4-29-1990

Microcomputer-based system for the study of the respiratory system in newborns

Nelson Remberto Claure
Florida International University

DOI: 10.25148/etd.FI14060852

ABSTRACT OF THE THESIS

Microcomputer Based System for the Study of
the Respiratory System in Newborns

by

Nelson Remberto Claure
Florida International University, 1990
Miami, Florida
Professor Wunnava V. Subbarao, Major Professor

A useful understanding of the respiratory system of premature infants and the factors contributing to different physiological mechanisms and diseases requires extensive clinical research. This project is the result of a need for a fast and reliable system to process the information obtained from biological sources and to obtain results from which different hypothesis can be tested.

This document presents a description of one such system and its different subsystems. It describes the biosignals of interest as well as the stages they have to go through in order to obtain an accurate and valid analysis.

The system is hardware and software oriented. The system hardware is subdivided into instrumentation system, which is used to pick up and condition the signals, and a data acquisition, monitoring and storage system, where the signals are digitized and stored for later processing. The system software, which is the basic and principal component of the project, participates in the hardware control for the data acquisition, storage and monitoring, as well as the posterior stages of signal processing and analysis, which constitute the key of the system.

The biosignals mentioned above can be classified as muscular or EMG, respiratory, chest wall motion, and cardiac signals. The muscular signals are obtained from measuring the electrical activity of the muscles participating in the process of ventilation and the respiratory signals reflect mechanical characteristics of the lungs and airway passages, the chest wall motion signals give a measurement to evaluate the chest

wall stability, and the cardiac signals which are measurements of the electrical activity irradiated by the cardiac muscle.

These biosignals require extensive processing, especially the EMG signals, before analysis. The signal processing stage uses digital signal processing techniques which were developed or adapted for this purpose.

The signal analysis stage is based on research protocols and physical relations to evaluate different respiratory parameters. Special data and file handling software was developed and applied as well as graphics software, to accomplish the stages mentioned above.

FLORIDA INTERNATIONAL UNIVERSITY
Miami, Florida

Microcomputer Based System for the Study of the Respiratory System in Newborns

A thesis submitted in partial satisfaction of the
requirements for the degree of Master of Science
in Electrical Engineering

by

Nelson Remberto Claure

1990

To Professors: Dr. Wunnava V. Subbarao

Dr. Tadeusz Babij

Dr. Malcolm Heimer

and

Dr. Shahnaz Duara

This thesis, having been approved in respect to form and mechanical execution, is referred to you for judgement upon its substantial merit.

Dean Dr. Gordon Hopkins
College of Engineering

The thesis of Nelson Remberto Claure is approved.

Dr. Tadeusz Babij

Dr. Malcolm Heimer

Dr. Shahnaz Duara

Dr. Wunnava V. Subbarao, Major Professor

Date of Examination: April 29, 1990

Dean Dr. Richard Campbell
Division of Graduate Studies

FLORIDA INTERNATIONAL UNIVERSITY, 1990

A mis padres Luis y Martha, mis hermanos y amigos.

ACKNOWLEDGEMENTS

I thank Dr. Wunnava Subbarao and Dr. Shahnaz Duara for their guide and support that help me accomplish this goal. I also thank Dr. Vijay Raman, Dr. Tadeusz Babij and Dr. Malcolm Heimer for the valuable knowledge they have shared with me.

Special acknowledgements to Dr. James Story, Director of the Electrical Engineering Department of Florida International University, and to Dr. Eduardo Bancalari, Director of the Division of Neonatology, Department of Pediatrics, University of Miami School of Medicine.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

1.1 SYNOPSIS

The immature respiratory system of the premature newborn is object of different research studies in order to obtain a useful understanding of its special characteristics, its development, and diseases.

The information obtained from these studies is used to modify therapy that newborns receive at the intensive care units. This information and new findings are also contributions to the area of pediatric applied physiology.

The clinical research objective is to study the relation and interaction between respiratory muscle activity, mechanics of the respiratory system, and the chest wall stability in the premature newborns.

The design and development of this microcomputer-based system for the study of the different components of the respiratory system in preterm infants is an application of Bioelectrical and Computer Engineering. The microcomputer-based system is in general a tool used for the analysis of the respiratory physiology in clinical research.

The different signals obtained from the respiratory system can be classified as respiratory, chest wall motion, cardiac and muscular signals. This classification is based on the type of hardware configuration necessary for their measurement.

The personal computer on which the system is based, is equipped with special hardware and software, customized to the different needs of the research projects. The signals mentioned above are digitized and stored for post acquisition processing and analysis.

Analog to digital conversion is performed using a special high performance A/D circuit board installed as an expansion card and with a memory address assigned to it. The software application and development necessary to execute this process is performed using the best software technology available, which will allow us high sampling rates as well as accuracy. The software is capable of switching back and forth between monitoring and data storage status.

The signals collected are called raw signals, and contain noise contamination. Therefore it is necessary to process these signals in order to obtain a reliable representation of the parameters in which we are interested.

The most important part of this project is the signal processing. The design and software implementation of the different filters and signal processing methods are applications of digital signal processing theories and techniques.

The stages mentioned above would not have any valuable meaning without the correct interpretation, to provide the researcher with the information necessary to test hypothesis and reach conclusions. This is accomplished by the signal analysis stage which provides the interpretation.

The software necessary to accomplish the different procedures mentioned before was developed in C, which is a medium level programming language that provides numerous routines and functions that make it one of the most popular languages. There is also available a large number of libraries and object modules for different applications such as signal processing, math, graphics, device drivers, etc, that are commercially available.

This project is the result of a collaboration between Florida International University, Electrical Engineering Department, and University of Miami, School of Medicine, Department of Pediatrics, Division of Neonatology.

1.2 CLINICAL RESEARCH PROTOCOL

The subjects studied using this system are premature newborns, who are life supported at the Neonatal Intensive Care Unit at Jackson Memorial Hospital.

The study of respiratory physiology in premature newborns covers different areas of the respiratory system. The laboratory procedures include different respiratory tests applied to the patients. The signals are then collected by the computer system and the information about the physiological response to the test are obtained from the signal analysis. The research protocols evaluate central drive respiratory muscle activity and the mechanics of the system, individually or as they relate to each other by relatively non-invasive methods.

The analysis of ventilation and respiratory system mechanics are done on signals obtained by spirometry, to be described later on. The respiratory measurements determine respiratory timing, tidal volume,breathing frequency, lung compliance, pulmonary resistance and the ventilatory response to various gas mixtures. The information is obtained from the patient with appropriate measuring devices and transducers, whose electrical output represents air flow, esophageal pressure, mouth pressure and tidal volume, which is an integration of the air flow.

Another parameter of interest in the respiratory process is chest wall stability. Since the chest wall of the preterm infants has visible paradoxical movement during breathing, wherein chest motion is not always in phase with abdominal motion. It can decrease the efficiency of the respiratory pump.

In neonates chest wall stability can be examined by changes in the circumference of the thoraco-abdominal system at two different levels, Rib cage and Abdomen, using a pair of non-invasive inductance coils, one at each level.

Ventilation is achieved by the output of a central neural drive to the respiratory pump which consists of the diaphragm and accessory muscles. A specific interest of these project is analysis of the respiratory muscle activity, muscle parameters such as the magnitude of electrical activity, timing, muscle fatigue, and spectral analysis are related to the mechanisms described above.

The measurement of electrical activity in the muscles of respiration provides a mean for studying muscular activity. These electromyographic examination gives information about the magnitude of activity with peak and mean activity, activation and deactivation times and frequency components of the signal. There are several muscles participating in the ventilatory process, but this study is focused on five muscles: Diaphragm, upper airway muscles: Posterior Cricoarytenoid and Genioglossus, and accessory muscles: Intercostal and Abdominal, which are the major muscles for the respiratory process available for surface electrode measurements.

1.3 ELECTRICAL ENGINEERING IN BIOMEDICAL SCIENCES

Electrical engineering in past decades has become the area in which the technology has evolved faster than any other science. One important area of electrical engineering is Bioelectrical engineering, in which the analysis and processing of biological signals is the basic objective.

A signal is a mean to convey information. It is sometimes generated directly by the original information source, in which case information about the structure or function of the source can be directly obtained from the signal. However, when the signals available do not directly yield the required information, applying special processing procedures to the signals may derive the relevant information.

General measurement and diagnostic systems are developed in order to extract the desired information and convert it to a mode suitable for processing, storage, and analysis. In subsequent steps, the signals can be classified according to its characteristics and if needed, corrective measures may be taken.

The complexity of the biological system often introduces difficulties in the measurement and processing procedures of biosignals. The biological system cannot be uncoupled in such a way that subsystems can be monitored and investigated individually because of their control linkages and many feedback paths. The biological system under investigation must remain in its natural environment, so the signals produced by the system are influenced by surrounding systems and inherently contaminated with the noise produced by them.

The Biomedical signal instrumentation system must be designed so as not to interfere with the biological system. Thus, noninvasive techniques should be applied, or if this is not possible, the information must be inferred from signals noninvasively available.

Biomedical signals are mechanical, chemical, or electromagnetic in nature. These signals are presented for analysis as electrical signals by a variety of transducers.

1.4 DIGITAL SIGNAL PROCESSING, AN INTRODUCTION

In recent years, tremendous advances have been made in the area of digital technology. Information is now most conveniently recorded, transmitted and stored in digital form. As a result of this, digital signal processing has become an extremely important tool.

Digital signal processing (DSP), deals with the representation of signals as ordered sequences of numbers and the numeric processing of those sequences. It estimates characteristic signal parameters, and eliminates or reduces unwanted interference.

Classical Digital Signal Processing functions generally include: Digital filtering, Discrete Fourier transforms, Signal modulation, Autocorrelation and Cross-correlation, and some other techniques used for specific purposes.

DSP is applied in many areas of application, such as: Speech Signals Processing, Processing of Seismic Signals, RADAR, Image Processing, and in this case, Biomedical Signal Processing. New techniques are continuously developed in those areas, and in some cases, techniques developed for a specific area, such as RADAR, can find application in biosignal processing.

Some signal processing techniques described in this document were created specifically for this project, while others are modifications of already known techniques to make them suitable for the project's objectives and needs.

1.5 CHAPTERS ORGANIZATION

This document is divided into seven chapters and one appendix. The background material in Chapter One provides a brief description of the areas in which this project is involved, such as Bioelectrical engineering, Computer Engineering and Digital signal processing.

Chapter Two includes information about the physiological and bioelectric principles on which this project was designed and developed. It also presents the different biosignals, their origins and classification.

Chapter Three describes the system hardware, the computer system, data acquisition system, and the instrumentation system.

The system software such as the operating system, the programming language used for its development, the data acquisition software and some data file handling programs are presented in Chapter Four.

Chapters Five and Six present the software development, which is the principal objective of the project, which applies different digital signal processing techniques and algorithms to the sampled raw data, as well as provides further analysis of the processed signals. It presents a detailed description of programming techniques, file handling and graphics features.

Being this an open ended system, Chapter Seven presents several enhancements and recommendations, some of which are in actual development.

The appendix contains flow diagrams and example routines from the source codes.

CHAPTER II

PHYSIOLOGICAL PRINCIPLES AND BIOSIGNALS

2.1 RESPIRATORY PHYSIOLOGY

Physiology is the science of processes and functions of living biologic systems. Respiration is the process by which air is breathed, oxygen is extracted by the blood and delivered to the tissues, and carbon dioxide is purged from blood to lungs and then breathed out. Gas is brought to one side of the blood-gas interface by airways and blood to the other side by the vessels. The airways consist of a series of branching tubes which become narrower, shorter and more numerous as they penetrate into the lungs.

During inspiration, the volume of the thoracic cavity increases and air is drawn into the lung. This increase in volume results from the contraction of the diaphragm, which causes its descent and lowers the pressure within the thoracic cavity to more sub-atmospheric levels. The intercostal muscles also contribute to this function, by elevating the rib cage. At the end of inspiration the elastic lung returns passively to its pre-inspiratory resting position.

There are three basic elements controlling the respiratory system: Peripheral sensors that gather information, a central controller in the brain that coordinates the information and has a central rhythm generator, and the effectors (respiratory muscles) which act upon the airway and lungs to cause ventilation.

In the control of ventilation, the different muscles work in a coordinated manner, directed by the central controller. There is evidence that some premature children have uncoordinated respiratory muscle activity, especially during sleep [1].

2.2 BIOSIGNALS

The respiratory system biosignals which this project deals with are classified as: Muscular or Electromyograms (EMG), Respiratory, Chest wall motion, and Cardiac (EKG) signals. This classification is based on the instrumentation system setup necessary to obtain these signals, and on the signal processing and analysis software.

2.2.1 Respiratory Signals

Flow:[Liters/Min]

Flow is defined as the amount of gas that flows in and out the respiratory passages with each breath, in a specific length of time. It is divided into inspiratory and expiratory flow. Timing information derived from this signal, such as inspiratory time and total breath time is very useful for analysis.
Frequency range: dc to 50 Hz [1].

Tidal Volume: [ml]

This parameter measures the quantity of gas exchanged with each breath, and is defined as the simple integration of inspiratory and expiratory flow.
Frequency range: dc to 50 Hz [1].

Esophageal Pressure: [cm H_2O]

The esophagus is the feeding tube which extends from the pharynx to the stomach. The pressure changes occurring within the chest as a result of the descent of the diaphragm muscle during inspiration causes the lungs to expand, create negative pressure on the outside of the lungs, and draw air into the lungs. This negative pressure in the pleural space is transmitted to the esophagus and is used as parameter to measure driving pressure.
Frequency range: dc to 50 Hz [1].

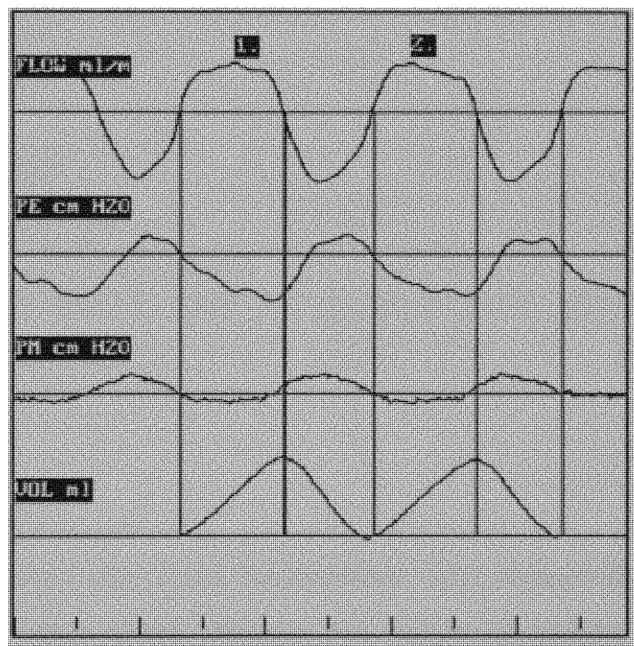
Mouth Pressure: [cm H2O]

The difference between the pressure in the pleural space, which is reflected in the esophageal pressure measurement, and the pressure at the mouth, which is the is the pressure lost in overcoming the resistance of external measuring devices, is the pressure required to move air through the air passages into the gas exchange units.

