58);
  strcpy(msg2,portDef[portDef[1].sig].tag);
  strcat(msg2,"/");
  DrawString(strcat(msg2,portDef[portDef[0].sig].tag));
  MoveTo(X0_Status+30,Y0_Status+71);
  strcpy(msg2,portDef[portDef[3].sig].tag);
  strcat(msg2,"/");

  DrawString(strcat(msg2,portDef[portDef[2].sig].tag));
  PenColor(LWhite);
  MoveTo(X0_Status+10,Y0_Status+92);
  DrawString("1989 (C)");

  BackColor(Black);
  SetPort(Wave_Prt);

} /*AvgeInfo*/

/*-----------------------------------------------------------------------
 * Label()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
Label(rect *Key, rect *Pop, pelDTV label, int maxLabels) {

  image *pslmage;

  SetPort(Init_Prt);
  InvertRoundRect(Key,8,8);
  SetPort(Wave_Prt);

  OpenWindow(&pslmage,Pop,Label_PopTag,OKAY,Red);

  PenColor(LGrey);

  MoveTo(90,73);
  DrawString(L1_Label);
  MoveTo(90,90);
  DrawString(L2_Label);

  BackColor(Black);

  FillRoundRect(&LabelP1_Box,8,8,0);
  FillRoundRect(&LabelP2_Box,8,8,0);
  FillRoundRect(&LabelP3_Box,8,8,0);
  FillRoundRect(&LabelP4_Box,8,8,0);

  EditScreen(maxLabels,label);

  CloseWindow(pslmage,Pop);
  PenMode(Invert);

  SetPort(Init_Prt);
  InvertRoundRect(Key,8,8);
  SetPort(Wave_Prt);

} /*Label*/
```

ARCHIVE.C

```
/*****************************************************************

File:    Archive.c
Version: 4.0
Release: 1.0
Author: Salih Kabay
Date:    1st July 1989
Purpose: PC-Based DSP Workstation using TMS320C25
Usage:   vport4
Arg Description: n/a
Returns: " Maximum Happiness !!! "
Import List: n/a

*****************************************************************

Copyright (C) Salih Kabay, The University, Department of Engineering,
Leicester LE1 7RH.  April 1989. All rights reserved.
No part of this program may be reprinted, reproduced or utilised in any
form or by any electronic, mechanical or other means, now known or
hereafter invented, including photocopying or recording, or in any
information storage or retrieval system, without permission in writing
from the author.

*****************************************************************/

#include "c:\c5\vport.h"

/*****************************************************************
 * Archive()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
Archive(rect *Key, rect *Pop, psEDTV label, int maxLabels) {

  int errsl=True,fileId;
  char edith;
  image *psImage;
  sHILIGHT hiLight;

  hiLight.dirStart=hiLight.xId=hiLight.yId=0;

  SetPort(Init_Prt);
  InvertRoundRect(Key,8,8);
  SetPort(Wave_Prt);

  OpenWindow(&psImage,Pop,Archive_PopTag,ARCHIVE,Red);

  PenColor(LGrey);
  MoveTo(X0_pstSur+20,Y0_pstSur-4);
  DrawString(Surname_Tag);

  MoveTo(X0_pstFor+12,Y0_pstFor-4);
  DrawString(Forename_Tag);
  MoveTo(X0_pstBir+3,Y0_pstBir-4);
  DrawString(Birth_Tag);
  MoveTo(X0_pstSta+12,Y0_pstSta-4);
  DrawString(Status_Tag);
  MoveTo(X0_pstJob+4,Y0_pstJob-4);
  DrawString(Job_Tag);
  MoveTo(X0_pstSex+16,Y0_pstSex-4);
  DrawString(Sex_Tag);
  MoveTo(X0_pstSur+20,Y0_pstDisk-4);
  DrawString(Disk_Tag);
  PrintCurDir(X0_pstSur+110,Y0_pstDisk-4);

  BackColor(Black);

  FillRoundRect(&pstSurname_Box,8,8,0);
  FillRoundRect(&pstFore_Box,8,8,0);
  FillRoundRect(&pstSex_Box,8,8,0);
  FillRoundRect(&pstBirth_Box,8,8,0);
  FillRoundRect(&pstSta_Box,8,8,0);
  FillRoundRect(&pstJob_Box,8,8,0);
  FillRoundRect(&pstDisk_Box,8,8,0);

  hiLight.maxFile=BuildDir();
  ListDir(&hiLight);
  HiLight(&hiLight);

  edith=EditScreen(maxLabels,label);
  PenColor(LGrey);
  do {
    if(KeyEvent(False,&Evnt)==True) {
      switch(Evnt.ASCII) {
        case 'o':
        case 'c':
          {
          errsl=False;
          break;
          }

        case 'l':LoadPAT(&hiLight,label,maxLabels)
        case 's':EraseFile(&hiLight)
        case 'a':Save(&hiLight)

        case 'd':
          {
          break;
          }
      } /*switch*/

      switch(Evnt.ScanCode) {

        case D_Arr:NxtFileDown(&hiLight)
        case U_Arr:NxtFileUp(&hiLight)
        case L_Arr:NxtFileLeft(&hiLight)
        case R_Arr:NxtFileRight(&hiLight)

      } /*switch*/
    }
  } while(errsl==True);

  CloseWindow(psImage,Pop);
  PenMode(Invert);
  WhichPlot();

  SetPort(Init_Prt);
  InvertRoundRect(Key,8,8);
  SetPort(Wave_Prt);

} /*Archive*/

/*****************************************************************
 * QueryErase()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void

QueryErase(int Id) {

  int errsl=True;
  char tmpstr[40];
  FILE *errstream;

  errstream=freopen("errlog.$$$","w",stderr);
  BackColor(LBlue);
  PenColor(White);
  MoveTo(X1_pstDisk-130,Y0_pstDisk-4);
  DrawString("Confirm? (y/n):");
  do {
  if(KeyEvent(False,&Evnt)==True) {
    switch(Evnt.ASCII) {
      case 'y':
      {
      MoveTo(X1_pstDisk-130,Y0_pstDisk-4);
      DrawString("  Erased.. ");
      sprintf(tmpstr,"erase %s a.out",dirPat[Id].name);
      system(tmpstr);
      sprintf(tmpstr,"erase %s a.out",dirTXT[Id].name);
      system(tmpstr);
      BackColor(Blue);
      MoveTo(X1_pstDisk-130,Y0_pstDisk-4);
      DrawString("          ");
      errsl=False;
      break;
      }
      case 'n':
      {
      MoveTo(X1_pstDisk-130,Y0_pstDisk-4);
      DrawString("  Cancelled  ");
      Delay1();
      BackColor(Blue);
      MoveTo(X1_pstDisk-130,Y0_pstDisk-4);
      DrawString("          ");
      errsl=False;
      break;
      }
    }
  }
  } while(errsl);
  fclose(errstream);
  system("erase errlog.$$$ a.out");
  system("erase a.out");
```

ARCHIVE.C

```
  BackColor(Black);
  PenColor(LGrey);
}

/*-----------------------------------------------------------------------
 * ListDir()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
ListDir(psHILIGHT hiLight) {

  char fname[15],tmpfname[15];
  int i,j=0;

  PenColor(LGrey);
  for(i=hiLight->dirStart;i<hiLight->maxFile;i++) {
    strcpy(tmpfname,dirPat[i].name);
    *(tmpfname+strlen(tmpfname)-strlen(patXtn))='\0';
    sprintf(fname,"%-8s",tmpfname);
    MoveTo(xDir[j],yDir[j]);
    DrawString(fname);
    j++;

  }
  if(hiLight->maxFile-hiLight->dirStart) {
    while(j) {
      MoveTo(xDir[j],yDir[j]);
      DrawString("        ");
      j++;
    }
  }

}

/*-----------------------------------------------------------------------
 * HiLight()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:

 */

void
HiLight(psHILIGHT hiLight) {

  int j;
  rect R;

  j=3*(hiLight->yId)+(hiLight->xId);
  SetRect(&R,xDir[j]-3,yDir[j]-11,xDir[j]+75,yDir[j]+3);
  InvertRoundRect(&R,8,8);
}

/*-----------------------------------------------------------------------
 * BuildDir()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

int
BuildDir(void) {

  int j=0,i=0;

  _dos_findfirst(diPAT,_A_NORMAL,&pat_file);
  _dos_findfirst(diTXT,_A_NORMAL,&txt_file);
  UpdateDir(j); j++;
  while(_dos_findnext(&pat_file)==0) {
    _dos_findnext(&txt_file);
    UpdateDir(j); j++;
  }
  return(j-1);

}

/*-----------------------------------------------------------------------
 * UpdateDir()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
UpdateDir(int j) {

  strcpy(dirPat[j].name,pat_file.name);
  dirPat[j].size=pat_file.size;

  strcpy(dirTXT[j].name,txt_file.name);
  dirTXT[j].size=txt_file.size;

}

/*-----------------------------------------------------------------------
 * PrintCurDir()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
PrintCurDir(int x, int y) {

  extern char currentDir[];

  getcwd(currentDir,MAXDIRNAME);
  MoveTo(x,y);
  DrawString(currentDir);

}

/*-----------------------------------------------------------------------
 * EraseFile()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
EraseFile(psHILIGHT hiLight) {

  int field;


  field=3*hiLight->yId+hiLight->xId+hiLight->dirStart;
  QueryErase(field);
  HiLight(hiLight);
  hiLight->dirStart=hiLight->xId=hiLight->yId=0;
  hiLight->maxFile=BuildDir();
  ListDir(hiLight);
  HiLight(hiLight);

}

/*-----------------------------------------------------------------------
 * Save()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
Save(psHILIGHT hiLight) {

  int i,maxlines;
  rect archSave_Pop;
  image *psQryImg;

  SetRect(&archSave_Pop,X0_ArchSavePop,Y0_ArchSavePop,\
          X1_ArchSavePop,Y1_ArchSavePop);
  SetRect(&archSave_Box,X0_ArchSavePop+128,Y0_ArchSavePop+120,\
          X0_ArchSavePop+218,Y0_ArchSavePop+140);

  OpenWindow(&psQryImg,&archSave_Pop,archSave_PopTag,OKAY,Blue);

  PenColor(LGrey);
  MoveTo(X0_ArchSavePop+5,Y0_ArchSavePop+45);
  DrawString(L1_archSavePop);
  MoveTo(X0_ArchSavePop+5,Y0_ArchSavePop+60);
  DrawString(L2_archSavePop);
  MoveTo(X0_ArchSavePop+5,Y0_ArchSavePop+90);
  DrawString(L3_archSavePop);
  BackColor(Black);
  FillRoundRect(&archSave_Box,8,8,0);

  switch(EditScreen(maxlines-1,fileName)) {
```

ARCHIVE.C

```
    Inse(AltO,SaveToDisk(hiLight))
  } /*switch*/

  CloseWindow(psQryImg,&archSave_Pop);
  PenMode(Invert);
  HiLight(hiLight);
  hiLight->maxFile=BuildDir();
  ListDir(hiLight);
  HiLight(hiLight);

} /*Save*/

/*****************************************************************
 * SaveToDisk()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
SaveToDisk(psHILIGHT hiLight) {

  VerifyFile(hiLight);
  SaveTXT();
  SavePAT();

}

/*****************************************************************
 * VerifyFile()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
VerifyFile(psHILIGHT hiLight) {

  char fname[15];
  int i;

  for(i=0;i<hiLight->maxFile;i++) {

    strcpy(fname,dirPat[i].name);
    *(fname+strlen(fname)-strlen(patXtn))='\0';
    if(strcmp(fname,fileName[0].aline)==0) {
      BackColor(Blue);
      PenColor(LWhite);
      MoveTo(X0_ArchSavePop+50,Y0_ArchSavePop+155);
      DrawString("WARNING: File already exists!");
      getch();
    }
  }
}

/*****************************************************************
 * SavePAT()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
SavePAT(void) {

  char PATfname[15];
  int i;
  FILE *PATstream;

  strcpy(PATfname,fileName[0].aline);
  strcat(PATfname,patXtn);
  PATstream=fopen(PATfname,"w+");

  for(i=0;i<4;i++)
    fprintf(PATstream,"%d\n",archLabel[i].aline);

  for(i=0;i<4;i++)
    fprintf(PATstream,"%d %d %s \n",portDat[i].sig.Port5[i].status,
                    portLabel[i].aline);
  fprintf(PATstream,"%d %d %d %d ",all.avgaRst,avgaPrm,avgaTot);
  fprintf(PATstream,"%d %d %d ",alarmCallAx,pidCallAx,x10CallAx);
  fprintf(PATstream,"%d %d %d ",avgaSts,sigSts,x10Sts,x10On);
  fprintf(PATstream,"%d %d %d\n",x10On,lmyOn,lmyOff);
  fclose(PATstream);

}

/*****************************************************************
 * SaveTXT()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
SaveTXT(void) {

  char TXTfname[15];
  int i;
  FILE *TXTstream;

  strcpy(TXTfname,fileName[0].aline);
  strcat(TXTfname,txtXtn);
  TXTstream=fopen(TXTfname,"w+");

  MoveTo(X0_ArchSavePop+230,Y0_ArchSavePop+133);
  BackColor(Blue);
  DrawString(" 0 %");
  SaveTime(TXTstream);
  MoveTo(X0_ArchSavePop+230,Y0_ArchSavePop+133);
  DrawString(" 25 %");
  SavePSD(TXTstream);
  MoveTo(X0_ArchSavePop+230,Y0_ArchSavePop+133);
  DrawString(" 50 %");
  SaveX.0(TXTstream);
  MoveTo(X0_ArchSavePop+230,Y0_ArchSavePop+133);
  DrawString(" 62 %");
  SavePhase(TXTstream);
  MoveTo(X0_ArchSavePop+230,Y0_ArchSavePop+133);
  DrawString(" 75 %");
  SaveCoh(TXTstream);
  MoveTo(X0_ArchSavePop+230,Y0_ArchSavePop+133);
  DrawString(" 87 %");
  SaveCov(TXTstream);
  MoveTo(X0_ArchSavePop+230,Y0_ArchSavePop+133);
  DrawString("100 %");
  BackColor(Black);

  fclose(TXTstream);

}

/*****************************************************************
 * SaveTime()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
SaveTime(FILE *stream) {

  int i;

  fprintf(stream,"Time_P1\t");
  for(i=0;i<4;i++)
    fprintf(stream,"%d\t",*(timbf+i));
  fprintf(stream,"\nTime_P2\t");
  for(i=512;i<516;i++)
    fprintf(stream,"%d\t",*(timbf+i));
  fprintf(stream,"\nTime_P3\t");
  for(i=1024;i<1028;i++)
    fprintf(stream,"%d\t",*(timbf+i));
  fprintf(stream,"\nTime_P4\t");
  for(i=1536;i<1540;i++)
    fprintf(stream,"%d\t",*(timbf+i));
  fprintf(stream,"\n");

}

/*****************************************************************
 * SavePSD()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
SavePSD(FILE *stream) {

  int i;
```

ARCHIVE.C

```
  fprintf(stream,"PSD_P1\t");
  for(i=0;i<i++)
    fprintf(stream,"%g\t",*(psdbf+i));
  fprintf(stream,"\nPSD_P2\t");
  for(i=512;i<i++)
    fprintf(stream,"%g\t",*(psdbf+i));
  fprintf(stream,"\nPSD_P3\t");
  for(i=1024;i<i++)
    fprintf(stream,"%g\t",*(psdbf+i));
  fprintf(stream,"\nPSD_P4\t");
  for(i=1536;i<i++)
    fprintf(stream,"%g\t",*(psdbf+i));
  fprintf(stream,"\n");

}

/*****************************************************************
 * SaveXfr()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
SaveXfr(FILE *stream) {

  int i;

  fprintf(stream,"Mag_A\t");
  for(i=0;i<i++)
    fprintf(stream,"%g\t",*(gain+i));
  fprintf(stream,"\nMag_B\t");
  for(i=512;i<i++)
    fprintf(stream,"%g\t",*(gain+i));
  fprintf(stream,"\n");

}

/*****************************************************************
 * SavePhase()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
SavePhase(FILE *stream) {

  int i;

  fprintf(stream,"Phase_A\t");
  for(i=0;i<i++)
    fprintf(stream,"%g\t",*(phase+i));
  fprintf(stream,"\nPhase_B\t");
  for(i=512;i<i++)
    fprintf(stream,"%g\t",*(phase+i));
  fprintf(stream,"\n");

}

/*****************************************************************
 * SaveCoh()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
SaveCoh(FILE *stream) {

  int i;

  fprintf(stream,"Coh_A\t");
  for(i=0;i<i++)
    fprintf(stream,"%g\t",*(coh+i));
  fprintf(stream,"\nCoh_B\t");
  for(i=512;i<i++)
    fprintf(stream,"%g\t",*(coh+i));
  fprintf(stream,"\n");

}

/*****************************************************************
 * SaveCov()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
SaveCov(FILE *stream) {

  int i;

  fprintf(stream,"XCov_A\t");
  for(i=0;i<i++)
    fprintf(stream,"%g\t",*(mag+i));
  fprintf(stream,"\nXCov_B\t");
  for(i=512;i<i++)
    fprintf(stream,"%g\t",*(mag+i));
  fprintf(stream,"\n");

}

/*****************************************************************
 * LoadPAT()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
LoadPAT(psHILIGHT hiLight, psEDTV label, int maxLabels) {

  char ratc,lineBuf[200],PATthame[15];
  int i,len,filield;
  unsigned int timer;
  FILE *PATstream;

  filield=3**hiLight->yId+hiLight->xId+hiLight->dir3art;
  strcpy(PATthame,dirPat[filield].name);
  PATstream=fopen(PATthame,"r");

  for(i=0;i<i++) {
    fscanf(PATstream,"%s",lineBuf);
    while((ratc=getc(PATstream))!='\n') {
      *(lineBuf + (len=strlen(lineBuf)) )=ratc;
      *(lineBuf+len+1)='\0';
    }
    strcpy(archLabel[i].aline,lineBuf);
  }

  i=0;
  while(1) {
    fscanf(PATstream,"%d",&portDef[i].sig);
    fscanf(PATstream,"%d",&Port3[i].status);

    fscanf(PATstream,"%s",lineBuf);
    while((ratc=getc(PATstream))!='\n') {
      *(lineBuf + (len=strlen(lineBuf)) )=ratc;
      *(lineBuf+len+1)='\0';
    }
    strcpy(portLabel[i].aline,lineBuf);
    i++;
  }

  fscanf(PATstream,"%d%d%d%d",&xfd,&avgaRat,&avgaPrm,&avgaTot);
  fscanf(PATstream,"%d%d%d",&timsCalIdx,&psdCalIdx,&xfrCalIdx);
  fscanf(PATstream,"%d%d%d",&avgaSta,&xfgSta,&xfrSta);
  fscanf(PATstream,"%d%d%d",&xfrOn,&xayOn,&xayOff);