Frequency range: dc to 50 Hz [1].

Figure 2.2.1

This figure shows the respiratory signals (flow, esophageal pressure, mouth pressure and tidal volume), the lines mark points of onset of inspiration and beginning of expiration for each breath.



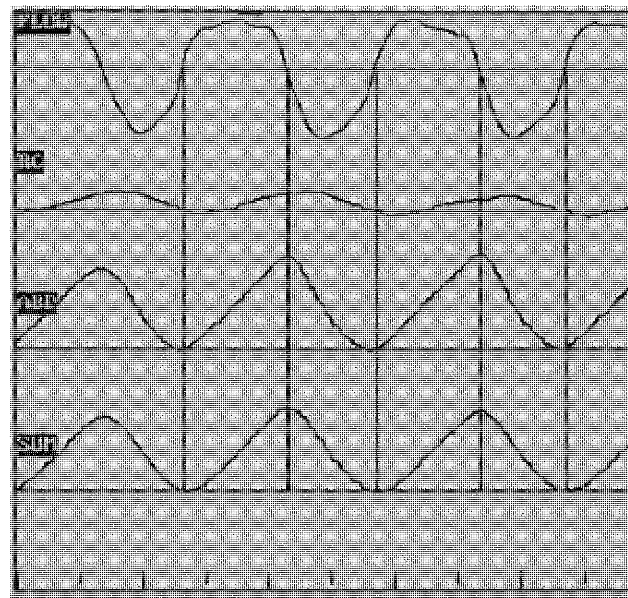
2.2.2 Chest Wall Motion Signals

The chest wall movement is observed at two levels, abdomen and rib cage. The transversal movement of the chest wall at these levels contains information about volume changes and timing.

Frequency range: dc to 5 Hz [1].

Figure 2.2.2

These are the ribcage and abdomen motion signals, and their sum. The lines indicate their time relation with the breath.



2.2.3 Muscular Signals

The skeletal muscle consists of cells with excitable membranes. The muscle is constructed from many separate fibers. These fibers contain two kinds of protein filaments, actin and myosin. These are arranged in parallel interlacing layers which can slide one into the other causing shortening of the muscle length. The sliding of the fibers is caused by chemical reactions.

The generation of motion or force by the muscle is activated when the fiber membrane is excited. There is an electrical potential across the surface membrane of a skeletal muscle fibre in the resting (polarized state), the interior is charged about 75 mV negative with respect to its surroundings. An action potential then propagates (depolarizing) along the surface membrane of the fiber triggering chemical reactions that, in turn, cause fiber contraction. When a muscle contracts, the action potentials generate an electric field that can be monitored by means of surface electrode. This field is a result of the contribution of many fibers at different times and with different rates. The EMG signal monitored this way will be a random signal with statistical properties that depend on the muscle function [5].

The neuron that activates the muscle is the motor nerve. The motor neuron-muscle connection is called neuromuscular junction or end plate. When the chemical substance that serves as a transmitter (Acetylcholine) is released from the neuron's axon endings, it diffuses toward the muscle membrane and is absorbed at the receptors sites, causing potential change in the muscle membrane. If the potential change crosses the threshold level, an action potential is generated and propagates along the muscle membrane. The process of transmitter release, diffusion and reception lasts 0.5 to 1 msec [2].

All the muscle fibers innervated by a single motor nerve fiber are called a motor unit. Usually muscle fibers of adjacent motor units overlap, allowing separate motor units to contract in support of each other rather than entirely as individual segments.

The current densities generated by membrane activity cause changes in the surrounding medium. The surrounding tissues, in which induced current changes occur, are called the volume conductor. In most applications, the fields of the volume conductor are monitored instead of the bioelectric source itself.

Muscle Fatigue:

Prolonged and strong contraction of a muscle leads to the state of muscle fatigue. This result simply from inability of the contractile and metabolic processes of the muscle fibers to continue supplying the same work output. The nerve continues to function properly, the nerve impulses pass normally through the neuromuscular junction into the muscle fiber, and even normal action potential spread over the muscle fibers, but the contraction becomes weaker because of depletion of energy supplies in the muscle fibers [6].

The clinical research protocol is interested in the activity of the muscles described below:

Diaphragm (DIA) EMG:

The diaphragm is the most important muscle of inspiration. It consists of a thin, dome-shaped sheet of muscle which is inserted into the lower ribs and divides the thoracic cavity from the abdomen. When it contracts, the abdominal contents are forced downward and forward, and the vertical dimension of the chest cavity is increased. In addition, the rib margins are lifted and moved out, causing an increase in the transverse diameter of the thorax. This movement creates negative pressure in the pleural space which acts on the outer surface of the lungs, drawing air into them [5].

Genioglossus (GGS) EMG:

This muscle is inserted to the Hyoid bone and into the body of the tongue, its respiratory function is to move forward the tong in order to keep the upper airway open.

Posterior Cricoarytenoid (PCA)EMG:

This is one of the muscles that abducts the vocal cords in synchronization with the activity of the diaphragm. It is an intrinsic muscle of the larynx and has a large role to play in lowering the resistance of the larynx when active. This is the site of the greatest amount of spontaneous fluctuation in resistance within the airways, and the present study is interested in its activity under stress condition.

Abdominal EMG:

The muscles of the abdominal wall are the most powerful expiratory and expulsive muscles of respiration. They are subject to a rhythmic respiratory activation in addition to their involvement in posture and other functions. During nose breathing their contribution becomes appreciable at ventilation slightly above quiet breathing. As the ventilation increases, the expiratory contribution of the abdominal muscles increases progressively [5].

Intercostal EMG:

The intercostal muscles, according to their name, are located between the ribs. The external intercostal muscle is purely inspiratory in its activity [5].

2.2.4 Cardiac Signals

As the impulse passes through the heart, electrical currents spread into the tissues surrounding the heart, and a small proportion of these spreads all the way to the surface of the body. The electrical potential generated by the heart is known as electrocardiogram (EKG). The electrocardiographic signal is composed by the occurrence of the PQRST complex. The P wave is caused by electrical currents generated as the atria depolarize prior to contraction. The QRS complex is caused by passage of the cardiac impulse through the ventricles. The T wave is caused by currents generated as the ventricles recover from the state of depolarization.

The electrical currents spreading through the tissues around the heart, influence the measurement of electrical activity in those tissues. In this project the cardiac artifact created by those currents in the respiratory muscle EMG is a non-desired contaminant that makes it necessary to use special processing techniques for its extraction for EMG signal validation. The cardiac or EKG signal is used as timing reference for these extraction procedure.

2.3 SUMMARY

The respiratory system analysis is based on the activity of specific muscles and some parameters of the breathing mechanics. Therefore, the biosignals used for analysis are classified as muscular , respiratory, chest wall motion, and cardiac signals.

The parameters of the respiratory mechanics are calculated from the respiratory signals group, which are: Flow, Tidal Volume, Esophageal Pressure and Mouth Pressure. The electrical muscle activity information of the different muscles participating in the respiratory process is related to these parameters.

2.4 REFERENCES

[1] Slonim N., Hamilton L. "Respiratory Physiology", Missouri, The C.V. Mosby Company, 1971.

[2] Cohen A. "Biomedical Signal Processing", Boca Raton, Florida, CRC Press, Inc.

[3] DuBovy J. "Introduction to Biomedical Electronics", Mc Graw Hill, Inc. 1978.

[4] Ruppel G., "Manual of Pulmonary Function Testing", Missouri, The C.V. Mosby Company, 1979.

[5] Campbell E., Agostoni E., Davis J., "The Respiratory Muscles, Mechanics and Neural Control", Philadelphia and London, W.B. Saunders Company, 1970.

CHAPTER III

SYSTEM HARDWARE

3.1 HARDWARE SYSTEM

A microcomputer system performs the functions required to obtain valid analysis of the different parameters. This microcomputer system includes different expansion devices and peripherals which provide the necessary capabilities to perform different tasks. The peripherals and external devices which are necessary to obtain information are an important part of the whole system. These devices are called the instrumentation system, and serve as sensor devices to convert physiological information into electrical signals.

As described before, the Hardware System is divided into three subsystems: Computer, Data Acquisition and Monitoring, and Instrumentation. Figure 3.1 is a complete block diagram of the Hardware setup.

3.2 MICRO COMPUTER SYSTEM HARDWARE

The system board is the primary part of the system and is sometimes called motherboard. This is a large printed circuit board that holds most of the main electronic parts. These include the processor and math coprocessor, as well as supporting electronics such as the clock chip. The system board is also the computer's basic complement of working memory and the read-only memory chips that hold the computer's built-in programs. The system's storage devices are the hard disk drive, the diskette drives and sometimes tape drives, that contain the application programs and data. The power supply contains basically a large transformer to lower the voltage and a fan that provides a cooling air flow. The system includes space for a number of optional parts called adapters. These cards plug into a row of sockets called expansion slots, representing the systems's open architecture.

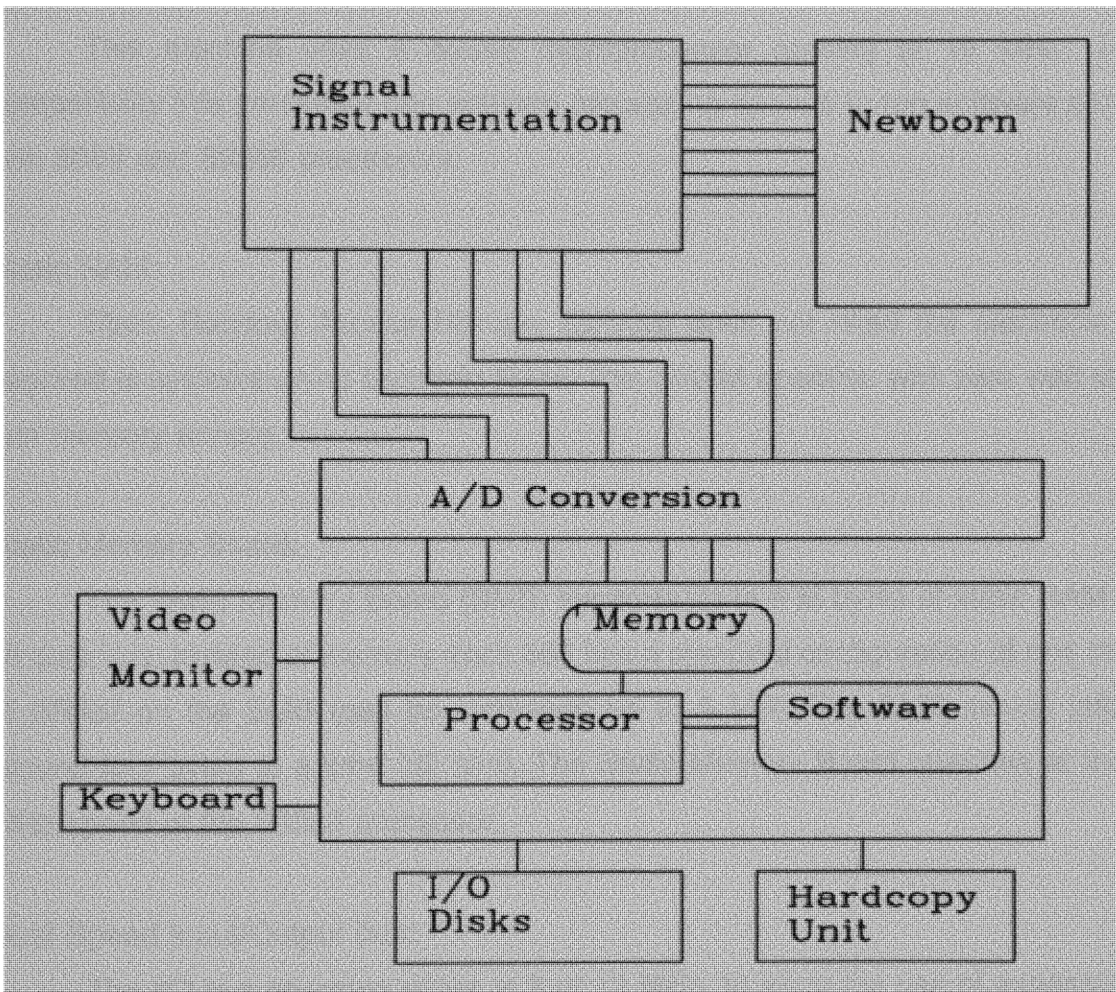


Figure 3.1 Block diagram of microcomputer based system

Our System, Hardware Configuration

The computer system used for this project is a PC's Limited 386 series machine from Dell Computer Corporation [1]. The following are some of the basic features:

Processor Speed:

This machine uses a 80386 microprocessor running at 16 MHz. To allow compatibility this processor can emulate processing speeds to run software compatible with 8088 and 80286 processors.

Static RAM:

Static Random Access Memory (SRAM) does not need the electronic refresh required is systems using Dynamic RAM. This SRAM operates with zero wait states for fast memory access.

Static Memory Board:

The Static Memory Board (SMB) supports up to 6 M bytes of static RAM on a 32 bit data bus. Instead of memory each 386 has a standard configuration of one megabyte of SRAM on the SMB (can be increased up to 5 MB). The SMB moves information around four bytes at a time (32 bits). The Static Memory Board operates at the full 16 MHz speed of the 80386 processor.

Dual Speed Input/Output bus:

The I/O bus operates either at 8 or 12 MHz, to allow the use of expansion cards available for 80286 systems, without sacrifying the microprocessor's power and speed.

Expansion Slots:

This machine provides 7 expansion slots, for modems, video cards, multiple I/O ports.

Video Display:

The system has an Enhanced Graphics Adapter (EGA) card, and comes with FAST EGA mode enabled. When this mode is enabled, it takes much less time for the system to update the screen.

Storage System:

The system includes a 1.2 megabytes floppy disk drive and a Hard drive provides high speed, high capacity storage for programs and information. The hard drive is also known as fixed disk, hard disk or Winchester drive. Our system has a 152 MB type 31 hard drive, with two partitions, drives C and D. Drive D contains most of the capacity and contains the working directories.

Tape Drive:

The size of the data files makes it necessary to use of a 40 megabytes tape drive used for large back up of large amounts of data, freeing disk space for new data.

Hardcopy Device:

The hardcopy device is a Fujitso DL2400 printer. It is a multi-purpose, 24-wire dot matrix impact printer. It is compatible with most of the IBM Graphics printer and Epson FX 80 codes. Prints up to 216 characters per second and it is connected at the first parallel port.

3.3 DATA ACQUISITION AND MONITORING SYSTEM

A/D Conversion

An analog to digital converter changes continuously-varying analog signal into an analog data value (a digital code of 0's and 1's), which is intelligible to a computer.

DT2821 BOARD

The DT2821 board (Data Translation, Inc) is a high speed analog and digital I/O board designed to be used with a personal computer. This board is plugged into one of the fully bussed expansion slots [2].

The board can be programmed to perform analog to digital (A/D) conversions; digital to analog (D/A) conversions; and digital input and output (DIO) transfers. The DT2821 series board can be configured to perform the analog I/O functions in programmed I/O (PI/O) mode or direct memory access mode (DMA), both with or without interrupt capability. The DMA interface is compatible with 16 bit data transfers and can be selected to use DMA channel 5, 6, or 7. DMA buffers may be located anywhere in the memory space of the computer and maybe up to 128 K bytes. Two DMA channels can be configured to support continuous performance DMA, which is a data sampling method that provides gap-free transfers of large volumes of data from memory or disk (D/A) or to memory or disk (A/D) without any loss of samples.

It also contains an on-board programmable pacer clock, which can be used to provide clock pulses to control the rate of conversions for the board's A/D and D/A subsystems.

This board is connected to the signal conditioning stages through the DT707 screw terminal, which is set up in a metallic box with BNC input connectors.

Base Address:

the base address is the lower I/O address location used by the DT2821 board. The base address can be set anywhere between HEX 200 and HEX 3E0 in increments of 20 (hex). The board is actually set to the address HEX 240.

Overlap Mode A/D Operation:

The DT2821 board A/D conversion subsystem is shown in the block diagram of figure 3.3.2.

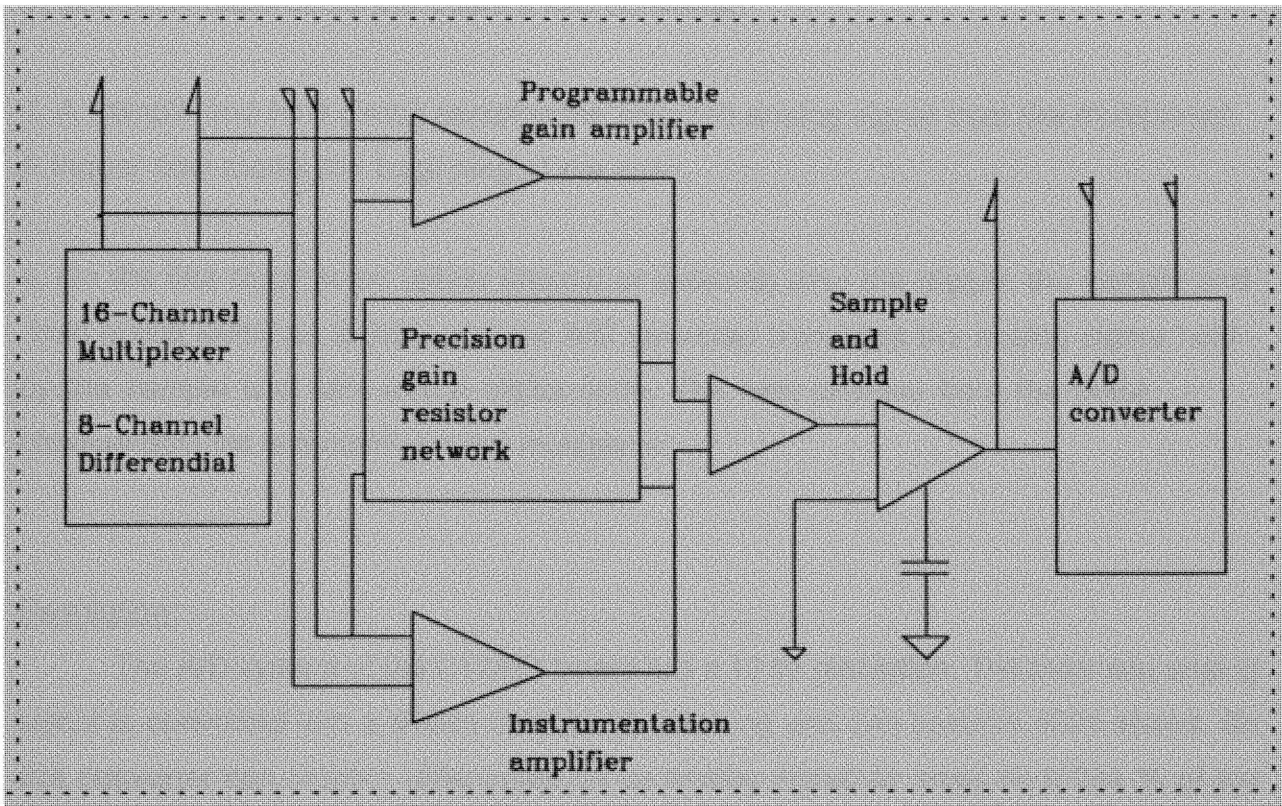


Figure 3.3.2 DT2821 board A/D conversion subsystem

This A/D subsystem of the data acquisition board DT2821 performs its functions in the following way:

1. The analog multiplexer chooses one input channel from those connected to the board. These channel addresses are preloaded into the RAM channel-gain list and the

first channel is applied to the multiplexer's address lines.

2. The programmable gain amplifier buffers the analog input, and may increase its voltage level based on the associated gain bits. Since the analog input signal from the transducer may be only +1.25 v maximum, an amplifier boosts the signal (by gain of 8) to a level required by the board's A/D converter.

3. A sample and hold circuit samples the selected analog signal from the multiplexer. A trigger now marks the beginning of the scan.

4. On the first clock pulse (either internal or external) after the trigger, the sample and hold circuit switches to hold and the A/D converter translates the analog signal held by the sample and hold into a digital code. The next channel from the RAM channel-gain list is loaded into the multiplexer.

Resolution:

The A/D system of the DT2821 board has a 12 bit resolution. The incoming analog signal is converted into a binary number 12 bits long and can assume 4096 states. So this converter can resolve differences on an analog signal as small as 0.024 % of the selected analog input range.

Video Monitoring: WFS-200PC Waveform Scroller Card

The WFS-200PC (DATAQ Instruments, Inc) waveform scroller circuit board is a high performance waveform graphics interface [3]. It makes it possible real time data display during data acquisition, and also allows post-acquisition data playback.

This card may display waveforms on several different monitors. It also supports the IBM Enhanced Graphics Adaptor (EGA) card used in this system. For this application, this circuit board was preset to the hardware address HEX 308.

3.4 INSTRUMENTATION SYSTEM AND SIGNAL CONDITIONING SYSTEM

The biosignals instrumentation setup follows noninvasive procedures, which not to interfere with the biological mechanisms of the respiration. The infants studied are asleep and relatively familiar with the respiratory therapy and monitoring equipment used in the intensive care unit. Thus, the instrumentation system is relatively a non-invasive system.

The instrumentation system, and most of the subsystems in this project, are divided by hardware configuration into respiratory, chest wall motion, muscular (EMG), and cardiac (EKG) instrumentation systems (see figure 3.4).

3.4.1 Respiratory Signals Instrumentation System

Flow Signal:

The air flow is measured by a Fleisch "00" pneumotach attached to the nostrils by nasal prongs. The pneumotach is a laminar device used to measure the difference in pressure created by the gas flowing through a wide bore tube of fixed length and radius. The pressure difference is proportional to the flow and is applied at the two pressure ports of a variable reluctance pressure transducer (Valydine MP45).

The variable reluctance pressure transducer translates the difference in pressure to change in reluctance. Reluctance in the ratio of magnetomotive force to the total magnetic flux. The deflection of the diaphragm due to change in pressure increases the gap in the magnetic flux path of one coil, and causes an equal decrease in the other. The reluctance varies with the gap, determining the inductance value.

The output of the variable reluctance pressure transducer is connected to a miniature carrier demodulator (Validyne CD 316), that converts the change in inductance to an output compatible with strain gage systems. The flow signal is then conditioned using a transducer amplifier (Gould 13-461550)[6]. This transducer amplifier is a single channel, direct coupled, plug-in dc bridge amplifier. It is capable of measuring strain gage based transducers, resistance temperature devices, and low level dc input signals.

Mouth and Esophageal Pressure:

The mouth or proximal airway pressure is measured through an air-filled catheter attached to the side port of the adapter (nosepiece) in the nasal prongs, while the esophageal pressure is measured by a wide bore water filled catheter in the lower third of the esophagus. These catheters conduct pressure to a hydrostatic fluid pressure transducer P23XL (Spectramed Inc.) [4] and have a flat frequency response in the band of normal breathing frequency of infants. The most important part of this transducer is a silicon chip linked to a metal diaphragm, onto which the strain elements of a Wheatstone bridge are diffused. When the diaphragm is deflected the silicon chip is stressed, causing an unbalance in the resistance of the bridge. This unbalance causes a proportional electrical output which is sent for signal conditioning to the transducer amplifier described above.

3.4.2 Chest Wall Motion Signals Instrumentation System

Chest wall motion is determined using respiratory inductance plethysmography (Respitrace Corp.)[8], with closely fitting elastic bands, one band placed around the chest at the level of the ribcage and the other around the abdomen at the level of the umbilicus.

The device consists of two coils of Teflon-insulated wire which zig-zag around the bands mentioned before. The respiratory movements change the inductance of each coil proportionally to the change in volume. The bands are connected to an oscillator module that provides approximately 20 m pp amplitude sine wave with a frequency of 300 KHz [8].

This change in inductance is converted into a proportional direct current voltage that can be amplified and recorded. The output signals from this device is conditioned by a medium gain dc preamplifier (Gould 13-4615-00)[6]. This is a balanced common medium gain preamplifier, with maximum sensitivity of 50 m full scale.

3.4.3 Muscular Signals Instrumentation System

The respiratory EMG signals are detected by silver-silver chloride surface electrodes. Three electrodes are attached to the infant for each muscle studied; two of them are placed in the area of interest, and the third one, acting as a ground reference is placed over a bony prominence, such as forehead or ankle. These three electrode leads are connected to a fiber optic isolated differential bioamplifier and transmitter (Coulburn S75-04B) [5], capable of carrying biopotential signals up to 1 Khz. Since this transmitter is isolated from the system ground, it has a very high common mode rejection ratio. The fiber optic cable driver uses an infrared light emitting diode (LED) to send the modulated signal through the fiber optic cable. At the end of this cable the signal is picked up by a fiber optic receiver (Coulburn S75-04)[5]. This receiver reconstructs the frequency modulated signals and conditions them for amplitude, providing an output level of 1 volt RMS [5].

The EMG signals are then band pass limited using an adjustable band pass filter (Coulburn S75-36) with 48 dB/octave roll off [5]. The signals are band limited between 10 Hz and 500 Hz. These band limits have been determined by PSD analysis which shows that more than 95 % of the signal lies below 500 Hz, while below 10 Hz the signals are affected by low frequency noise.

The only muscle instrumentation set up that is slightly different is for the Posterior Cricoarytenoid or PCA; it is necessary to use a special ring electrode, which is a thin catheter conducting two wires connected to a couple of silver rings at the end. The catheter is introduced into the upper airway, where the PCA muscle is located.

3.4.4 Cardiac Signal Instrumentation System

The electrocardiogram (EKG) is measured using the same type of surface electrodes as used for EMG signal measurements. Three leads are placed in a special configuration over the upper chest and connected to an isolated ECG/Biotach amplifier (Gould 13-4615-65)[6]. This amplifier is a multifunction signal conditioner with both isolated EKG and biological rate measurement capabilities.

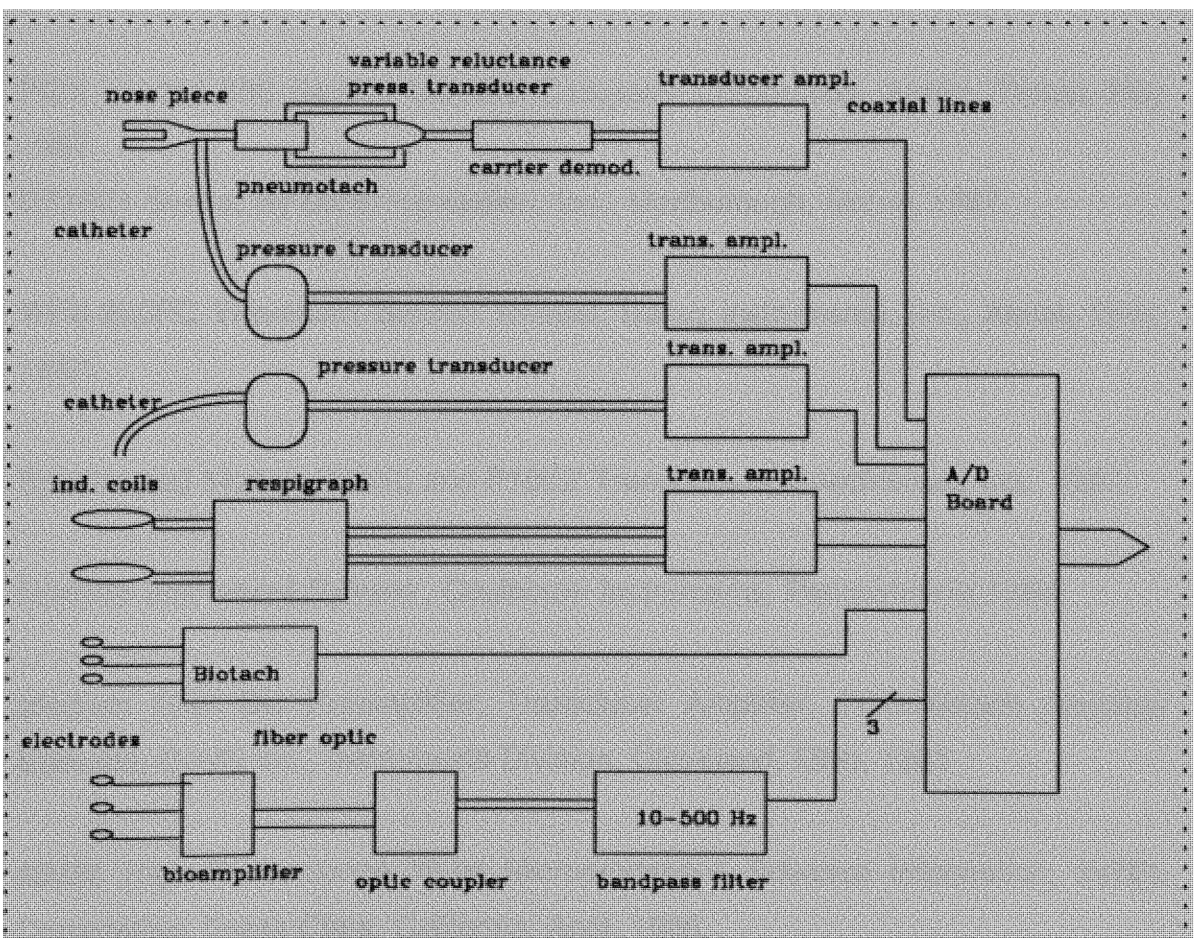


Figure 3.4 Instrumentation system diagram.