/* Update patient data */
  SetUpScreen(0,maxLabels,label);

/* Update Average Info */
  AvgsInfo();

/* Update TMS I/O map */
  PutMem16('d',XPIPA_portDef[0].sig);
  PutMem16('d',XPOPA_portDef[1].sig);
  PutMem16('d',XPIPB_portDef[2].sig);
  PutMem16('d',XPOPB_portDef[3].sig);

/* Update Sampling frequency */
  MoveTo(380,300);
  PenColor(LRed);
  sprintf(msg2,"%3d Hz",SFreq[sfi]);
  DrawString(msg2);
  timer=(0xffff-(3125000.0/(PScal[sfi]*SFreq[sfi])));
  timer++;
  PutMem16('d',TIMER,timer);
```

ARCHIVE.C

```
  PutMem16('d',PSCAL,PScal[st]-1);

  fclose(PATstream);
  LoadTXT(hiLight);

}

/*****************************************************************
 * LoadTXT()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
LoadTXT(psHILIGHT hiLight) {

  char lineBuf[200],TXTfname[15];
  int  field;
  FILE *TXTstream;

  field=3*hiLight->yld+hiLight->xld+hiLight->dirStart;
  strcpy(TXTfname,dirTXT[field].name);
  TXTstream=fopen(TXTfname,"r");

  PenColor(LWhite);
  BackColor(LBlue);

  MoveTo(X1_patDist-40,Y0_patDist-4);
  DrawString("Wait");

  LoadTime(TXTstream);
  LoadPSD(TXTstream);
  LoadXfr(TXTstream);
  LoadPhase(TXTstream);
  LoadCoh(TXTstream);
  LoadCov(TXTstream);

  MoveTo(X1_patDist-40,Y0_patDist-4);
  DrawString("Done");
  Delay1();
  BackColor(Blue);
  MoveTo(X1_patDist-40,Y0_patDist-4);
  DrawString("    ");
  BackColor(Black);

  fclose(TXTstream);

}

/*****************************************************************
 * LoadTime()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
LoadTime(FILE *stream) {

  char lineBuf[200];
  int portn,sample,offset=0;

  for(portn=0;portn;portn++) {
    fscanf(stream,"%s",lineBuf);
    for(sample=0;sample;sample++)
      fscanf(stream,"%d",(timbf+sample+offset));
    offset+=sample;
  }
}

/*****************************************************************
 * LoadPSD()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
LoadPSD(FILE *stream) {

  char lineBuf[200];
  int portn,sample,offset=0;

  for(portn=0;portn;portn++) {
    fscanf(stream,"%s",lineBuf);
    for(sample=0;sample;sample++)
      fscanf(stream,"%g",(psdbf+sample+offset));
    offset+=sample;
  }
}

/*****************************************************************
 * LoadXfr()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
LoadXfr(FILE *stream) {

  char lineBuf[200];
  int portn,sample,offset=0;

  for(portn=0;portn;portn++) {
    fscanf(stream,"%s",lineBuf);
    for(sample=0;sample;sample++)
      fscanf(stream,"%g",(gain+sample+offset));
    offset+=sample;
  }
}

/*****************************************************************
 * LoadPhase()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
LoadPhase(FILE *stream) {

  char lineBuf[200];
  int portn,sample,offset=0;

  for(portn=0;portn;portn++) {
    fscanf(stream,"%s",lineBuf);
    for(sample=0;sample;sample++)
      fscanf(stream,"%g",(phase+sample+offset));
    offset+=sample;
  }
}

/*****************************************************************
 * LoadCoh()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
LoadCoh(FILE *stream) {

  char lineBuf[200];
  int portn,sample,offset=0;

  for(portn=0;portn;portn++) {
    fscanf(stream,"%s",lineBuf);
    for(sample=0;sample;sample++)
      fscanf(stream,"%g",(coh+sample+offset));
    offset+=sample;
  }
}

/*****************************************************************
 * LoadCov()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
LoadCov(FILE *stream) {

  char lineBuf[200];
  int portn,sample,offset=0;
```

ARCHIVE.C

```
  for(portn=0;portn<portn++) {
    fscanf(stream,"%s",lineBuf);
    for(sample=0;sample<sample++)
      fscanf(stream,"%g",(mag+sample+offset));
    offset+=sample;
  }
}

/*************************************************************
 * NxtFileDown()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
NxtFileDown(peHILIGHT hiLight) {

  HiLight(hiLight);
  if((hiLight->yId)&&((3*(hiLight->yId+1)+hiLight->xId+hiLight->dirStart)
            <hiLight->maxFile))
    hiLight->yId+=1;
  else
    ((hiLight->maxFile-hiLight->dirStart)>=12)?(hiLight->dirStart+=3):NULL;
  ListDir(hiLight);
  HiLight(hiLight);

}

/*************************************************************
 * NxtFileUp()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
NxtFileUp(peHILIGHT hiLight) {

  HiLight(hiLight);
  if(hiLight->yId>0)
    hiLight->yId-=1;
  else
    (hiLight->dirStart>=3) ? (hiLight->dirStart-=3) : NULL;
  ListDir(hiLight);
  HiLight(hiLight);

}

/*************************************************************
 * NxtFileLeft()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
NxtFileLeft(peHILIGHT hiLight) {

  HiLight(hiLight);
  if(hiLight->xId>0)
    hiLight->xId-=1;
  ListDir(hiLight);
  HiLight(hiLight);

}

/*************************************************************
 * NxtFileRight()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
NxtFileRight(peHILIGHT hiLight) {

  HiLight(hiLight);
  if((hiLight->xId)&&((3*hiLight->yId+(hiLight->xId+1)+hiLight->dirStart)
            <hiLight->maxFile))
    hiLight->xId+=1;
  ListDir(hiLight);
  HiLight(hiLight);

}
```

EDITOR.C

```
/*****************************************************************************

File:    Editor.c
Version: 4.0
Release: 1.0
Authors: Salih Kabay
Date:    1st July 1989
Purpose: PC-Based DSP Workstation using TMS320C25
Usage:   vport4
Arg Description: n/a
Returns: " Maximum Happiness !!! "
Import List: n/a

*****************************************************************************

Copyright (C) Salih Kabay, The University, Department of Engineering,
Leicester LE1 7RH.  April 1989. All rights reserved.
No part of this program may be reprinted, reproduced or utilised in any
form or by any electronic, mechanical or other means, now known or
hereafter invented, including photocopying or recording, or in any
information storage or retrieval system, without permission in writing
from the author.

*****************************************************************************/

#include "c:\c5\vport.h"

/*****************************************************************************
 * EditScreen()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

char
EditScreen(int maxLabels, psEDTV label) {

  int cntl=True,pNum=0;

  SetUpScreen(pNum,maxLabels,label);
  PenColor(LCyan);

  do {
    if((KeyEvent(False,&Evnt1)==True)) {

      switch(Evnt1.ScanCode) {
        case RETN:
        case D_Arr:

          label[pNum].len=strlen(label[pNum].aline);
          XorTxtCursor(pNum,label);
          label[pNum].curPos=label[pNum].newPos=0;
          PenColor(LGrey);
          TxtDisplay(pNum,label);
          (pNum<maxLabels-1) ? (pNum++) : (pNum=0);
          PenColor(LCyan);
          strcpy(saveLine,label[pNum].aline);
          label[pNum].len=strlen(label[pNum].aline);
          label[pNum].curPos=label[pNum].newPos=0;
          XorTxtCursor(pNum,label);
          TxtDisplay(pNum,label);
          break;

        case U_Arr:

          label[pNum].len=strlen(label[pNum].aline);
          XorTxtCursor(pNum,label);
          label[pNum].curPos=label[pNum].newPos=0;
          PenColor(LGrey);
          TxtDisplay(pNum,label);
          (pNum>0) ? (pNum--) : (pNum=maxLabels-1);
          PenColor(LCyan);
          strcpy(saveLine,label[pNum].aline);
          label[pNum].len=strlen(label[pNum].aline);
          label[pNum].curPos=label[pNum].newPos=0;
          XorTxtCursor(pNum,label);
          TxtDisplay(pNum,label);
          break;

        case R_Arr:

          (label[pNum].newPos)++;
          MovCursor(pNum,label);
          break;

        case L_Arr:

          (label[pNum].newPos)--;
          MovCursor(pNum,label);
          break;

        case HOME:

          label[pNum].newPos=0;
          MovCursor(pNum,label);
          break;

        case END:

          label[pNum].newPos=label[pNum].len;
          MovCursor(pNum,label);
          break;

        case(DEL,DelChar(pNum,label))

        case ESCP:

          TxtClear(pNum,label);
          label[pNum].len=strlen(label[pNum].aline);
          break;

        case BKDEL:

          if(label[pNum].newPos>0) {
            (label[pNum].newPos)--;
            MovCursor(pNum,label);
            DelChar(pNum,label);
          }
          break;

        default:

          if((Evnt1.ASCII>=32)&&(Evnt1.ASCII<126))
            AddChar(Evnt1.ASCII,pNum,label);
          break;

        } /*Switch*/

        switch(Evnt1.ASCII) {
          case ZERO:

            if(Evnt1.ScanCode==AltC) {
              cntl=False;
              strcpy(label[pNum].aline,saveLine);
            }
            switch(Evnt1.ScanCode) {

              case AltO:
              case AltD:
              case AltE:
              case AltL:
              case AltS:

                cntl=False;
                break;