3.5 SUMMARY

The base of the system is its hardware configuration. The microcomputer digitizes the signals coming from the instrumentation system which is divided into respiratory, muscular (EMG), chestwall distortion and cardiac signals subsystems.

The signals are digitized using an A/D board inserted in one of the expansion slots of the personal computer and the data is stored in the hard disk storage system. During data acquisition the signals can be video monitored using a special card that provides real time data displaying.

3.6 REFERENCES

[1] PC's Limited 386 Series, The Dell Computer Corporation, Austin, Texas, May 1987.

[2] DT2821 Series, By Data Translation, Inc., Marlborough, Maine, 1987.

[3] WFS-200PC Waveform Scroller Manual, Dataq Instruments, Inc., Akron, Ohio, 1987.

[4] Spectramed Models P23XL and P10EZ Transducers, Spectramed, Inc., 1987.

[5] Modular Instrument System , Coulbourn Instruments, Pennsylvania 1984.

[6] Medium Gain DC Preamplifier, Transducer Amplifier, ECG/Biotach Amplifier Manuals, Gould Inc., Instruments Division, Cleveland, Ohio, 1981

[7] 80386 High Performance 32-bit Microprocessor, Advance Information, INTEL Corporation, Santa Clara, California, April, 1986.

[8] RESPIGRAPH tm, NIMS, Inc. , Miami Beach, Florida.

CHAPTER IV

SYSTEM SOFTWARE

4.1 INTRODUCTION

The system software is based on the MS-DOS operating system. Different application programs have been created and some commercial programs have been adapted to perform the tasks necessary to transform the microcomputer system into a powerful tool for the clinical research.

This chapter presents a description of the operating system that manages the hardware resources of the computer. It also introduces the C language for a better understanding of the software implementations described in this and following chapters.

The data acquisition software drives the hardware interface to the hardware instrumentation system as well as to the computer resources.

4.2 MS-DOS OPERATING SYSTEM

The Disk Operating System (DOS) belongs to a class of computer programs that are known as supervisors, control programs, or operating systems. The task of an operating system is basically to supervise and direct the work, the operation of the computer.

MS-DOS is partitioned into several layers that serve to isolate the kernel logic of the operating system, and the user's perception of the system, from the hardware it is running on. These layers are:

. BIOS (Basic I/O System)

. DOS Kernel

. The command processor (Shell) [1]

BIOS Module:

The BIOS contains the default resident hardware-dependent drivers for the following devices:

. Console display and keyboard (CON)

. Line printer (PRN)

. Auxiliary device (AUX)

. Date and time

. Boot disk device (block device)

DOS Kernel:

The kernel is a proprietary program that provides a collection of hardware independent services called system functions. These functions include the following:

. File and record management

. Memory management

. Character-device I/O

. Spawning of other programs

. Access to the real-time clock

Programs can access system functions by loading registers with function specific parameters and then transferring to the operating system by means of a software interrupt.

Command Processor:

The command processor or shell is the user's interface to the operating system. It is responsible for parsing and carrying out user commands, including the loading and execution of other programs from disk. The default shell provided with MS-DOS is found in a file called COMMAND.COM, which is a special class of program running under the control of MS-DOS.

Loading MS-DOS:

When the system is started, program execution begins at address 0ffff0H. This is a feature of the 80x86 family of microprocessors. Systems based on these processors are designed so that this address lies within an area of ROM and contains a jump machine instruction to transfer control to system test code and ROM bootstrap routine.

The ROM bootstrap routine reads the disk bootstrap routine from the boot sector of the disk into memory and transfers control to it. The disk bootstrap routine checks to see if the disk contains a copy of MS-DOS (reads the root directory looking for IO.SYS and MSDOS.SYS files); then the disk bootstrap reads them into memory and transfers control to IO.SYS. The IO.SYS file that is loaded consists of two modules: BIOS and SYSINT.

SYSINT is called by the BIOS initialization code. It determines the amount of contiguous memory present in the system and then relocates itself to high memory. Then it locates the DOS kernel to its final memory location and calls the initialization code in MSDOS.SYS.

The DOS kernel initializes its internal tables and work areas, sets up the interrupt vectors, and traces through the linked list of resident device drivers and initializes them. At this point SYSINT calls the normal MS-DOS file services to open the CONFIG.SYS file, which contains commands that enable the user to customize the MS-DOS environment.

The CONFIG.SYS file is loaded into memory and its commands are for processed, memory is allocated for the disk buffer cache and the internal file control blocks used by the handle file and record system functions. Any device drivers indicated in the

CONFIG.SYS file are sequentially loaded, initialized and linked to the device-driver list. Then SYSINT closes all file handles and reopens the console, printer and auxiliary devices as standard devices. Finally, SYSINT calls the function EXEC to load the command interpreter (COMMAND.COM as default).

EXEC Function:

Both types of programs are loaded in memory for execution by the EXEC function, that is called by COMMAND.COM with the filename of a program. It can also be called by other shells, or by another program that was previously loaded.

EXEC allocates a block of memory to hold the new program, builds the program segment prefix (PSP) at its base, and then reads the program into memory above the PSP. Then it sets up the segment registers and the stack and transfers control to the program.

Program Segment Prefix:

The program segment prefix is a reserved area, 256 bytes long, that is setup by DOS at the base of the memory block allocated to a transient program. The PSP contains linkages to MS-DOS that can be used by the transient program, some information MS-DOS saves for its own purposes, and some information MS-DOS passes to the program.

Loading .EXE Programs:

The MS-DOS loader always brings a .EXE program into memory immediately above the program segment prefix, although the order of the code, data and stack segments may vary. The .EXE file has a header, which is a block of control information with a characteristic format. The size of this header varies with the number of instructions that need to be allocated at load time (multiple of 512 bytes).

Before the MS-DOS transfers control to the program, the initial values of the code segment register (CS) and instruction pointer register (IP) are calculated from the information in the file header and the loading address. This information derives in the source code for one of the program's modules. The data segment register (DS) and the extra segment register (ES), are made to point to the PSP so the program accesses the environment-block pointer, command tail and other information. The initial contents of the stack segment register (SS) and stack pointer register (SP) come from the header. This information is obtained from the declaration of a segment with the attribute STACK in the program's code. When a .EXE program finishes processing, it returns control to MS-DOS through Int 21H Function 4CH.

Custom System Configuration and Initialization Files:

The computer system used for this project is configured using the CONFIG.SYS that contains the following configuration commands.

. SHELL sets the MS-DOS command processor to the COMMAND.COM located in the directory c:\DOS\. e:nnnn specifies the environment size. /p keeps the secondary command processor in memory and does not automatically return to the primary command processor.

. DEVICE sets a search path to find the device driver being added to the system. The CONFIG.SYS file in this system installs the following devices: ANSI.SYS, allows you use ANSI escape sequences in real mode. VDISK allowing you to create a virtual disk (area of virtual memory that is used to emulate real disk). IBMCGI, IBMEGA and IBMGPR (GSS GRAPHICS) are device independent interface to graphics devices.

. Sets the number of buffers=25 and files=25.

. BREAK=on sets the Ctrl-C check.

The AUTOEXEC.BAT file sets the PATH, that tells MS-DOS where to find external commands. The SET command sets one string of characters in the environment equal to another string for later use in programs.

4.3 DEVELOPING SOFTWARE IN C LANGUAGE

In recent years C has become the most useful programming language and almost an industry standard. Because of that, a large number of software libraries, hardware drivers and object modules are available for application in different areas such as engineering, graphics, etc.

The C language is an ideal language for software development. It offers a selection of data types and control structures while handling additional tasks like I/O, graphics, math calculations and the handling of peripheral devices by additional libraries. Being a medium level language, it can be interfaced with different programming languages such as FORTRAN, PASCAL, BASIC, and ASSEMBLY.

The software developed for this project was implemented using a Microsoft C compiler, Version 5.1 [3]. The following is a brief introduction to the C language for a better understanding of the software implemented for this project.

A C source file consists of preprocessor directives, declarations of variables, a main function and other functions. Each function contains expressions and statements.

Preprocessor Directives:

This unique facility enable users to design programs that are modular, readable and easy to use in different computer systems. These directives provide three important features: include contents of a file into a program (file inclusion), replacing one string with another (token replacement and macro processing), and compiling selected portions of a program (conditional compilation).

Process Control:

One way to reduce software development time is to use existing source code. This can be accomplished using function calls and even call routines written in another languages. In case the existing code is an executable file, the C library provides the facilities that allow another program to be executed from your program (child process) or spawning.

A process is an executable program in memory and its associated environment. Anytime a program is run, a process is created. The program's environment is stored in the PSP (Program Segment Prefix) and it includes all the information needed to execute a program. This includes information about the location of its code and data in memory, and record of files opened during execution.

Environment of a Process:

When MS-DOS is running, the user talks with a process running under COMMAND.COM control. In this case the environment includes the variables PATH, COMSPEC, PROMPT, INCLUDE, LIB, etc.
The environment variables are used to pass information to processes. C provides the capacity to access environment variables using the library routines GETENV and PUTENV, used to obtain the environment variable and alter it respectively. All the changes made during the program to the environment will vanish because the program environment is only a copy of the parent environment.

Command Line Arguments:

When the command line PROGRAM ONE TWO THREE is typed at the DOS prompt, one two three are called command line arguments. In this command line, PROGRAM has three arguments, the first one is an integer ARGC, containing the number of command line arguments, being the first argument the full pathname of the program. The argument ARGV is a pointer to an array of C strings, each containing one command line argument. The environment is passed to a process in the same manner as the command line arguments, ENVP is a pointer to an array of null-terminated strings, each containing one environment setting.

Signals:

Signals are the ways the operating system interrupts a process when error conditions (exceptions) occur. Each recognized exception has a routine to handle the exception, and the library routine signal can be used to handle a particular signal, when a signal occurs the appropriated handler is called. The raise function generates a signal artificially.

Memory Layout:

C enables the user to request blocks of memory at runtime and release the blocks when your program no longer needs them. Thus it is possible to design an application to exploit all available memory in the system. This capabilities come in the form of a set of library routines, called memory allocation routines.

File Handles:

The pathname of a file is not the only way to identify a file. When a file is created or opened using the functions OPEN, SOPEN or CREAT, an integer identifier is returned, this is called "handle" of the file. The handle is used by the system to access a structure where certain pertinent information about the file is stored.

Input and Output Routines:

Input and output can involve reading from and writing to files in the disk or reading input from the keyboard, sending output to the display screen or commands and information to peripherals.

The C library supports three types of I/O: Stream routines, low level file I/O routines, and console-port I/O routines. The stream routines refer to I/O performed using the model of files as a stream of bytes together with a buffer associated with a file. The buffer is a temporary storage area for the stream of bytes being read from or written to the file. The low level routines are similar but they do not use the buffer. Console and port I/O is the direct input and output to keyboard, monitor, and peripherals.

System Calls:

The C library includes a set of functions that provide access to the BIOS and DOS services from the programs. These functions help to obtain the full potential without having to write in most cases, any assembly code. The assembly language instruction INT generates an interrupt on a microprocessor, providing access to the BIOS and DOS function object codes, no matter where they reside.

C Software Interrupts:

C provides the routines INT86 and INT86X for generating arbitrary software interrupts. These routines accept the register settings in a union named REGS, which is defined in the include file DOS.H and is the overlay of two structures, one named x of type WORDREGS and the other named h of type BYTEREGS. The member x provides access to the 16-bit word registers AX, BX, CX and DX, while the member h is used for accessing the 8-bit halves AH, AL, BH, BL, CH, CL, DH and DL. The segment registers ES, CS, SS and DS are passed via the structure names SREGS, also defined in DOS.H.

C Interface to DOS and BIOS:

The interface routine in the C library loads the registers from these data structures before generating the software interrupt necessary to access the desired BIOS or DOS service. Upon the return from the interrupt, the interface routine copies the register values into another structure which is allocated and whose address is passed to the routine, This allows the program to obtain the results or error codes returned.

4.4 DATA ACQUISITION SOFTWARE

4.4.1 Introduction

Data acquisition is used to interface between the real world of physical parameters and the "artificial" world of digital computation and control. Sensors and transducers generate voltages that vary in proportion to the physical properties they measure. The data acquisition software is one of the most important pieces in the project. This software drives the data acquisition and storage systems at rates that will allow it to obtain accurate information without signal distortion due to frequency aliasing.

There are two type of data acquisition required: Short term (1 minute) and long term (several minutes). To accomplish these tasks, two types of software are available for each type of data acquisition: CODAS (DATAQ Instruments) [4] and GETDATA.PRG running under ASYST Scientific System [5].

4.4.2 Data Acquisition Setup

The data acquisition system configuration obtains the maximum performance possible from the hardware resources. The CPU speed (16 MHz) limits the total sampling rate to a maximum of 7000 samples per second. This factor reduces the capacity of the system to seven channels for data collection, with the following configuration:

CHANNEL	SIGNAL
1	Flow (FLO)
2	Esophageal Pressure (ESO) or Rib Cage Movement (RIB)
3	Mouth Pressure (PMM) or Abdomen Movement (ABD)
4	EMG signal #1, Diaphragm (DIA) or Abdomen (ABM)
5	EMG signal #2, Posterior Cricoarytenoid (PCA) or Intercostal Lateral (INT)
6	EMG signal #3, Genioglossus (GGS) or Intercostal Parasternal (PAS)
7	Electrocardiogram (EKG)

This configuration can be altered according to the clinical research objectives.

4.4.3 Sampling Rate

Respiratory signals have very low frequency spectrum (50 Hz maximum), while the muscular signals have frequency components up to 500 Hz. According to the Nyquist theorem, the bandwidth of the EMG signals requires the sampling rate to be set at least twice the bandwidth to avoid aliasing effect.

$fs = 2*fm$ where fs is the sampling rate per signal, and fm is the bandwidth of the signals.

The sampling rate chosen was 7000 Hz divided by the number of channels (7), yielding a sampling rate per channel of 1 KHz.

4.4.4 CODAS Data Acquisition Software

CODAS is a commercially available engineering software package by Dataq Instruments, Inc [4]. This package was customized to the system setting up the size of the system data buffers, the dual mode overlapping A/D conversion in the DT2821 board, for maximum performance of the board and system in terms of speed, in order to obtain the sampling frequency required with real time displaying of the signals.

This package is used for short term or a predetermined data acquisition time. It requires file sizes of 860 KB for one minute collection. CODAS provides a variety features such as:

. Real time displaying (generated by the WFS-200pc waveform scroller board).
. Control over data acquisition parameters such as gain, sampling rate, and number of channels enabled.