              } /*switch*/
              break;

          } /*Switch*/

        } /*If*/

    } while(cntl==True); /*Do While*/

    return(Evnt1.ScanCode);

} /*EditScreen*/

/*****************************************************************************
 * SetUpScreen()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
SetUpScreen(int pNum, int maxLabels, psEDTV label) {

  int i;

  for(pNum=0;pNum<maxLabels;pNum++) {
    (pNum==0) ? PenColor(LCyan) : PenColor(LGrey);
    label[pNum].len=strlen(label[pNum].aline);
    strcpy(saveLine,label[pNum].aline);
    TxtClear(pNum,label);
    strcpy(label[pNum].aline,saveLine);
    if(pNum!=0)
      XorTxtCursor(pNum,label);
    TxtDisplay(pNum,label);
```

EDITOR.C

```
  }
  strcpy(saveLine,label[pNum=0].aline);

} /*SetUpScreen*/

/*--------------------------------------------------------------------
 * TxtClear()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
TxtClear(int pNum, psEDTV label) {

  int i;

  for(i=0;label[pNum].maxlen;i++)
    *(label[pNum].aline+i)='\0';
  PenMode(zREPz);
  PtfRoundRect(label[pNum].R,8,8,0);
  label[pNum].newPos=0;
  XorTxtCursor(pNum,label);
  MovCursor(pNum,label);

} /*TxtClear*/

/*--------------------------------------------------------------------
 * MovCursor()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
MovCursor(int pNum, psEDTV label) {

  int newPos=label[pNum].newPos;

  (newPos) ? (newPos=0) : NULL;
  (newPos>label[pNum].len) ? (newPos=label[pNum].len) : NULL;
  XorTxtCursor(pNum,label);
  label[pNum].curPos=newPos;
  XorTxtCursor(pNum,label);

} /*MovCursor*/

/*--------------------------------------------------------------------
 * XorTxtCursor()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
XorTxtCursor(int pNum, psEDTV label) {

  int xPos=label[pNum].R->xmin+3,yPos=label[pNum].R->ymin+15;

  xPos+=8*(label[pNum].curPos);
  PenMode(zXORz);
  MoveTo(xPos,yPos);
  LineTo(xPos+5,yPos);

} /*XorTxtCursor*/

/*--------------------------------------------------------------------
 * TxtDisplay()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */
void
TxtDisplay(int pNum, psEDTV label) {

  int xPos=label[pNum].R->xmin+3,yPos=label[pNum].R->ymin+12;

  if(label[pNum].curPos<label[pNum].len) {
    XorTxtCursor(pNum,label);
    xPos+=8*(label[pNum].curPos);
    MoveTo(xPos,yPos);
    DrawText(label[pNum].aline,label[pNum].curPos,\
        label[pNum].len-label[pNum].curPos);
    DrawString(" ");
    XorTxtCursor(pNum,label);
  }

} /*TxtDisplay*/

/*--------------------------------------------------------------------
 * AddChar()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
AddChar(char ch, int pNum, psEDTV label) {

  extern char msg2[];
  int i;

  if(label[pNum].len<label[pNum].maxlen-1) {
    i=0;
    while(label[pNum].curPos) {
      *(msg2+i)= *(label[pNum].aline+i);
      i++;
    }
    *(msg2+i)=ch;
    i++;
    while(label[pNum].len+1) {
      *(msg2+i)= *(label[pNum].aline+i-1);
      i++;
    }
    *(msg2+i)='\0';
    strcpy(label[pNum].aline,msg2);
    label[pNum].len=strlen(label[pNum].aline);
    TxtDisplay(pNum,label);
    label[pNum].newPos=label[pNum].curPos+1;
    MovCursor(pNum,label);
  }

} /*AddChar*/

/*--------------------------------------------------------------------
 * DelChar()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
DelChar(int pNum, psEDTV label) {

  extern char msg2[];
  int i;

  if((label[pNum].curPos>0)&& \
    (label[pNum].curPos<label[pNum].len)) {
    i=0;
    while(label[pNum].curPos) {
      *(msg2+i)= *(label[pNum].aline+i);
      i++;
    }
    while(label[pNum].len) {
      i++;
      *(msg2+i-1 )= *(label[pNum].aline+i);
    }
    *(msg2+i-1 )='\0';
    strcpy(label[pNum].aline,msg2);
    label[pNum].len=strlen(label[pNum].aline);
    TxtDisplay(pNum,label);
  }

} /*DelChar*/
```

GRAFIX.C

```
/*****************************************************************

File:    Grafix.c
Version: 4.0
Release: 1.0
Authors: Salih Kabey
Date:    1st July 1989
Purpose: PC-Based DSP Workstation using TMS320C25
Usage:   vport4
Arg Description: n/a
Returns: " Maximum Happiness !!! "
Import List: n/a

*****************************************************************

Copyright (C) Salih Kabey, The University, Department of Engineering,
Leicester LE1 7RH.  April 1989. All rights reserved.
No part of this program may be reprinted, reproduced or utilised in any
form or by any electronic, mechanical or other means, now known or
hereafter invented, including photocopying or recording, or in any
information storage or retrieval system, without permission in writing
from the author.

*****************************************************************/

#include "c:\c5\vport.h"

/*****************************************************************
 * DrawDetails()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
DrawDetails(void) {

  char hiVal[8],loVal[8],title[8],units[8], *fbotPtr;
  int limit;

  switch(sigSts) {

    case TIME:
      if(timsCal[timsCalIdx]==1.0) {

        sprintf(hiVal," %1.0f",timsCal[timsCalIdx]);
        sprintf(loVal," -%1.0f",timsCal[timsCalIdx]);
      }
      else {
        sprintf(hiVal," %1.1f",timsCal[timsCalIdx]);
        sprintf(loVal,"-%1.1f",timsCal[timsCalIdx]);
      }
      sprintf(title,"Time");
      sprintf(units," (V)");
      break;

    case PSD:
      if(psdCal[psdCalIdx]==1)
        sprintf(hiVal," %1.0f",psdCal[psdCalIdx]);
      else
        sprintf(hiVal,"%1.2f",psdCal[psdCalIdx]);
      sprintf(loVal," 0 ");
      sprintf(title," PSD");
      sprintf(units," (V)");
      break;

    case MAG:
      if(xfrCal[xfrCalIdx]==1)
        sprintf(hiVal," %1.0f",xfrCal[xfrCalIdx]);
      else
        sprintf(hiVal,"%1.2f",xfrCal[xfrCalIdx]);
      sprintf(loVal," 0 ");
      sprintf(title," MAG");
      if(xfrSts==XFERA)
        sprintf(units," 'A'");
      else
        sprintf(units," 'B'");
      break;

    case PHAS:
      sprintf(hiVal,"+180");
      sprintf(loVal,"-180");
      sprintf(title,"Phas");
      if(xfrSts==XFERA)
        sprintf(units," 'A'");
      else
        sprintf(units," 'B'");
      break;

    case COH:
      sprintf(hiVal," 1.0");
      sprintf(loVal," 0 ");
      sprintf(title," Coh");
      if(xfrSts==XFERA)
        sprintf(units," 'A'");
      else
        sprintf(units," 'B'");
      break;
  }

  BackColor(Black);
  PenColor(LRed);
  MoveTo(X_AxisHi,Y_AxisHi);
  DrawString("  ");
  MoveTo(X_AxisHi,Y_AxisHi);
  DrawString(hiVal);
  MoveTo(X_AxisTitle,Y_AxisTitle);
  DrawString(title);
  MoveTo(X_AxisUnits,Y_AxisUnits);
  DrawString(units);
  MoveTo(X_AxisLo,Y_AxisLo);
  DrawString("  ");
  MoveTo(X_AxisLo,Y_AxisLo);
  DrawString(loVal);

}

/*****************************************************************
 * KeyDrw(K,L,n)
 *
 * Purpose:   Draws a frame and flood fills it with the background
 *            colour and adds the label string. The selection cahracter
 *            is underscored.
 * Parameters: The rectangle K, label string L and underscore char n.
 * Returns:   None.
 * Action:    Defines key labels for display on the active bit-map.
 */

 void
KeyDrw(rect *K, char *L, int n) {

  FillRoundRect(K,8,8,0);
  if(n>0) {
    MoveTo(K->xmin+6+n*8,K->ymin+15);
    DrawString("_");
  }
  MoveTo(K->xmin+6,K->ymin+13);
  DrawString(L);

} /* End of KeyDrw */

/*****************************************************************
 * KeySts()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

 void
KeySts(rect **menKey) {

  rect R,S;

  SetPort(Init_Prt);
  if(keyOff!=OFF) {
    SetRect(&R,menKey[keyOff]->xmin+73,menKey[keyOff]->ymin+2,
    menKey[keyOff]->xmax-3,menKey[keyOff]->ymax-2);
    BackColor(LBlue);
    FillRect(&R,0);
  }
  if(keyOn!=OFF) {
    SetRect(&S,menKey[keyOn]->xmin+73,menKey[keyOn]->ymin+2,\
    menKey[keyOn]->xmax-3,menKey[keyOn]->ymax-2);
    BackColor(LPurp);
    FillRect(&S,0);
  }
  BackColor(Black);
  SetPort(Wave_Prt);

} /*KeySts*/

/*****************************************************************
 * PortSel()
 *
 * Purpose:   To display the current status of the selected at port on
 *            Wave_Port display panel.
 * Parameters: P - port frame, port label, port background pattern and
 *            the port frame limits.
 * Returns:   None.
```

GRAFIX.C

```
* Action:    The pen settings are set on entry, and active ports displayed
*           with a blue background and inactive ports are shaded.
*/

void
PortSel(void) {

  int i;
  rect P;

  i=actPort;
  SetRect(&P,PortS[i].x0,PortS[i].y0,PortS[i].x1,PortS[i].y1);
  PenColor(LWhite);
  (PortS[i].status==True) ? BackColor(Grey) : BackColor(Blue);
  FillRoundRect(&P,8,8,0);
  MoveTo(PortS[i].x0+12,PortS[i].y0+12);
  DrawString(PortS[i].tag);   /*Draw the Port label*/
  BackColor(Black);

} /*End of PortSel*/

/*-----------------------------------------------------------------------*
 * GridDrw(R)
 *
 * Purpose:   To draw a grid over the waveform plotting window.
 *
 * Parameters: The rectangle R, defining the window area.
 * Returns:   None.
 * Action:    The grid points are equispaced and drawn in red.
 */

void
GridDrw(rect *R) {

  int i,j,hstep,vstep;

  hstep=(R->xmax-R->xmin-10)/10;
  vstep=(R->ymax-R->ymin-10)/10;
  BackColor(Black);
  PenColor(LRed);
  for(j=0;j<11;j++) {
    for(i=0;i<11;i++)
      SetPixel(R->xmin+i*hstep+5,R->ymin+j*vstep+5);
  }

} /*End of GridDrw */

/*-----------------------------------------------------------------------*
 * TickAxes(R)
 *
 * Purpose:   To calculate and draw 10 equidistant tick marks around a
 *           rectangle R, with offsets of 5-pixels from either boundary
 *           and a tick length of 4-pixels.
 * Parameters: The rectangle R, conforming to MacWindow RECT structure.
 * Returns:   None.
 * Action:    The ticks will be drawn with the pen settings on entry.
 */

void
TickAxes(rect *R) {

  int i,hstep,vstep,tick=4;

  hstep=(R->xmax-R->xmin-10)/10;  /*Compute the Horizontal Step Length*/
  vstep=(R->ymax-R->ymin-10)/10;  /*  "    "  Vertical Step Length  */

/*Do the upper horizontal axis first*/
  for(i=0;i<11;i++) {
    MoveTo(R->xmin+5+i*hstep,R->ymin);
    LineTo(R->xmin+5+i*hstep,R->ymin+tick);
  }
/*Now the lower horizontal axis*/
  for(i=0;i<11;i++) {
    MoveTo(R->xmin+5+i*hstep,R->ymax);
    LineTo(R->xmin+5+i*hstep,R->ymax-tick);
  }
/*On to the left vertical axis*/
  for(i=0;i<11;i++) {
    MoveTo(R->xmin,R->ymin+5+i*vstep);
    LineTo(R->xmin+tick,R->ymin+5+i*vstep);
  }
/*Finally, the right vertical axis*/
  for(i=0;i<11;i++) {
    MoveTo(R->xmax-tick,R->ymin+5+i*vstep);
    LineTo(R->xmax,R->ymin+5+i*vstep);
  }

} /*End of TickAxes*/

/*-----------------------------------------------------------------------*
 * TrakSig(x,y)
 *
 * Purpose:   To place a highlight over the displayed signal
 *
 * Parameters: (x) is the horizontal displacement & (y) the amplitude
 *           as stored in an array.
 * Returns:   None.
 * Action:    A highlight is placed just offset to the displayed signal.
 */

void
TrakSig(int x, int y) {

  int j,offset=0;
  float X,Y,Xm,Ym,dX,dY,Fs;

  Fs=2.56*SFreq[sfi];
  Xm=X_Mrk-50;

  switch(sigSts) {
    case TIME:
      Y= *(timbf+x+1+N2*actPort)*5.0/2048.0;
      Ym= *(timbf+(int)Xm+N2*actPort)*5.0/2048.0;
      X=x/Fs; dX=(x-Xm)/Fs;
      break;

    case PSD:
      Y= *(psdbf+x+N2*actPort);
      Ym= *(psdbf+(int)Xm+N2*actPort);
      X=x*Fs/1024; dX=(x-Xm)*Fs/1024;
      break;

    case MAG:
      if(xfrSts==XFERB)
        offset=N2;
      Y= *(mag+x+offset);
      Ym= *(mag+(int)Xm+offset);
      X=x*Fs/1024; dX=(x-Xm)*Fs/1024;
      break;

    case PHAS:
      if(xfrSts==XFERB)
        offset=N2;
      Y= *(phas+x+offset);
      Ym= *(phas+(int)Xm+offset);
      X=x*Fs/1024; dX=(x-Xm)*Fs/1024;
      break;

    case COH:
      if(xfrSts==XFERB)
        offset=N2;
      Y= *(coh+x+offset);
      Ym= *(coh+(int)Xm+offset);
      X=x*Fs/1024; dX=(x-Xm)*Fs/1024;
      break;
  }

  if(avgeSts==True) {
    Y=avgePrm;
    Ym=avgePrm;
  }

  if(Delta_Axs==True) {

    dY=Y-Ym;

    switch(sigSts) {
      case TIME:dXYTime(dX,dY))
      case PSD:dXYSpec(dX,dY))
      case MAG:dXYSpec(dX,dY))
      case PHAS:dXYPhas(dX,dY))
      case COH:dXYCoh(dX,dY))
    }
  }
  else {
    switch(sigSts) {
      case TIME:XYTime(X,Y))
      case PSD:XYSpec(X,Y))
      case MAG:XYSpec(X,Y))
      case PHAS:XYPhas(X,Y))
      case COH:XYCoh(X,Y))
    }
  }
  PenColor(LRed);
  MoveTo(60,300);
  DrawString(msg2);

} /*TrakSig*/

/*-----------------------------------------------------------------------*
 * dXYTime()
 *
```

GRAFIX.C

```
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
dXYTime(float dX, float dY) {

  int j;

  if(dX)
    j=sprintf(msg2,"dX: %+3.3f mSec  ",1000*dX);
  else
    j=sprintf(msg2,"dX: %+6.3f Sec   ",dX);
  if(abs(dY))
    j+=sprintf(msg2+j," dY: %+3.1f mV    ",1000*dY);
  else
    j+=sprintf(msg2+j," dY: %+6.3f  V    ",dY);

}

/*-------------------------------------------------------------------------*
 * dXYSpec()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
dXYSpec(float dX, float dY) {

  int j;

  if(dX)
    j=sprintf(msg2,"dX: %+6.5f Hz   ",dX);
  else
    j=sprintf(msg2,"dX: %+6.3f Hz   ",dX);
  if(abs(dY))
    j+=sprintf(msg2+j," dY: %+3.3f mV    ",1000*dY);
  else
    j+=sprintf(msg2+j," dY: %+6.3f  V    ",dY);

}

/*-------------------------------------------------------------------------*
 * dXYPhase()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
dXYPhase(float dX, float dY) {

  int j;

  if(dX)
    j=sprintf(msg2,"dX: %+4.3f Hz   ",dX);
  else
    j=sprintf(msg2,"dX: %+6.3f Hz   ",dX);
  if(abs(dY))
    j+=sprintf(msg2+j," dY: %+4.3f degs  ",dY);
  else
    j+=sprintf(msg2+j," dY: %+6.3f degs  ",dY);

}

/*-------------------------------------------------------------------------*
 * dXYCoh()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
dXYCoh(float dX, float dY) {

  int j;

  if(dX)
    j=sprintf(msg2,"dX: %+4.3f Hz   ",dX);
  else
    j=sprintf(msg2,"dX: %+6.3f Hz   ",dX);
  if(abs(dY))
    j+=sprintf(msg2+j," dY: %+4.3f     ",dY);
  else
    j+=sprintf(msg2+j," dY: %+6.3f     ",dY);

}

/*-------------------------------------------------------------------------*
 * XYTime()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
XYTime(float X, float Y) {

  int j;

  if(X)
    j=sprintf(msg2,"X: %+3.3f mSec  ",1000*X);
  else
    j=sprintf(msg2,"X: %+6.3f Sec   ",X);
  if(abs(Y))
    j+=sprintf(msg2+j," Y: %+3.1f mV     ",1000*Y);
  else
    j+=sprintf(msg2+j," Y: %+6.3f  V     ",Y);

}

/*-------------------------------------------------------------------------*
 * XYSpec()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
XYSpec(float X, float Y) {

  int j;

  if(X)
    j=sprintf(msg2,"X: %+4.3f Hz   ",X);
  else
    j=sprintf(msg2,"X: %+6.3f Hz   ",X);
  if(abs(Y))
    j+=sprintf(msg2+j," Y: %+3.3f mV     ",1000*Y);
  else
    j+=sprintf(msg2+j," Y: %+6.3f  V     ",Y);

}

/*-------------------------------------------------------------------------*
 * XYPhase()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
XYPhase(float X, float Y) {

  int j;

  if(X)
    j=sprintf(msg2,"X: %+4.3f Hz   ",X);
  else
    j=sprintf(msg2,"X: %+6.3f Hz   ",X);
  if(abs(Y))
    j+=sprintf(msg2+j," Y: %+4.3f degs   ",Y);
  else
    j+=sprintf(msg2+j," Y: %+6.3f degs   ",Y);

}

/*-------------------------------------------------------------------------*
 * XYCoh()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
```

GRAFIX.C