The CODAS software is used for real-time signal display during data collection, which allows the monitoring of the signal quality, noise levels, breathing pattern and movement of the patient, for an interactive data acquisition process.

CODAS Data File Architecture:

The CODAS data file storage format is presented in figure 4.4.5-a. It consists basically of a header and the data itself. The header contains information such as the number of channels enabled, sampling rates, gains and analog to digital conversion full scale code. The header is 578 words long (each word is 2 bytes) [4].

The data is placed immediately after the header. The time multiplexed data stream of seven channels coming from the A/D board is stored in 2 bytes words that contain the value in two's complement binary format, where the fifth bit (D4) is the least significant bit. Bit D0 is channel 0 marker and is set to 1. See figure 4.4.5-b [4].

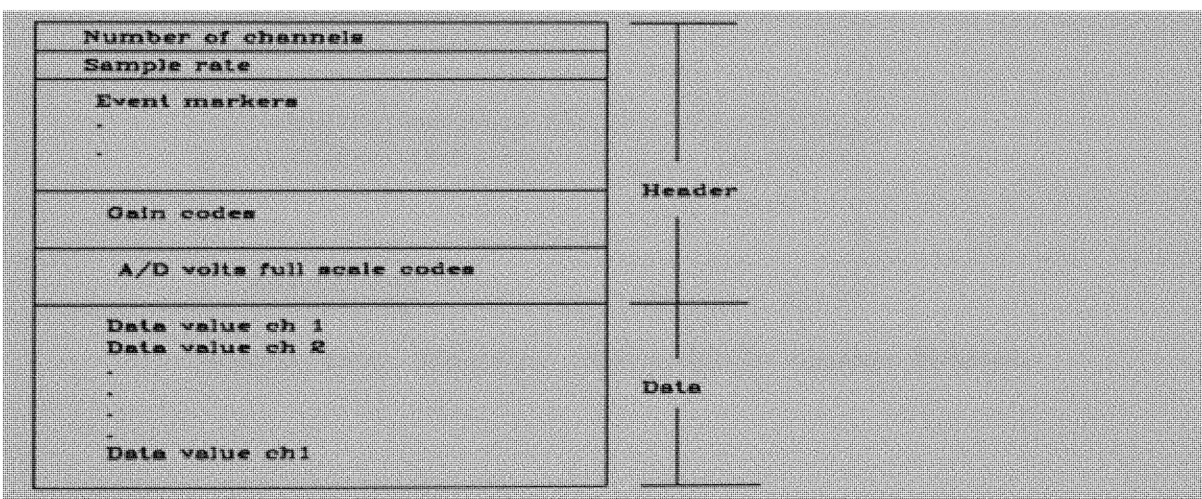


Figure 4.4.5-a CODAS data file storage format

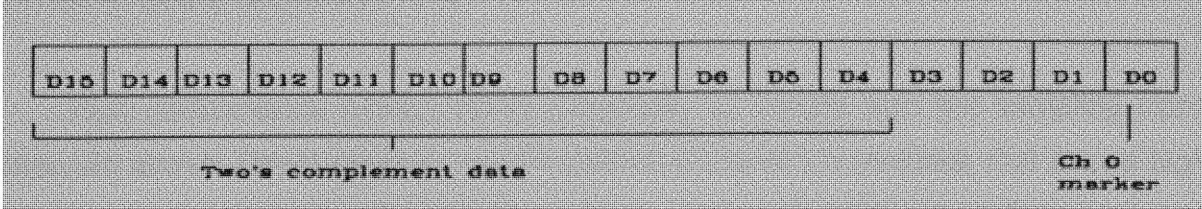


Figure 4.4.5-b CODAS data format

4.4.5 ASYST Scientific System

ASYST has been designed by Adaptable Laboratory Software, Inc.[5]. It incorporates features of standard computer languages, and it has its own application environment. It contains prewritten software tools which can be used by themselves or incorporated into the application programs. ASYST data acquisition modules supply tools to input analog information from instruments.

ASYST Data Storage Format:

ASYST represents data in a bipolar range using offset binary configuration. The A/D system input supports signals within the range of +/- 10 volts and the output range is from 0 to 4096. The stream of data written in the destination file, contains the multiplexed sampled values of each channel as shown in figure 4.4.6.

A/D data value ch 1
A/D data value ch 2
. .
A/D data value ch N
A/D data value ch 1
. . . End of File

Figure 4.4.6 Program GETDATA.PRG storage format.

Concept of Direct Memory Access:

High performance analog I/O is necessary to comply with the sampling rates mentioned above. Direct memory access (DMA) is one of the methods that allows high speeds storing large amounts of data directly to disk with minimum software support [5][1].

Direct memory access is a process which allows acquisition to take place independent of the computer's processor. This frees the processor to perform other duties while the analog input or output takes place. DMA is a method for moving data from the data acquisition system directly into memory using hardware capabilities built into the PC. The important point here is that the data movement is accomplished entirely through hardware; no software intervention is require to transfer the data.

The personal computer has seven DMA channels. It uses channel 0 for memory refresh, channel 2 for floppy disk support, channel 3 for hard disk support; this leaves four channels for DMA transfers. The DT2821 board is configured to use dual DMA using channels 5 and 6.

Program: GETDATA.PRG (appendix A.1)

This program collects seven channels using direct memory access (DMA). The sampling rate is set to 7000 Hz (1 kHz per channel) and it runs under ASYST system environment. This program allows the signal acquisition time limited only by the storage device. The rate of acquisition system writes 840 k bytes in the hard disk per minute.

ASYST supports the data acquisition hardware with a special configuration called "template", that contains information about the number of channels enabled, and the type of data transfers.

The following are the routines executed to accomplish the data acquisition:

1. Once the ASYST system is installed, the instruction LOAD GETDATA.PRG loads the program automatically on to the system. The variables and the data array used for DMA transfers are declared. Next, the A/D template is defined, with information about the A/D board, number of channels, name of data array, and the type of DMA transfer.
2. The keyboard buffer is reset and the function keys F1 and F2 are assigned to acquisition and exit functions respectively, displaying the options to the user.

3. Once the key F1 is pressed, the destination file is opened by the routine MAKE.FILE, that checks for file existence.

4. Then the routine START.ACQ takes control. The A/D system is initialized, sampling rate is set, and the data acquisition is started.

5. The routine COLLECT controls the data acquisition procedure. The data buffer array is divided in two parts, one half is being filled while the other half is written into the destination file. The routine checks for A/D overruns, with the relation between the BUFFER.INDEX and the time elapsed. This process continues until any key is pressed, interrupting the acquisition.

6. Once the acquisition is stopped, the file is closed, and the system is reset and the user may continue recording in a different file or exit the program.

4.5 FILE FORMAT TRANSLATION SOFTWARE

After the data acquisition procedure has finished, the data files containing the signals are prepared for the next stages, which are signal processing and analysis procedures. The data file handling software was developed to customize the different data storage formats to the project's special needs.

The data acquisition system provides the system with two types of data file formats, depending on the software used for that purpose. Thus, two separate programs are necessary for data format translation: IMPORT.EXE and RETRIEVE.EXE. These programs are also used to import files from the directories used for data acquisition into the current working directory.

Program: IMPORT.EXE (appendix A.2)

This program converts the data file collected with CODAS data acquisition software to binary format without the file header. This conversion is necessary because of the special data storage format that CODAS uses for its data files. As explained before, CODAS stores data values in a two byte word, where the value is stored in two's complement format, with its LSB value at the bit D4 of the first byte.

IMPORT.EXE performs the data conversion and imports the files containing seven signals from its original directory to a working directory (with the extension .DAT), and after this is done, it spawns separate processes to create a file containing the tidal volume signal, which is obtained is the digital integration of flow. These processes are: ONE-CH.EXE, INTEG.EXE and ONETO8.EXE. They extract the flow signal out of the seven signals file, calculate the tidal volume signal, and then create a destination file with the tidal volume signal incorporated, respectively.

Spawning a process is the execution of a child process, either by destroying the parent process in memory or leaving it intact and returning to it when the child terminates. The child receives a copy of the parents environment. (SPAWNL routine).

IMPORT.EXE performs the following tasks during execution:

1. Checks if there is enough disk space for the destination file. To do this, it uses the _DOS_GETDISKFREE system call.

2. If there is enough disk space, it opens the original and the destination files, for buffered input/output operation, with the directive FOPEN, which assigns it to the dat_file_hdl structure of the type FILE.

3. The process gets into a loop where a buffer is read and each value is rotated bitwise 4 bits to the right. It continues until the signals are detected to be zero volts during a

certain interval, or the end of file is reached, and then, both files are closed.

4. IMPORT.EXE spawns a child process called ONE-CH.EXE, that extracts the flow signal and stores it in a temporary working file.

5. INTEG.EXE is a second process spawned by IMPORT.EXE. This program creates a second temporary file containing the digital integration of the flow signal (Tidal Volume signal).

5. The Tidal Volume signal is incorporated into the seven channels file, by the spawned process ONETO8.EXE, that creates a file that contains eight signals.

Program: RETRIEVE.EXE

The GETDATA.PRG data acquisition program running under ASYST Scientific System, stores data in offset binary format, with a 12 bit resolution. It means that there is an offset of 2048 added to each number, to represent positive and negative voltage values with numbers from 0 to 4096.

The program executes the same steps as IMPORT.EXE, except step # 3, where the offset is extracted.

Other File Handling Programs

There are several tasks performed in the signal processing and analysis stages, and it is necessary to present the processes that make it possible. The programs ONE-CH.EXE, ONE-CH8.EXE, GET1OF8.EXE AND ONETO8 extract or insert one signal to the stream of data from or to the file containing the multiplexed stream of seven or eight signals.

4.6 INTERVAL SELECTION AND SIGNAL DISPLAY SOFTWARE

Selected time intervals are selected from the files containing the "raw" or unprocessed signals. This selection is based on signal quality (signal to noise ratio), patterns of breathing, or intervals when a special test was applied to the patients.

The program SELECT.EXE displays the signals and enables the user to select intervals and store them in defined files.

Program: SELECT.EXE (appendix A.3)

This program displays eight non-overlapping channels simultaneously. Its basic function is to allow the user to select a desired interval for later processing and analysis.

The following are the basic features this program provides for modification of display:

. Left and Right signal scrolling.
. Gain and Offset adjustment.
. Waveform compression.
. Timing information.

SELECT.EXE drives the WFS-200 PC waveform scroller board for signal displaying, using subroutines provided in the WFSC.OBJ object module.

The different steps and features are accomplished as follows:

1. Opens files with extensions .DAT (original file with unprocessed data) and .SEL (file that will contain the selected intervals) in binary form for buffered I/O.

2. Sets the video mode (ERESCOLOR, EGA color card, 16 colors). Sets up the scroller board, with the array SETARY, that contains information about channels enabled, scaling factors, offset level, etc. Then it initializes the scroller board.

3. Starts plotting the signals from right to left for the first time (routine FIRST_PLOT), then by pressing assigned keys, it can start scrolling (routines SCREEN_SCROLL and SCROLLING), adjust gains (routine GAIN), and adjust offsets (routine (set_pos).

4. The routine TEST_KEY checks for key pressing and controls flow according to the key pressed.

5. If the interval(s) desired is(are) identified, the user mark the limits (routines CHOOSE_LIMIT and SELECTING) and the program writes the data on the destination file (routine SEND_TO_FILE).

This program keeps track of the file pointer position to change compression factors, mark interval limits, and timing.

4.7 SUMMARY

The MS_DOS operating system controls and supervises the operation of the computer system. It consists of the basic input and output system (BIOS), the system functions (Kernel) and the command processor (COMMAND.COM).

The application programs were developed using C programming language. This language provides a wide variety of libraries and object modules that facilitate the programming tasks.

The data acquisition stage is performed using CODAS (DATAQ Instruments) for short term collections and monitoring, and the program GETDATA.PRG running under ASYST Scientific System. The files collected with these systems contain the multiplexed stream of seven channels of sampled data.

The use of two systems make necessary the use of data translation routines, and import from the original directories to the working directory. After the data is translated to straight binary format, the system allows the user to observe the signals using the program SELECT.EXE and select special intervals for analysis.

4.8 REFERENCES

[1] Duncan R., "Advanced MS DOS Programming", Second Edition, Microsoft Press, Redmond, Washington, 1988.

[2] Barkakati N., "Microsoft C Bible", Howard W. Sams & Company, Indianapolis, Indiana, 1988.

[3] Microsoft C, Run-time Library Reference, Microsoft Corporation, 1987.

[4] CODAS user's manual, Dataq Instruments Inc., Akron, Ohio, 1987.

[5] ASYST Guide, Module 3 Acquisition, Macmillan Software Company, New York, New York

CHAPTER V

DIGITAL SIGNAL PROCESSING AND SOFTWARE DEVELOPMENT

5.1 INTRODUCTION

The applications of digital signal processing have increased with the decrease in cost of computer equipment, and its improvement in terms of speed. This has been reinforced by the concurrent development of efficient numerical procedures (algorithms). So it is common to find digital signal processing applied in many diverse areas, some of which were applied in this project.

Since biological signals cannot be measured independently from their surrounding environment, therefore such signals contain certain amount of artifact contaminants that make analysis of unprocessed data inaccurate. The signal validation requires the application of several filtering and noise extraction techniques, especially in the case of the EMG or electromyographic signals.

5.2 PROCESSING OF EMG SIGNALS

5.2.1 Noise and Contaminants in the EMG Signals

In the measurements of respiratory muscle EMG activity in the neonate, the signals detected by surface electrodes have very low amplitude. These signals are susceptible to electrical noise and frequently they are contaminated by the systems surrounding the muscle and the natural motion of breathing.

Low Frequency and 60 Hz Noise:

Low frequency noise detected in the signals is result of the natural breathing movements or some other kind of spontaneous movement of the patient. Part of this low frequency noise is eliminated using a lower limit of 10 Hz in the Adjustable Band Pass Filters described in Chapter Three.

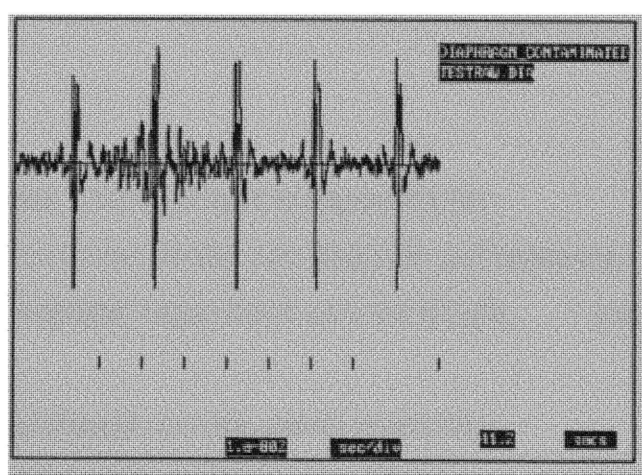
The electrode leads behave like antennas, so that any other electromagnetic activity may be picked up by them [7]. The 60 Hz noise in the system is produced by such effect, and even though the Bioamplifiers described in Chapter Three have very high CMRR at 60 Hz, there is still a good amount of noise seen in the unprocessed signals.

Cardiac EKG Artifact:

The electrical activity of the cardiac muscle spreads out through the surrounding tissue, this electrical activity is picked up during the measurements of electrical activity in the muscles relatively close to the heart.

This cardiac impulse is called EKG contaminant artifact, and its voltage amplitude is relatively large compared to EMG activity affecting waveform and timing analysis of the EMG signals. The frequency spectrum of the EKG contaminant overlaps the spectrum of the EMG signals, distorting the PSD analysis of respiratory muscle EMG activity [6][8]. Figure 5.2.1 shows the Diaphragm EMG signal, contaminated with the intrinsic EKG cardiac artifact and 60 Hz noise.

Figure 5.2.1
EMG signal (diaphragm muscle) with EKG artifact contamination and 60 Hz noise.



5.2.2 Filtering EMG Signals

Finite Impulse Response Filters:

The filters whose impulse response is of finite duration are referred to as finite impulse response (FIR) digital filters. For the FIR filter the response of the filter depends only on the present and past input samples.

A causal non-recursive or FIR filter have difference equations of the form:

$$y(n) = \sum_{K=0}^{L} h(n) * x(n-k)$$ where L represents a finite number of delays.

where h(n) represents the unit sample response.

and its frequency sample response is determined by:

$$H(e^{j@}) = \sum_{n=-oo}^{n=oo} h(n)\, e^{-j@}$$

where @ is the digital frequency, @ = wT = 2*Pi / fs where fs is the sampling frequency.

Characteristics of FIR digital filters [1]:

. FIR filters can be designed with linear phase. LInear phase is important for applications where phase distortion due to nonlinear phase can degrade performance.

. FIR filters are inherently stable, that is, the impulse response is of finite length, and therefore its output is bounded.

. Quantization noise due to finite precision arithmetic can be made negligible.

. A disadvantage of FIR filters is that an appreciably higher order filter is required to achieve a specified magnitude response, thereby requiring more filter coefficient storage.

Fourier Series Method for filter design:

1. Decide upon frequency response $H(e^{j@})$ which is determined by the application.
2. Determine the unit sample response h(n) that will produce the desired frequency response.
3. Modify the unit sample response h(n) to produce a practical filter (size).
4. Implement the digital filter as a program, or microcoded digital signal processor, or integrated circuit.

Gibbs Phenomenon:

Truncating the Fourier series results in FIR filters with undesirable oscillations in the passband and stopband, which will result in slow convergence of the series. These effect is known as the Gibbs phenomenon[1].

The Hamming Window:

To reduce the Gibbs phenomenon, a particular class of functions are used to modify the fourier coefficients (impulse response). These time-limited weighting functions are generally referred to as window functions, one of the most popular window function is the Hamming window [2].

The Hamming window function is given by

$$a(n) = \begin{cases} 0.54 + 0.46 \cos (2*Pi*n/N-1) & \text{for } abs(n) =< N-1/2 \\ 0 & \text{otherwise} \end{cases}$$

The filter design procedure is altered only by the step added to alter the unit sample response of the filter:

h(n) = h(n) * W(n)   where W(n) is the window function.

Filter Design:

The FIR filter performs the following tasks:

. Eliminate low frequency noise (movement) with a low bandpass limit of 20 Hz.

. Eliminates 60 Hz frequency noise.

. Sets up the highest frequency cutoff at 500 Hz.

The following steps describe the design of the filter:

1. The frequency response of the filter is setup to eliminate frequencies below 20 Hz, to band stop frequencies between 57 Hz and 65 Hz, and to limit the spectrum at 500 Hz.

The frequency desired response of the filter is shown in figure 5.2.2.

$$Hd(f) = \begin{cases} 0 & \text{for } f < 20 \text{ Hz} \\ 1 & \text{for } 20 \text{ Hz} <= f <= 57 \text{ Hz} \\ 0 & \text{for } 57 \text{ Hz} < f < 65 \text{ Hz} \\ 1 & \text{for } 65 \text{ Hz} <= f <= 500 \text{ Hz} \end{cases}$$

2. Replacing this values in the integral to determine the unit sample response h(n) of the filter.

$$h(n) = (1/\ 2*Pi) * \int_{-Pi}^{Pi} Hd(e^{j@}) \cos n@d@$$

3. The number of coefficients is truncated with +/- I terms which yields a filter with (2I+1) coefficients and a time window length of (2*I*T) seconds. For this application the filter has 129 coefficients, with a time window of 129 msec.

4. The filter coefficients are multiplied by the Hamming window coefficients.

5. The coefficients are then shifted to the right by 64 (I) terms to make the filter causal.

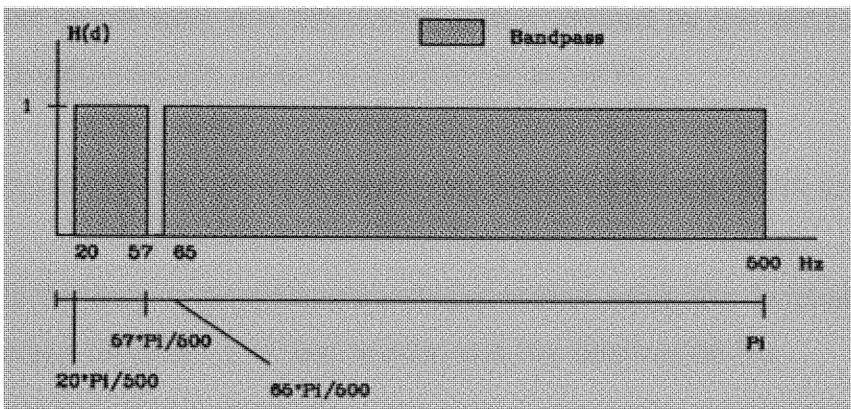


Figure 5.2.2 FIR filter design frequency response

<u>Software Implementation: FIREMG.EXE</u> (appendix A.4)

The program FIREMG.EXE is a implementation of the filter design from the previous section. This is finite impulse response filter with 129 coefficients.

Calling format: FIREMG ORIGIN-FILENAME DESTINATION-FILENAME

Program execution:

The following is a list of steps the program executes and the events that occur during the filtering of the signals:

1. The program receives the origin and destination file names via the passing parameters ARGC and ARGV.

2. Files are opened for buffered I/O, for read and write only.

3. The main program transfers control to the subroutine FILTERING().

4. The number of time intervals or windows of data to be filtered is computed.

6. Computes the window coefficients and store them in the integer array WIN.

7. The filter coefficients are calculated, and multiplied by the window coefficients, the modified coefficients are stored in the integer array H.

8. The program reads 129 data points into the input floating point array.

9. The filtered data is converted to integer format using the function FP2INT from the Fortran library SMSFFT [10] and written into the destination file.

10. The program repeats steps 8 and 9 according to the computed number of signal intervals, giving time information to the user with the function GETTIME(), that uses the system call _DOS_GETTIME and the defined structure DOSGETTIME 0xc.

Function FP2INT: The Fortran library SMSFFT.LIB [10] that contains the function FP2INT is provided with the application package 87FFT developed by Micro Way, Inc. The function FP2INT converts floating point numbers to 16 bits integers in two's complement format.

The calling format is FP2INT (source, destination, n, bits), where source and destination are the source and destination arrays, n is the number of points to be converted, and bits represents the number of significant bits on the destination (byte or word).

Processing Speed:

The high number of filter coefficients slows down the processing speed, but there is trade off. It produces a more rapid transition when going from the passband to the stopband, and the attenuation in the stopband is greater.

5.2.3 EKG ARTIFACT EXTRACTION

Introduction:

As explained in section 5.1.1, the presence of the contaminant EKG artifact on the EMG signals cause serious problems for the analysis of such signals. The EMG signal validation is a very important part of this project. Any posterior analysis of the signals would become inaccurate if this artifact is not completely removed from the signals.

This section presents the technique used for this artifact extraction, preceded by a review of the theory on which it is based.

Cross Correlation Coefficient:

The cross correlation coefficient is used to evaluate the similarity between two sequences of the same size. It has a maximum value of one when the sequences are identical [1].

The correlation coefficient r for two sequences X,Y of size n is defined as:

r = Covariance XY / (St.Dev X * St. Dev. Y)

EKG Artifact Extraction Technique:

For a better understanding of this technique it is necessary to review the following terminology:

Respiratory EMG signal: Electrical activity of the muscles associated with the depolarization that accompanies contraction of the muscle.

EKG signal: Electrocardiographic signal, it is defined by the occurrence of the PQRST complex, which accompanies cardiac muscle contraction.

QRS complex: The QRS complex is caused by passage of the cardiac impulse through the ventricles that occurs every heart beat and is recorded as the EKG signal that it is used in this system for timing purposes.

EKG artifact: This is the unwanted EKG contaminant artifact that is detected in the EMG signals.

Standard Template: It is a sequence that represents certain repetitive event occurring on the signals that has to be identified for processing purposes.

This technique is a modification of the Bloch technique [6] for extraction of the EKG artifact from the respiratory electromyography (EMG). This technique is based upon the timing relationship between the occurrence of the EKG complex and its contaminating presence in the EMG signal.

The procedure is basically divided in two parts: The identification of an EKG artifact within the EMG signal, and the extraction of subsequent occurrences of such artifact.

1. EKG Artifact Template Identification:

It is necessary to obtain a representative sequence (template) of the EKG artifact from the EMG signal. The user is provided with the facility of scanning the EMG signal along with the Flow, Esophageal Pressure and the EKG signal (program GETTEMP4.EXE). It allows the user to identify the occurrence of an EKG artifact during an expiratory interval where there is usually no muscle activity, and manually select the template containing a representative sequence of the contaminant.

When the user selects this template the program automatically separates two templates, one containing the EKG contaminant artifact from the EMG signal (figure 5.2.3-a), and the second template contains a representative QRS (figure 5.2.3-b) complex from the EKG signal. This sequences are stored in a temporary file that will be used for the extraction procedure.

Figure 5.2.3-a
EKG artifact contaminant template obtained from diaphragm EMG during expiratory interval. Duration 175 ms.

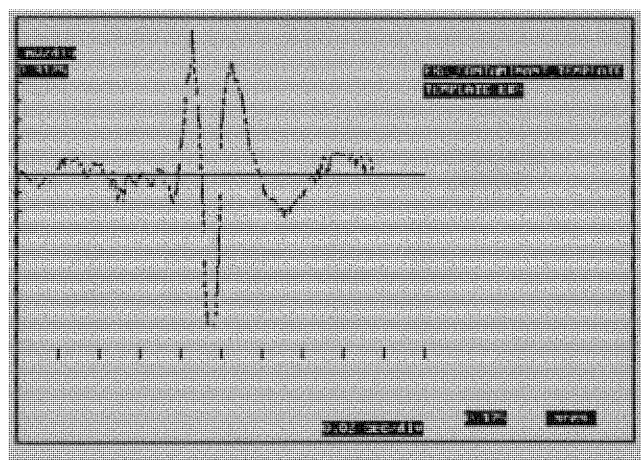
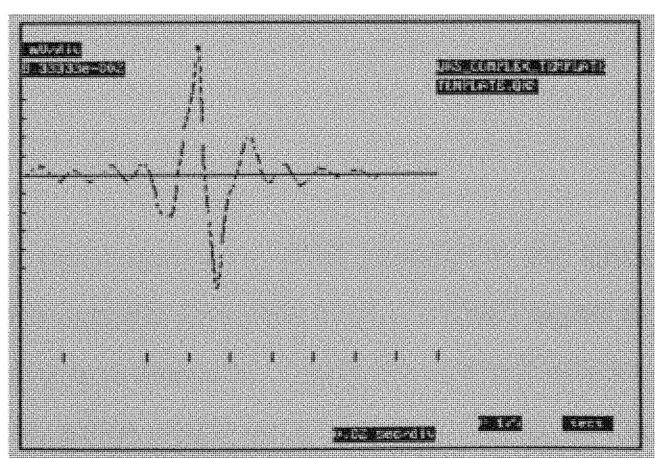


Figure 5.2.3-b
QRS complex template obtained from the EKG signal. It will be used for detection of cardiac muscle activity.



2. Searching for QRS Complexes:

The QRS complex template obtained by the user, is used by the program CLEANEMG.EXE for an automatic searching procedure for the occurrence of the QRS complexes.

For this purpose the program correlates the QRS complex with intervals of the same length of the EKG signal (figure 5.2.3-c), obtaining a correlation coefficient that is compared with an empiric threshold level previously set during programming. If the calculated correlation coefficient is greater than the threshold level, the algorithm searches for the occurrence of a peak on the correlation coefficient value by shifting the QRS complex template one millisecond at a time. This gives a maximum time resolution, in the order of milliseconds, to detect the occurrence of a cardiac QRS complex on the EKG signal.

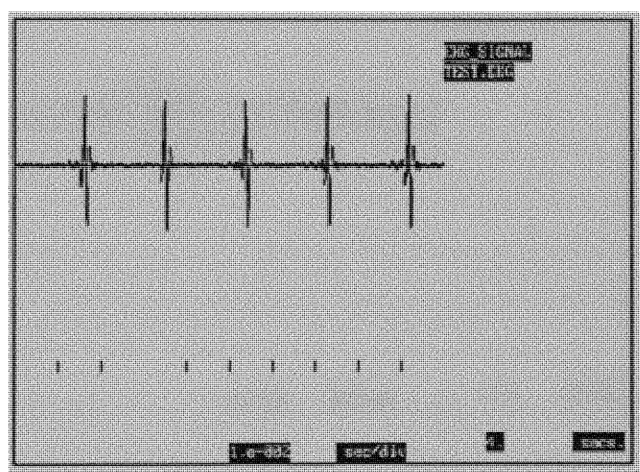


Figure 5.2.3-c
EKG signal, the algorithm identifies the occurrence of each QRS complex using this signal.

3. EKG Artifact Removal:

When the program has detected the occurrence of a QRS complex in the EKG signal, it assumes the occurrence of an EKG contaminant artifact on the EMG signal at the same point in time where the QRS complex was detected. This is the major difference of this technique from the original technique, which searches for the occurrence of the EKG contaminant artifact on the EMG signal itself. This modification is necessary to avoid errors detecting this artifact, since in some cases it is buried in the EMG signal, or the EMG activity is coincidentally is similar to the EKG artifact causing false detections.

Once the EKG artifact is detected in time, the program checks if the EKG contaminant encountered is a real artifact, with the cross correlation coefficient compared with a second threshold level (lower). If the coefficient is above this threshold the algorithm subtracts the EKG artifact template point by point from the EMG signal at that point in time, leaving only the "pure" EMG activity.