```
XYCoh(float X, float Y) {

  int j;

  if(X)
    j=sprintf(msg2,"X: %+4.3f Hz   ",X);
  else
    j=sprintf(msg2,"X: %+6.3f Hz   ",X);
  if(abs(Y))
    j+=sprintf(msg2+j," Y: %4.3f        ",Y);
  else
    j+=sprintf(msg2+j," Y: %6.3f        ",Y);

}

/*-----------------------------------------------------------------------
 * OpenWindow (R,TITLE,n)
 *
 * Purpose:   To open a pop-up window over a defined screen area.
 *
 * Parameters: (R): defines the pop-up window dimensions.
 *          (TITLE): is the label attached to the window, and
 *          (n): determines the type of response buttons.
 * Returns:   None.
 * Action:    The ticks will be drawn with the pen settings on entry.
 */

void
OpenWindow (Image **img, rect *R, char *TITLE, char n, int barCol) {

  rect tR,tP;
  long imBytes;

  if ((imBytes = ImageSize(R)) 64000) {
    ClearText();
    SetDisplay(TextPg0);
    printf("Image too large %05d \n",imBytes);
    exit(1);
  }
  if((*img = (Image *)_fmalloc((unsigned int) imBytes)) == NULL) {
    ClearText();
    SetDisplay(TextPg0);
    printf("Insufficient memory\n");
    printf("size = 0x%x bytes\n",imBytes);
    printf("ptr = %lp\n",*img);
    exit(2);
  }
  PenColor(Blue);
  ReadImage(R,*img);      /* Save the window area */
  FillRect(R,1);          /* Clear the window     */

  SetRect(&tR,R->xmin,R->ymin,R->xmax,R->ymin+15);
  PenColor(barCol);
  FillRect(&tR,1);

  MoveTo(tR.xmin+3,tR.ymin+12);
  BackColor(barCol);
  PenColor(White);
  DrawString(TITLE);

  BackColor(LBlue);
  switch(n) {
    case YesNo:
      {
      PenColor(LWhite);
      SetRect(&tP,R->xmin+85,R->ymax-30,R->xmin+125,R->ymax-10);
      FillRoundRect(&tP,8,8,0);
      MoveTo(tP.xmin+8,tP.ymin+13);
      DrawString(Yes_Tag);
      SetRect(&tP,R->xmin+175,R->ymax-30,R->xmin+215,R->ymax-10);
      FillRoundRect(&tP,8,8,0);
      MoveTo(tP.xmin+12,tP.ymin+13);
      DrawString(No_Tag);
      break;
      }
    case ARCHIVE:
      {
      PenColor(LWhite);

      SetRect(&tP,R->xmin+20,R->ymax-30,R->xmin+64,R->ymax-10);
      FillRoundRect(&tP,8,8,0);
      MoveTo(tP.xmin+8,tP.ymin+13);
      DrawString(archDisk_Tag);

      SetRect(&tP,R->xmin+70,R->ymax-30,R->xmin+114,R->ymax-10);
      FillRoundRect(&tP,8,8,0);
      MoveTo(tP.xmin+8,tP.ymin+13);
      DrawString(archLoad_Tag);

      SetRect(&tP,R->xmin+120,R->ymax-30,R->xmin+164,R->ymax-10);
      FillRoundRect(&tP,8,8,0);
      MoveTo(tP.xmin+8,tP.ymin+13);
      DrawString(archSave_Tag);

      SetRect(&tP,R->xmin+170,R->ymax-30,R->xmin+214,R->ymax-10);
      FillRoundRect(&tP,8,8,0);
      MoveTo(tP.xmin+3,tP.ymin+13);
      DrawString(archErase_Tag);

      SetRect(&tP,R->xmin+250,R->ymax-30,R->xmin+282,R->ymax-10);
      FillRoundRect(&tP,8,8,0);
      MoveTo(tP.xmin+8,tP.ymin+13);
      DrawString(OK_Tag);

      SetRect(&tP,R->xmin+287,R->ymax-30,R->xmin+340,R->ymax-10);
      FillRoundRect(&tP,8,8,0);
      MoveTo(tP.xmin+4,tP.ymin+13);
      DrawString(Cancel_Tag);
      break;
      }
    case OKAY:
      {
      PenColor(LWhite);
      SetRect(&tP,R->xmin+85,R->ymax-30,R->xmin+149,R->ymax-10);
      FillRoundRect(&tP,8,8,0);
      MoveTo(tP.xmin+24,tP.ymin+13);
      DrawString(OK_Tag);
      SetRect(&tP,R->xmin+199,R->ymax-30,R->xmin+263,R->ymax-10);
      FillRoundRect(&tP,8,8,0);
      MoveTo(tP.xmin+8,tP.ymin+13);
      DrawString(Cancel_Tag);
      break;
      }

  } /*Switch*/

  PenColor(White);
  BackColor(Blue);

} /*OpenWindow*/

/*-----------------------------------------------------------------------
 * CloseWindow (R)
 *
 * Purpose:   To close a pop-up window over a defined screen area.
 *
 * Parameters: (R): defines the pop-up window dimensions.
 *
 * Returns:   None.
 * Action:    Restores the image that was under the pop-up.
 */

void
CloseWindow(Image *img, rect *R) {

  RasterOp(Replace);
  WriteImage(R,img);  /* Restore the window area */
  _ffree((void *)img);
  BackColor(Black);
  PenColor(LWhite);

} /*CloseWindow*/

/*-----------------------------------------------------------------------
 * Grid()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
Grid(void) {

  SetPort(Init_Prt);
  InvertRoundRect(&Grid_Key,8,8);
  SetPort(Wave_Prt);
  GridDraw(&Axes_Wave);
  Delay1();
  SetPort(Init_Prt);
  InvertRoundRect(&Grid_Key,8,8);
  SetPort(Wave_Prt);

}

/*-----------------------------------------------------------------------
 * Trace()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
```

GRAFIX.C

```
*/

void
Trace(rect **menKey) {

  int j,errsl;
  sTRAKPt trakPt;

  InvertRoundRect(&Trace_Key,8,8);
  trakPt.x=200+50;
  trakPt.y=InitTrakBuf()+200;
  TracerOn(trakPt.x,plotbf[trakPt.y]);
  TrakSig(trakPt.x-50,trakPt.y);
  errsl=True;
  do {
    if((KeyEvent(False,&Evnt1)==True)) {
    switch(Evnt1.ScanCode) {

      case(L_Arr,LTrace(&trakPt))
      case(R_Arr,RTrace(&trakPt))
      case(U_Arr,UTrace(&trakPt))
      case(D_Arr,DTrace(&trakPt))

      default:

        switch(Evnt1.ASCII" ') {
        case 'r':

          TracerOn(trakPt.x,plotbf[trakPt.y]);
          InvertRoundRect(&Trace_Key,8,8);
          if(Delta_Axs==True) {
          Delta_Axs=False;
          PenColor(LRed);
          for(j=25;j<275;j+=5)
            SetPixel(X_Mrk,j);
          }
          TrakSig(trakPt.x-50,trakPt.y);
          errsl=False;
          break;

        case 'g':

          Grid();
          break;

/* Mark */    case 'k':

          InvertRoundRect(&Mark_Key,8,8);
          if(Delta_Axs==True) {
            Delta_Axs=False;
            /*TrakSig(trakPt.x-50,trakPt.y);*/
          }
          else {
            Delta_Axs=True;
            X_Mrk=trakPt.x;
            TrakSig(X_Mrk-50,trakPt.y);
          }
          PenColor(LRed);
          for(j=25;j<275;j+=5)
            SetPixel(X_Mrk,j);
          Delay1();
          InvertRoundRect(&Mark_Key,8,8);
          break;

/* Time */    case 't':

          xfrOn=False;
          keyOff=sigStr;keyOn=sigStr=TIME;
          KeyStat(menKey);
          WhichPlot();
          TracerOn(trakPt.x,plotbf[trakPt.y]);
          TrakSig(trakPt.x-50,trakPt.y);
          break;

/* Pwr Spec */  case 'w':

          xfrOn=False;
          keyOff=sigStr;keyOn=sigStr=PSD;
          KeyStat(menKey);
          WhichPlot();
          TracerOn(trakPt.x,plotbf[trakPt.y]);
          TrakSig(trakPt.x-50,trakPt.y);
          break;

/* XferPn */  case 'x':

          if(xfrOn==False) {
            xfrOn=True;
            keyOff=sigStr;keyOn=sigStr=TIME;
            KeyStat(menKey);
          }
          SetPort(Init_Prt);
          BackColor(LBlue);
          PenColor(LWhite);
          if(xfrSts==XFERA) {
            xfrSts=XFERB;
            KeyDrw(&TranF_Key,TranFB_Tag,0);
          }
          else {
            xfrSts=XFERA;
            KeyDrw(&TranF_Key,TranFA_Tag,0);
          }
          InvertRoundRect(&TranF_Key,8,8);
          Delay1();
          InvertRoundRect(&TranF_Key,8,8);
          SetPort(Wave_Prt);
          WhichPlot();
          trakPt.y=InitTrakBuf()+trakPt.x-50;
          TracerOn(trakPt.x,plotbf[trakPt.y]);
          TrakSig(trakPt.x-50,trakPt.y);
          break;

/* Modules */  case 'm':

          xfrOn=True;
          keyOff=sigStr;keyOn=sigStr=MAG;
          KeyStat(menKey);
          WhichPlot();
          trakPt.y=InitTrakBuf()+trakPt.x-50;
          TracerOn(trakPt.x,plotbf[trakPt.y]);
          TrakSig(trakPt.x-50,trakPt.y);
          break;