The program shifts then the QRS template in time until a new complex is detected and the procedure is repeated. Figure 5.2.3-d shows the same interval of figure 5.2.1 after the artifact has been extracted.

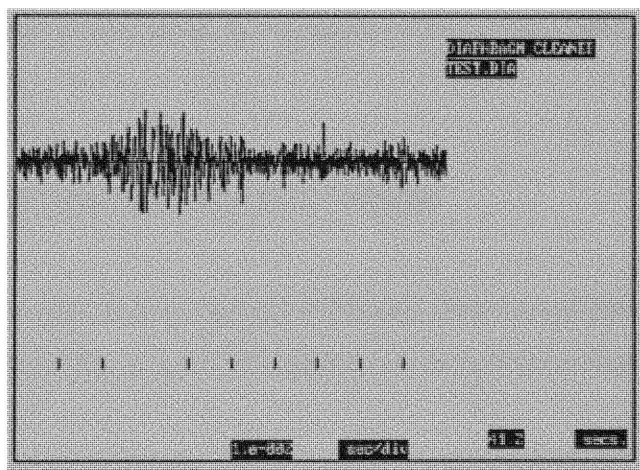


Figure 5.2.3-d
This is the EMG signal showed in figure 5.2.1 after filtering and EKG extraction.

Software Implementation: CLEANEMG.EXE (appendix A.5)

This program performs the functions mentioned in the previous section.

Calling format: CLEANEMG FILENAME.TWO FILENAME.TEM FILENAME.FLT

Where FILENAME.TWO is a file that contains the EMG and EKG signals, the file FILENAME.TEM contains the templates of the EKG artifact and QRS complex mentioned above, and the file FILENAME.FLT will contain the EMG signal free of EKG artifacts.

Program execution:

The program goes through the following steps during execution:

1. Checks if it has received the right number of parameters, then it sets up the interrupt signal handlers Ctrl-C (SIGINT) and Floating point Interrupt (SIGFPE).

2. Opens the three files for buffered I/O.

3. Initializes variables PEAK (0.0) and LEVEL (0.9). PEAK is the variable that contains the peak correlation coefficient when a QRS complex is identified, while LEVEL is the threshold level variable, that contains the empirical level for identification of the QRS complex occurrence.

4. Reads the QRS complex and EKG artifact templates into buffer arrays.

5. The control is transferred to the subroutine CROSS_CORRELATION(), that reads the EMG and EKG signals, determines the occurrence of a QRS when a peak value on the correlation coefficient is reached.

6. If a QRS complex is detected, it passes an array containing the EMG signal and the EKG contaminant artifact template to the function CHECK_ARTIFACT that correlates these two arrays.

7. If this second coefficient (variable COEFF_CHECK) is above 0.5 then the EKG template is subtracted from the EMG signal, point by point.

8. Steps 4 to 7 are repeated until the end of file (EOF) is reached.

9. The program provides information to the user about execution times, number of QRS

complexes detected and extracted.

10. At the end it closes the files and returns control to the calling process.

This program uses two interrupt handling subroutines: CTRLC_HANDLER and FPERR_HANDLER, that uses the library routine SIGNAL. It also uses the Fortran library subroutine FP2INT described above.

The program GETTEMP4.EXE that is used for the EKG contaminant artifact and the QRS complex templates selection, is a variation of the program SELECT.EXE from chapter four. It scrolls 4 signals, and sends the selected interval to the destination file.

Processing Speed:

The processing speed of this programs is reduced because of the large number of computation it requires for the cross correlation coefficient calculation. The accuracy of the timing resolution is to the order of milliseconds, which increases the processing time.

To accelerate the process, the program handles large data buffers, to avoid disk access time. It also decreases the time resolution, and automatically increases it when it detects the presence of cardiac activity.

5.2.4 Moving Time Average

Introduction:

One of the major difficulties in interpreting electromyograms (EMG) measured with surface electrodes, is that the instantaneous value of the signal contains little information about the mechanical response of the muscle [9]. The information needed from the EMG signals is generally of two types. First the activation and deactivation times, and second, as well as the magnitude of the electrical activity related to the average force of the muscular contraction.

Averaging several instantaneous responses point by point forms a composite waveform representing the mean response. It also reduces the level of random noise in the electrical response of the muscle.

In order to obtain this information the EMG signals are rectified and passed to an averager filter which obtains the envelope of the composite rectified EMG signal as shown in figure 5.2.4.

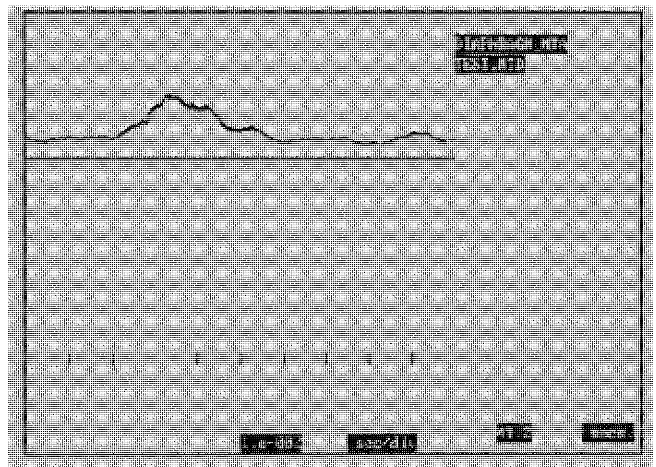


Figure 5.2.4
Output signal of the moving time averager filter (see input signal in figure 5.2.1).

Moving Time Averaging Technique:

A moving time averager filter is an basically a low pass filter excellent for this type of application because of its rapid dynamic response.

Digital Paynter Filter:

The third order digital lowpass Paynter filter is basically a three pole Butterworth filter transformed to digital form [5].

Butterworth filters are characterized by maximally flat response in the pass band, that is, there is a minimum amount of ripple on their frequency response for the passbands.

The analog transfer function of this filter is the following [5]:

$$H(s) = \frac{1 + (s/wp^2)}{(1 + 2s/wp)\,[1 + 1.2s/wp + 1.6(s/wp)^2]}$$

where wp = Pi/Tp, being Tp the averaging interval. In our case the averaging interval is 100 ms. The averaging interval was chosen empirically, it makes the response of the filter to follow the dynamic activity of the muscles.

Bilinear Analog to Digital Transformation:

This method for designing digital filters from an already designed analog filter may be interpreted as a mathematical transformation from the s-domain (Laplace) to the z-domain [1]. A primary advantage of the bilinear transformation is that it provides a one-to-one mapping of poles and zeros from the continuous time S-plane to the discrete time Z-plane. Since the entire imaginary axis on the S-plane is uniquely mapped onto the unit circle in the z-plane, the bilinear transformation has a compression effect, known as frequency warping, on the frequency response characteristics. This effect is alleviated by prewarping critical frequencies and using scaling [3].

Scaling frequencies: $H(s) = H(s)\Big|_{s = s/\tan(wc/2) = s/\tan(Pi*fc*T)}$

where wc is the cutoff frequency in radians and T is the sampling interval.

Using the bilinear transformation:

That is: $H(z) = H(s)\Big|_{s= (z-1)/(z+1)}$

This transformation may be written as: $sz + s - z + 1 = 0$ equation that shows this transform is linear in both domains (bilinear).

The transfer function of the Paynter filter is transformed using this method, to obtain its transfer function in the Z domain (discrete):

$$H(z) = \frac{A0\,Z^3 + A1\,Z^2 + A2\,Z + A3}{B0\,Z^3 + B1\,Z^2 + B2\,Z + B3}$$

and the inverse Z transform, yielding the differential equation:

$$y(n) = A0\,x(n) + A1\,x(n-1) + A2\,x(n-2) + A3\,x(n-3) - B1\,y(n-1) - B2\,(y(n-2) - B3\,y(n-3)$$

which can be easily implemented as a digital filter.

<u>Software Implementation: MOVING.EXE</u> (appendix A.6)

This program is the implementation of the digital Paynter filter for the moving time averaging of the EMG signals. The input of these program is the rectified EMG signal delivered by the program RECTIF.EXE, and the output is the filtered signal stored in a file.

Calling format: MOVING FILENAME.EMG FILENAME.MTA

where FILENAME contains the rectified EMG signal and FILENAME.MTA contains its moving time average.

Program execution:

The sequence of steps occurring during program execution is the following:

1. The program calculates first the filter coefficients with the routine CALC_COEFFS(), and the averaging interval is defined as Tp = 100 ms.

2. The routine FILTERING() reads buffers of integers into the array buff_get_int, the sequence is converted to floating point format for calculation.

3. Once the destination array is full the floating point sequence is converted to integer and written into the destination file. This procedure is repeated until the end of file is reached.

5.3 PROCESSING OF RESPIRATORY AND CHEST WALL SIGNALS

5.3.1 Noise

The respiratory and chest wall motion signals have very good signal to noise ratios. However, in some cases the signals are contaminated with high frequency and 60 Hz noise [9].

The esophageal pressure is sometimes contaminated by cardiac motion artifact, which is a non-electrical contamination. Instead, the heart movement depresses the esophagus wall causing differences in pressure that are detected by the pressure transducers (See figure 5.3.1-a).

The flow signal may also be contaminated, but in this case the contamination is caused by the inertia of a mechanical one-way non rebreathing valve during changes in the direction of the air flow, which causes some oscillation in the flow signal before it opens completely (see figure 5.3.1-b).

Figure 5.3.1-a

Esophageal pressure signal contaminated by the cardiac on the esophagus.

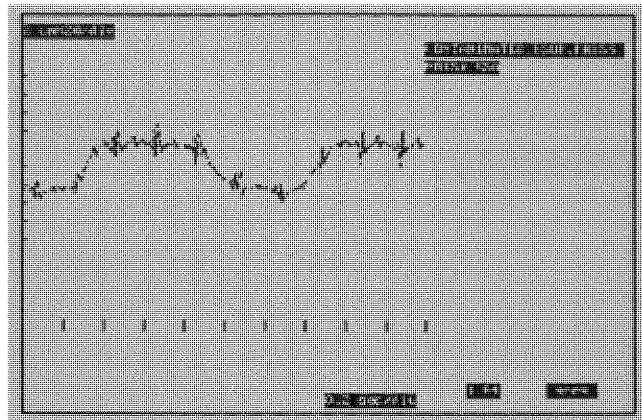
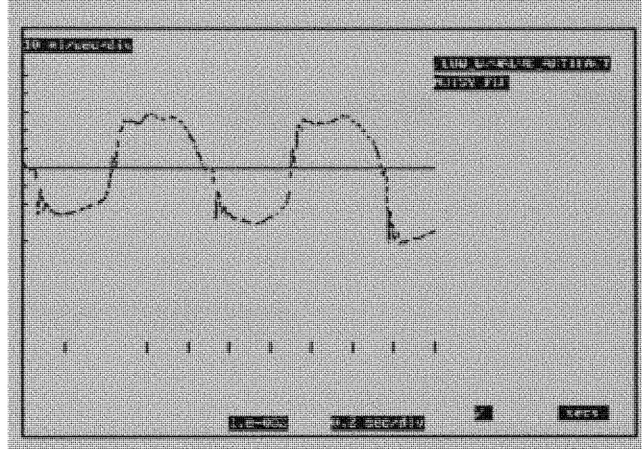


Figure 5.3.1-b

Flow signal with valve noise.



5.3.2 Signal Smoothing

Elimination of noise is begun by using a low pass "smoothing" filter. This filter eliminates frequencies above 10 Hz, eliminating high frequency noise and contaminants, improving signal quality for later analysis. Figures 5.3.2-a and 5.3.2-b show the signals mentioned showed before after smoothing.

Figure 5.3.2-a

Esophageal pressure after smoothing.

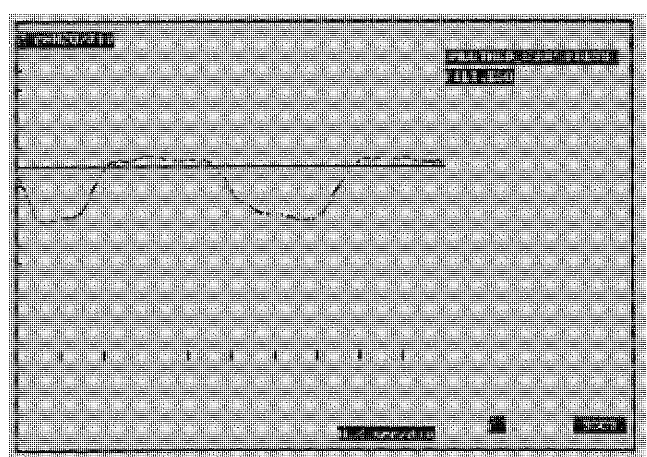
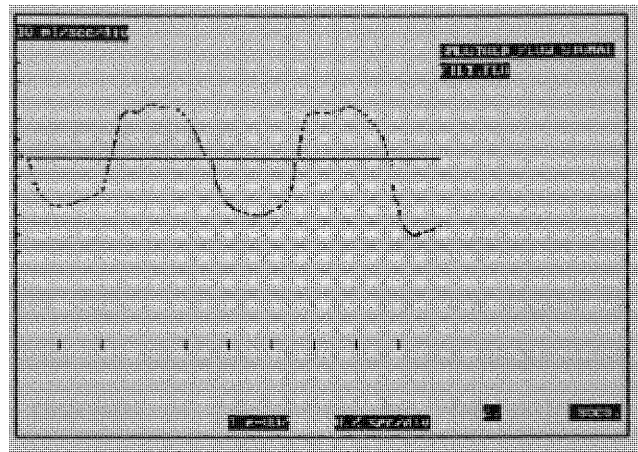


Figure 5.3.2-b

Smoothed flow signal without valve artifact.



This filter is a digital first order lowpass filter of the form:

$H(s) = 1/(1 + s/wc)$

where wc is the cutoff frequency.

This filter is digitized using the bilinear transformation to obtain the Z transform of the filter which is implemented as a digital filter:

$H(z) = (A0\ z\ + A1) / (B0\ z + B1)$

where

$A0 = wc$ $\quad\quad$ $B0 = wc + 1$

$A1 = wc$ $\quad\quad$ $B1 = wc - 1$

yielding the equation:

$y(n) = A0\ x(n) + A1\ x(n-1) - B1\ x(n-1)$

<u>Software Implementation: SMOOTH.EXE</u> (appendix A.8)

This program is the digital implementation of a lowpass filter.

Calling format: SMOOTH FILEDATA DATA_FILTERED

Where FILEDATA is a file that contains any of the signals mentioned above, and FILTERED_DATA is the file that contains the signals after smoothing.