/* Phase */    case 'p':

          xfrOn=True;
          keyOff=sigStr;keyOn=sigStr=PHAS;
          KeyStat(menKey);
          WhichPlot();
          trakPt.y=InitTrakBuf()+trakPt.x-50;
          TracerOn(trakPt.x,plotbf[trakPt.y]);
          TrakSig(trakPt.x-50,trakPt.y);
          break;

/* Coherence */ case 'c':

          xfrOn=True;

          keyOff=sigStr;keyOn=sigStr=COH;
          KeyStat(menKey);
          WhichPlot();
          trakPt.y=InitTrakBuf()+trakPt.x-50;
          TracerOn(trakPt.x,plotbf[trakPt.y]);
          TrakSig(trakPt.x-50,trakPt.y);
          break;

        } /* switch for esc or grid */

      } /* switch */
      } /* if */
  } while(errsl==True);
}

/*-----------------------------------------------------------------------
 * TracerOn()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
TracerOn(int x, int y) {

  int i;

  PenColor(LRed);

  for(i=12;i>0;i-=2)
    if(y<25+i)
      SetPixel(x,y-i);

  for(i=12;i>0;i-=2)
    if(y>275-i)
      SetPixel(x,y+i);

}

/*-----------------------------------------------------------------------
 * TracerOff()
 *
 * Purpose:
```

GRAFIX.C

```
 * Parameters:
 * Returns:
 * Action:
 */

void
TracerOff(int x, int y) {

  int i;

  for(i=12;i0;i-=2)
    if(y25+i)
      SetPixel(x,y-i);

  for(i=12;i0;i-=2)
    if(y -i)
      SetPixel(x,y+i);

}

/*****************************************************************
 * InitTrakBuf()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

int
InitTrakBuf(void) {

  int i;

  if(xfrOn==True) {
    switch(xfrWin) {
      case(XFERA):i=0)
      case(XFERB):i=N2)
    }
  }
  else
    i=N2*actPort;

  return(i);
}

/*****************************************************************
 * LTrace()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
LTrace(prTRAKPt trakPt) {

  TracerOn(trakPt-x,plotbf[trakPt-y]);
  if(trakPt-x50) {trakPt-y--; trakPt-x--;}
  else {trakPt-x=450; trakPt-y=InitTrakBuf()+399;}
  TracerOff(trakPt-x,plotbf[trakPt-y]);
  TrakSig(trakPt-x-50,trakPt-y);

}

/*****************************************************************
 * RTrace()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
RTrace(prTRAKPt trakPt) {

  TracerOn(trakPt-x,plotbf[trakPt-y]);
  if(trakPt-x%=) {trakPt-y++; trakPt-x++;}
  else {trakPt-x=50; trakPt-y=InitTrakBuf();}
  TracerOff(trakPt-x,plotbf[trakPt-y]);
  TrakSig(trakPt-x-50,trakPt-y);

}

/*****************************************************************
 * UTrace()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
UTrace(prTRAKPt trakPt) {

  TracerOn(trakPt-x,plotbf[trakPt-y]);
  if(trakPt-xÕ) {trakPt-y+=10; trakPt-x+=10;}
  else {trakPt-x=50; trakPt-y=InitTrakBuf();}
  TracerOff(trakPt-x,plotbf[trakPt-y]);
  TrakSig(trakPt-x-50,trakPt-y);

}

/*****************************************************************
 * DTrace()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
DTrace(prTRAKPt trakPt) {

  TracerOn(trakPt-x,plotbf[trakPt-y]);
  if(trakPt-x50) {trakPt-y-=10; trakPt-x-=10;}
  else {trakPt-x=450; trakPt-y=InitTrakBuf()+399;}
  TracerOff(trakPt-x,plotbf[trakPt-y]);
  TrakSig(trakPt-x-50,trakPt-y);

}

/*****************************************************************
 * Delay1()
 *
 * Purpose:
 * Parameters:
 * Returns:
 * Action:
 */

void
Delay1(void) {

  float n=1;

  do {
    n=(n*n*n*n)/(n*n*n);
    n++;
  }while(n--);
}
```

GLOBSVP.C

```
/*************************************************************************

File: globsvp.c
Version: 4.0
Release: 1.1
Author: Salih Kabay
Date:  3rd July 1989
Purpose: Global variables file for VPORT4.EXE application programs
Usage: n/a
Arg Description: n/a
Returns: n/a
Import List: n/a

*************************************************************************

Copyright (C) Salih Kabay, The University, Department of Engineering,
Leicester LE1 7RH.  April 1989. All rights reserved.
No part of this program may be reprinted, reproduced or utilised in any
form or by any electronic, mechanical or other means, now known or
hereafter invented, including photocopying or recording, or in any
information storage or retrieval system, without permission in writing
from the author.

*************************************************************************/

#include "hashvp.h"
#include "mstavp.h"

/************************************************************************/

/* Chars */

char *menuTitle = "              DSP Workstation ";

#ifdef USETHISSTUFF

/* Not Used Yet -
in struct PortStruc PortS[] initialisations */

char  *Port1_Tag= "P1";   /* Port Labels */
char  *Port2_Tag= "P2";
char  *Port3_Tag= "P3";
char  *Port4_Tag= "P4";

#endif


char  *SpanKey_Tag= "Freq";
char  *SpanInc_Tag= "+";
char  *SpanDec_Tag= "-";

/* Function Key Tags */

char  *Help_Tag=   "Help";
char  *Archive_Tag=  "Archive";
char  *Label_Tag=   "Label";
char  *Avge_Tag=   "Average";
char  *Grid_Tag=   "Grid";
char  *Trace_Tag=   "Trace";
char  *Mark_Tag=   "Mark";
char  *Set_Tag=    "Setup";
char  *Eqn_Tag=   "Equation";
char  *Jet_Tag=    "Jet Drive";
char  *Exit_Tag=   "Exit";

char  *Start_Tag=   "Start";
char  *Halt_Tag=   "Halt";
char  *Time_Tag=   "Time";
char  *Spectra_Tag=  "Pwr Spec";
char  *TranFA_Tag=   "Xfr Fn A";
char  *TranFB_Tag=   "Xfr Fn B";
char  *Mod_Tag=    "Modulus";
char  *Phase_Tag=   "Phase";
char  *Coh_Tag=    "Coherence";

char  *Yes_Tag=    "Yes";
char  *No_Tag=    "No";
char  *Cancel_Tag=  "Cancel";
char  *OK_Tag=    "OK";

char  *archDisk_Tag= "Disk";
char  *archLoad_Tag= "Load";
char  *archSave_Tag= "Save";
char  *archErase_Tag="Erase";

/* Pop Up Window Help & Background Information */

/*              "12345678901234567890123456789012345678" */
char  *Exit_PopTag="       Exit Key Options       ";
char  *L1_Exit=   "Make sure you have archived analysis";
char  *L2_Exit=   "data collected during this session. ";
char  *L3_Exit=   "          ...oooOoooo...          ";

char  *Exit_Ask=   "Do you still wish to Exit?       ";

char  *Set_PopTag= "       Setup Key Options       ";
char  *L1_Set=    "-  Use [illegible] to change the  ";
char  *L2_Set=    "   selected Port Number.          ";
char  *L3_Set=    "-  Press
 to accept the Port. ";

char  *Avge_PopTag= "     Spectral Averaging Options    ";
char  *L1_Avge=   "The accuracy and statistical reliability";
char  *L2_Avge=   "of measurements can be increased by";
char  *L3_Avge=   "spectral averaging (Overlap: 75%).";
char  *L4_Avge=   "-  Use [illegible] to select the number";
char  *L5_Avge=   "   of frames to be averaged.     ";

char  *Label_PopTag="     Port Label Editing Options    ";
char  *L1_Label=  " Use the keyboard to define customised ";
char  *L2_Label=  " labels for individual input ports.  ";

char *archSave_PopTag="         Filename Entry Menu    ";
char *L1_archSavePop= " The current patient data can now be ";
char *L2_archSavePop= " archived.";
char *L3_archSavePop= " Please enter an 8 character filename.";

char  *XfrA_Tag=  "System A";
char  *XfrB_Tag=  "System B";

char  *patXtn=   ".PAT";
char  *txtXtn=   ".TXT";
char  *filPAT=   "*.PAT";
char  *filTXT=   "*.TXT";

charTime_Axis,Spectra_Axis,TranF_Axis,Coh_Axis,Delta_Axis;

char Halt_Flg;

char msg2[80];
unsigned char near buffer[BUFLN];

/************************************************************************/

/* Integers */

int nfi=3;
intX_Mrk,xfrOn=False,xfrSts=XFERA,sigSts=TIME,actPort=0;
intavgeSts=False,avgeRst=True,avgePrm=0,avgeTot=1,avgeBak;
intkeyOn,keyOff,rmxPlot=False;

intxfrCalIdx=0,psdCalIdx=0,timsCalIdx=0;

int near curbase;

int timbf[2048],plotbf[NN];
intSFreq[]={1,2,5,10,20,50};
intPScal[]={100,50,20,10,5,2};

int xDir[]=
{Xr1c1,Xr1c2,Xr1c3,Xr2c1,Xr2c2,Xr2c3,Xr3c1,Xr3c2,Xr3c3,Xr4c1,Xr4c2,Xr4c3};

int yDir[]=
{Yr1c1,Yr1c2,Yr1c3,Yr2c1,Yr2c2,Yr2c3,Yr3c1,Yr3c2,Yr3c3,Yr4c1,Yr4c2,Yr4c3};

#ifdef USETHISSTUFF

/* These are defined in GPXquery.c which is the interface to GRquery.c */

int OrafixCard,CommsPort;

#endif

/************************************************************************/

/* Floats */

float scale=1.0,psdrms;

floatpsdbf[2048],gain[1024],phase[1024],coh[1024],msg[1024];

float timsCal[]={5, 2, 1, 0.5, 0.2};
float psdCal[]={20,10, 5, 2, 1, 0.5, 0.2};
float xfrCal[]={20,10,5, 2, 1, 0.5, 0.2, 0.1, 0.05};

/************************************************************************/

/* External structures */

mstaPort *Wave_Prt,*Init_Prt;
Image *imagePtr;
event Evnt;
rectPlot_Wave,Axes_Wave,Trace_Key,Grid_Key,Mark_Key,TranF_Key;

/************************************************************************/
```

GLOBSVP.C

```
/* Private structures */

#define Port1_Tag   "P1"    /* Port Labels */
#define Port2_Tag   "P2"
#define Port3_Tag   "P3"
#define Port4_Tag   "P4"

struct portDef3 {
  char *tag;
  unsigned int portn,sig;
  int x,y;
};

struct portDef3 portDef[]={
  {Port1_Tag,PORT1,PORT1,X_XfipA,Y_XfipA},
  {Port2_Tag,PORT2,PORT2,X_XfOpA,Y_XfOpA},
  {Port3_Tag,PORT3,PORT3,X_XfipB,Y_XfipB},
  {Port4_Tag,PORT4,PORT4,X_XfOpB,Y_XfOpB}
};

struct PortStruc {
  char *tag;
  int status,x0,y0,x1,y1;
};

struct PortStruc Port3[]={
  {Port1_Tag,True,X0_P1,Y0_P1,X1_P1,Y1_P1},
  {Port2_Tag,False,X0_P2,Y0_P2,X1_P2,Y1_P2},
  {Port3_Tag,False,X0_P3,Y0_P3,X1_P3,Y1_P3},
  {Port4_Tag,False,X0_P4,Y0_P4,X1_P4,Y1_P4}
};

typedef struct {
  char *sline;
  int len,maxlen,newPos,curPos;
  rect *R;
}sEDTV, *psEDTV;

rectLabelP1_Box,LabelP2_Box,LabelP3_Box,LabelP4_Box,archSave_Box;

char currentDir[MAXDIRNAME+1];

char p1Lbl[MAXEDITCHARS]="P1: Ventilator Valve Drive Signal";
char p2Lbl[MAXEDITCHARS]="P2: Airways Pulsatile Pressure";
char p3Lbl[MAXEDITCHARS]="P3: Thoracic Wall Movements";
char p4Lbl[MAXEDITCHARS]="P4: Abdominal Wall Movements";

sEDTV portLabel[]={
  {p1LM,MAXEDITCHARS,MAXEDITCHARS,0,0,&LabelP1_Box},
  {p2LM,MAXEDITCHARS,MAXEDITCHARS,0,0,&LabelP2_Box},
  {p3LM,MAXEDITCHARS,MAXEDITCHARS,0,0,&LabelP3_Box},
  {p4LM,MAXEDITCHARS,MAXEDITCHARS,0,0,&LabelP4_Box}
};

char fname[10]="";

sEDTVfileName[]={fname,9,9,0,0,&archSave_Box};

char *Archive_PopTag=" Patient Database and Archive Management";
char *Surname_Tag="Surname";
char *Forename_Tag="Forenames";
char *Sex_Tag="Sex";
char *Birth_Tag="Date of Birth";
char *Status_Tag="Status";
char *Job_Tag="Occupation";
char *Disk_Tag="Directory";

charpatSurname[MAXSURNAME]="Richards";
charpatFore[MAXFORENAME]="Clarfenkel";
charpatBirth[MAXBIRTH]="13/01/54";
charpatSts[MAXSTATUS]="Single";
charpatJob[MAXJOB]="Tailor";
charpatSex[MAXSEX]="Male";

charpatDisk[MAXDISK]="HOWARD";

rectpatSurname_Box,patFore_Box,patBirth_Box,patSts_Box;
rectpatJob_Box,patSex_Box,patDisk_Box;

sEDTV archLabel[]={
   {patSurname,MAXSURNAME,MAXSURNAME,0,0,&patSurname_Box},
   {patFore,MAXFORENAME,MAXFORENAME,0,0,&patFore_Box},
   {patBirth,MAXBIRTH,MAXBIRTH,0,0,&patBirth_Box},
   {patSts,MAXSTATUS,MAXSTATUS,0,0,&patSts_Box},
   {patJob,MAXJOB,MAXJOB,0,0,&patJob_Box},
   {patSex,MAXSEX,MAXSEX,0,0,&patSex_Box},
};

#include <_____.h>
struct find_t pat_file,txt_file;

typedef struct {
  char attr,name[13];
  long size;
}sDIR, *psDIR;

sDIR dirPat[100],dirTXT[100];

typedef struct {
  int dirStart,maxFile,xId,yId;
} sHILIGHT, *psHILIGHT;

typedef struct {
  int x,y;
}sTRAKPt, *psTRAKPt;

/*------------------------------------------------------------------------------*/

char saveLine[100];

/*-------------------------*/
```

Appendix 6

Source Code Listings for MRAC Simulation Described in Chapter 7.

```
PROGRAM AC2PD

INITIAL
      CINTERVAL   CINT = 0.01
      CONSTANT    R = 10.0, C = 0.05, L = 0.153 ...
                  ,Rp = 10.0, CpI = 0.05, Cp = 0.025, Lp = 0.153
      CONSTANT    Gamma = 1.0, Theta = 0.015
      CONSTANT    P2aI = 0.0, P2apI = 0.0, dP2aI = 0.0, dP2apI = 0.0
      CONSTANT    deI = 0.0, K = 5.0, Kd = 1.0, PiIC=8.0
      CONSTANT    TSTP=24.0, TSTRT=0.01
      CONSTANT    PWIDTH=3.0,TDELAY=0.0,PERIOD=6.0

      Lambda=Gamma
      Thi=Theta
      T1 = R/L
      T2 = 1/(L*C)
      T1p = Rp/Lp

END $" OF INITIAL "
DERIVATIVE
      T2p = 1/Lp*RSW(T .LT. TSTRT, CpI, Cp)
      Pi = PiIC*PULSE(TDELAY,PERIOD,PWIDTH)

      dP2a = T2*(-Pa+Pi)-T1*P2a
      P2a = INTEG(dP2a,dP2aI)
      Pa = INTEG(P2a,P2aI)

      dP2ap = T2p*(-Pap+Pil)-T1p*P2ap
      P2ap = INTEG(dP2ap,dP2apI)
      Pap = INTEG(P2ap,P2apI)

      Pil = K2*Pi-K1*Pap
      de = P2a-P2ap
      e = INTEG(de,deI)
      PDe = K*e+Kd*de
      Ep = PDe*Pap
      Ei = PDe*Pi/T2p

      dK1 = -(Gamma*Ep)-Thi*DERIVT(0.0,Ep)
      dK2 = Lambda*Ei+Theta*DERIVT(0.0,Ei)
      K1 = INTEG(dK1,0.0)
      K2 = INTEG(dK2,1.0)
      TERMT(T.GE.TSTP)

END $" OF DERIVATIVE "
END $" OF PROGRAM "
```

Environments For Real-Time Measurement And Control: A Study Of HFJV In Anaesthesia

Salih Kabay

Thesis Submitted For The Degree Of
Doctor Of Philosophy

Abstract

This thesis is concerned with intelligent computer-based instrumentation which can be easily adapted for measurement, modelling and control in a range of application domains. The particular application area selected for study and used to illustrate the main features of the scheme was in anaesthesia for measurement and control of high-frequency jet ventilation (HFJV). The analytical methods and experimental procedures required for this area are also applicable to many other areas throughout engineering and biomedicine. A prototype general-purpose signal processing computer (SPC) encompassing many new design concepts was built to provide a flexible and user-friendly system for performing dedicated measurement and control tasks - as specified by the application program interface.

The other objective of this study was to develop a new measurement and control environment for investigating the underlying respiratory dynamics of patient airways during HFJV - to facilitate a study of the efficacy of this mode of ventilation. Drawing on past experience with the SPC, a new computer-system was designed which overcame the bandwidth limitations of the original SPC. Based around powerful, modern and cost-effective commercial system hardware it is shown that this second-generation SPC can perform real-time measurement, modelling and control in HFJV as required.

Modifications were carried out on an existing high-frequency jet ventilator to allow new modes of respiratory excitation. The signal processing system described together with these modifications to the jet ventilator coupled with the development of a new non-invasive fibre-optic chest-wall displacement transducer forms a complete environment which permits systematic identification of respiratory dynamics with extremely high precision in a fraction of the time taken by previous workers in this field. This is achieved with only minor changes to existing jet ventilation equipment and procedures. The system is intended to cope with the volume of information that needs to be considered during HFJV and the level of complexity that this method of ventilation entails.

The measurement environment has undergone clinical trials on a small population of patients. The study clearly validated the hypothesis that the respiratory airways exhibit characteristics similar to an acoustic resonant circuit over the range of frequencies covered by HFJV. Based on this study, a Lyapunov model-reference adaptive control (MRAC) system has been designed and simulated for performing automatic control of HFJV and tested for stability and convergence over a wide range of operating conditions.

The thesis concludes with a consideration of how the presented approach can be extended to take account of new hardware and software developments.