5.3.3 Tidal Volume: Digital Integration of Flow Signal

As described in chapter two, the Tidal volume is defined as the integration of the air flowing in and out of the lungs [11]. The digital integration of the flow signal is performed when the files containing the signals are imported from the data acquisition directory to the working directory. This process is performed by the program IMPORT.EXE that executes three child processes: ONE_CH8.EXE extracts the flow signal from the file containing seven signals, INTEG.EXE performs the digital integration, and ONETO8.EXE creates a file containing the original seven signals plus the TIdal Volume signal added to the data stream.

The reason for adding the Tidal volume signal as an extra channel to the rest of the signal is because during the process of selecting the breaths for analysis it is important to have consistency in the lung volumes from breath to breath.

Discrete Approximation of Integration:

The process of integrating a continuous-time signal x(t) from an initial time t0 the time t is approximated by the difference equation:

$$y(nT) = T\,x(nT) + y(nT-T)$$

where y(nT) is the value of the integral at the sampling instant t = nT,

y(nT-T) is the value of the integral at the previous sample

t = nT - T, x(nT) is the value at t = nT of the sequence being integrated,

and T is sampling interval [1].

The machine computation of an integral always introduces potential for errors. In this case the errors are minimal because the sampling interval is much smaller than the signal's interval.

Digital Integrator Design:

The mathematical formula for the area of a trapezoid can be used to develop the difference equation that approximates the mathematical operation of integration.

The integral of a function x(t) can be interpreted as the area under the curve x(t) vs t. The area under the curve during the interval (n-1)T to nT (using the trapezoid's area formula) is [1]:

$$area = T * (x(n) + x(n-1)) / 2$$

if y(n) is the total area under the curve in the interval -oo to nT, and y(n-1) is the area in the interval -oo to (n-1)T then:

$$y(n) = y(n-1) + T * (x(n) + x(n-1))/2$$

where T is the sampling interval.

Software implementation: INTEG.EXE (appendix A.7)

The difference equation that represents the area under the curve of the sequence x(t) can be easily implemented in a digital algorithm. It is basically the summation of the previous value plus the increase in the area during the next interval.

The digital integration is performed by the program INTEG.EXE that is "spawned" by the program IMPORT.EXE (see chapter three).

Calling format: INTEG FILENAME.FLO FILENAME.VOL

where the extensions .FLO and .VOL indicate the name of the files containing the flow and Tidal volume respectively.

5.4 FILTERING CARDIAC SIGNALS

The cardiac signal or EKG is affected by the low frequency movement noise from the breathing motion and the 60 Hz noise.

This signal is filtered using a FIR filter with the following frequency response:

$$H(f) = \begin{cases} 1 & \text{for } 20 < f < 50 \text{ Hz} \\ 0 & \text{elsewhere.} \end{cases}$$

The EKG signal has most of its frequency contents between 20 and 50 Hz, which includes the Q,R, and S waves. The P and T waves are eliminated because of they have little influence on the EMG signal and they are not needed for our timing purposes.

The program FIREKG.EXE is the digital implementation of this filter. This filter was implemented with the same basic algorithm used for the program FIREMG.EXE from section 5.1. This filter has only 33 coefficients, requiring less memory space for processing.

5.5 EMG SIGNAL PROCESSING MENU SYSTEM

The large amount of files and the sizes of each of them, required the creation of a menu system for the signal processing of the EMG signals. This menu system is the interface between the operator and the signal processing routines mentioned before.

Once the data files have been imported from the data acquisition directory to the working directory (either by IMPORT.EXE or RETRIEVE.EXE), the Tidal volume signal has been created and incorporated to the data stream, and the user has selected the interval of interest (a portion of or the complete file), the files are ready to be processed and prepared for posterior analysis. The files containing the selected interval are assigned with the extension .SEL (selected).

Software Implementation: PROCESS.EXE

The program PROCESS.EXE is the interface the user has for the execution of those routines. This program guides the user through all the steps required for the EMG signals processing.

Calling format: PROCESS FILE1 FILE2 ...FILEn

where FILE1, FILE2..FILEn are the selected files, and they are checked for the extension .SEL. This program allows a maximum number of twenty files. This number can be increased, but it is not likely to have more than ten files per study.

The design of this program was based on the need for having an automated system with the minimum amount of interaction with the operator. This automation became a necessity because of the time required to process long data files, which are on average 800 KB long, thus allowing the operator to perform other activities while the signals are processed.

The filenames are passed to the program through the passing parameters ARGC and ARGV, the filenames formatted and stored in a special structure called FILE_TO_ANALYZE for posterior retrieval and processing. The string arrays containing the filenames for the different signals and data files are setup using string handling routines (STRCPY, STRCAT).

The following are the options the user is provided during the processing stage:

F1. Separate file into 8 channels

The selected files contain 8 different signals in a continuous stream of data. The subroutine SEPARATE_FILES inquires for the channel configuration and then spawns the program GET1OF8.exe that extracts one signal at a time and stores in file according to the filename assigned to the structure FILE_TO_ANALYZE.

F2. Filter EMG signals

After the signals have been separated into individual files, the EMG signals are filtered with the programs FIREMG.EXE and the EKG signal is filtered with the program FIREKG.EXE. These filters produce some delay in the signals that is corrected with the program DELAY.EXE. The filtered signals overwrite the noisy signals in the data files.

F3. Obtain Artifact templates

After the signals have been filtered, the control is transferred to the subroutine SELECT_TEMPLATES, that creates a temporary file containing the flow, esophageal pressure, EMG, and the EKG signal. This file is then passed as parameter to the program GETTEMP4.EXE , and the user scans the signals and selects the EKG artifact from the EMG signal and the QRS complex from the EKG signal during an expiratory interval. This sequences are stored in a file with the termination .TED.

F4. Extract EKG artifact from EMG

Once the templates have been selected, the operator initiates the extraction of the EKG artifact. The control is transferred to the subroutine CLEAN_ARTIFACT and it calls a child process (program CLEANEMG.EXE) that performs the extraction of the artifact, and outputs the cleaned signals into a temporary file.

F5. Performs F3 and F4 together.

F6. Performs F1 through F4.

F7. Moving time average

This option calculates the moving time average of the signals, the subroutine CALC_MTA executes a child process (program MOVING.EXE) that receives as input any EMG signal and outputs the moving time average of that signal. Once the moving time averages (MTA) of the three muscles have been obtained, it creates a file with the extension .MOV that contains the flow, Diaphragm MTA, Genioglossus MTA and the

Posterior cricoarytenoid or PCA MTA.

5.6 SUMMARY

The software development for the processing of the EMG signals eliminates low frequency noise from respiratory movement (breathing frequency of infants is 40 to 60 breaths/minute). It also eliminates 60 Hz electrical noise. The most important part of the EMG signal processing is the extraction of the EKG artifact, to obtain an EMG signal that is free from the EKG contaminant. Once the EMG signal is cleaned, it is passed through a moving time averager for later analysis. The EKG signal that is used as timing reference for the cleaning of the EMG signals is bandpass filtered.

The respiratory and chest wall motion signals are lowpass filtered or smoothed, eliminating any high frequency noise content on these signals. The Tidal volume signal is obtained from the digital integration of the flow signal.

The processing of the signals mentioned before is controlled by a menu system that acts as interface between the user and the different subroutines and programs.

5.7 REFERENCES

[1] Strum R., Kirk D., "First Principles of Discrete Systems and Digital Signal Processing", Addison-Wesley Publishing Company, Inc., 1988.

[2] DeFatta D., Lucas J., Hodgkiss W., "Digital Signal Processing: A System Design Approach", John Wiley & Sons, Inc., 1988.

[3] Stearns S., David R., "Signal Processing Algorithms",Prentice-Hall, Inc., 1988.

[5] Bruce E., Goldman M., Mead J., "A digital Computer Technique for Analyzing

Respiratory Muscle EMG's", Journal of Applied Physiology, 1984.

[6] Bloch R., "Subtraction of Electrocardiographic Signal from Respiratory Electromyogram", Journal of Applied Physiology, 1983.

[7] Cobbold R., "transducers for Biomedical Measurements: Principles and Applications", John Wiley, New York, 1972.

[8] Levine S., Gillen J., Weiser P., Gillen M., Kwatny E., "Description and Validation of an ECG removal procedure for EMGdi power spectrum analysis", Journal of Applied Physiology, 1986.

[9] Gottlieb G., Agarwal C., "Filtering of Electromyographic Signals", American Journal of Physical Medicine, 1970.

[10] 87FFT Reference Manual, Micro Way Inc., 1985.

[11] West J., "Respiratory Physiology", Williams & Wilkins, 1990.

CHAPTER VI

SOFTWARE DEVELOPMENT AND SIGNAL ANALYSIS

6.1 INTRODUCTION

The computer aided analysis of biological data augments the ability of the researcher to identify certain activity patterns in the respiratory system. Without this tool, certain analysis procedures would require lengthy calculations with a large potential for errors.

The methods for analysis of different components of the respiratory system have been adapted for computer aided analysis from pre-established criteria.

6.2 SOFTWARE DEVELOPMENT FOR DATA ANALYSIS

The software implemented for signal analysis requires graphics display of the signals, and some special plotting functions for imparting important information to the user.

The software for data analysis was implemented in Microsoft C Version 5.1, and the graphics features were developed using GSSCGI Version 2.1, which is a special graphics applications development tool kit [17].

GSSCGI is a device independent interface to graphics devices. It allows the computer to control several devices simultaneously. GSSCGI provides for the output of graphics primitives (such as lines and text) with control over primitive attributes (such as color, size).

The GSSCGI functions can be accessed through high level language calls, using language bindings that access graphics functions directly as high level language calls with formal parameters. This ensures computer independence.

GSSCGI provides a special binding that interfaces between the C program source code and the CGI interface. The binding interface (MCLCGI.LIB for Microsoft C) is linked as an external function library after compilation; in this way the software developed is completely portable across any compatible system.

GSSCGI is consistent with the "Computer Graphics - Interfacing techniques for dialogue with graphical devices" developed by the American National Standards Institute (ANSI) and the International Standards Organization (ISO), providing source code portability, and device independence [17].

Raster Technology and Bitmaps:

The demand for high performance graphics systems and software has increased in recent years. The tool utilized for this purpose is called raster technology and it is based on television technology.

In the raster display, the refresh memory is arranged as a two dimensional grid or array, with each screen location and memory location is referenced by a pair of coordinates. Image refreshing is done by sequential raster scan through the display buffer by scan line rather than by output primitive.

Each memory location defines one point element of the image, and they are called pixels or pel's (picture elements). New graphic display systems can store and scan images very quickly, so higher resolution and flexibility can be achieved with the manipulation of the display buffer array, which is commonly called bitmap.

Device Drivers:

There are two device drivers that are incorporated into the CONFIG.SYS system file, IBMEGA.SYS and IBMGPR, that control the monitor and the hardcopy unit respectively. The output devices are directed initially with the instruction SET in the AUTOEXEC.BAT file. However, they can be redirected to obtain different type of output such as hardcopy.

6.3 EMG SIGNAL ANALYSIS

The analysis of the EMG signals is focused in two important areas: electrical characteristics of the composite EMG signal (moving time average of the EMG signals) and the frequency contents of the EMG signal itself [5].

6.3.1 Analysis of Moving Time Averaged EMG Signals

To obtain information about the mechanical response of the muscles it was necessary to determine the moving time averaging (MTA) of the EMG signals measured with surface electrodes (see Chapter Five).

The output signal of the MTA Paynter filter is analyzed using special software developed for this purpose. The clinical research has interest in the following parameters:

1. Respiratory timing information.

The flow signal is used as a reference signal to obtain the timing patterns of breathing. The points where inspiration begins and expiration ends mark the duration of a complete breath, and is labeled Ttot (breath duration time). Ti is the inspiratory interval, and Te is the expiratory interval.

Determination of beginning and end of inspiration/expiration:

The algorithm identifies zero crossing points in the flow signal, that change from negative to positive values, and thus marking the point of onset for of inspiration. Once all the N beginning of inspiration points are marked, the algorithm assumes the existence of N-1 breaths. The points of end inspiration or beginning of expiration are marked as the zero-crossings between two beginning of inspiration points. The time between two consecutive points marking beginning of inspiration is Ttot, while Ti is the time between beginning of expiration and beginning of inspiration within that Ttot, and te = Ttot - Ti.

2. Baseline level or non-phasic tonic activity.

After the EMG signals have been processed, and show an acceptable signal to noise ratio (3:1), they will still contain what is called baseline noise voltage level. The EMG signals obtained from the respiratory muscles generally have phasic activity patterns that coincide with the breathing. However in some cases the muscles have prolonged tonic or non-phasic activity that can be confused with baseline noise levels.

Determination of the baseline level on the MTA of the EMG signals:

To determine the baseline level, the algorithm obtains the mean activity level of the MTA EMG, this level is used as a threshold value to obtain the baseline level. Sets of three points separated by 250 ms from each other are evaluated. Two slopes are calculated between these three points, slope #1 is the slope between the first and the second point, while slope #2 is the slope between third and second points. If the product of both slopes is negative or zero, and slope #1 is negative while slope #2 is positive, then the second point is included for the average of the baseline level points as a minimum value if it is below the mean activity threshold obtained before. The algorithm shifts one point in time and evaluates the next set of three points. All the second points included for the average represent what we call the baseline activity level on the MTA of the EMG signals.

3. Peak and delta activity :

The moving time average of the EMG signals obtained with the Paynter filter follows the dynamic response of the signals. This response reaches its maximum level when the activity of the motor potential units (MUP) depolarizing within the muscle is maximal. The delta peak activity is measured as the Peak activity, which is the difference between the maximum value of the signal and the baseline level.

This information is valuable in determining motor output recruitment of additional motor units and timing measurements.

Determination of peak activity level:

The peak activity level is simply obtained as the maximum value occurring on the MTA signal during the interval Ttot. The point of occurrence is stored in memory as well as the difference between these peak value and the baseline level.

4. Activation and deactivation time:

This parameters give information about the duration of the muscle activity, which is useful for the determination of neural drive time, total activity, and timing relation with the ventilatory process.

The muscle activation time is determined as the point when the signal reaches a level above the baseline level. Deactivation time is determined as when the signal returns to the baseline level after reaching a peak activity value.

Activation/Deactivation times determination:

Once the peak activity value is found, the algorithm searches in both directions for the points where the activity returns to a baseline level for the first time. These points mark the on-set and off-set of EMG activity.

5. Total activity:

The total activity is proportional to the mechanical force the muscle exerts during the total duration of activity. This parameter is obtained by the integration of the EMG MTA signal during the activation and deactivation period.

Determining the total activity per breath:

The total activity of the muscle is determined by the area under the curve MTA of EMG vs the duration of the interval between on-set and off-set of the activity.

<u>Software Implementation: PLOTMTA.EXE</u> (appendix A.9)

The algorithms used to determine the parameters mentioned above are implemented in the program PLOTMTA.EXE. These algorithms were adjusted to the characteristics of the signals, but they can be applied in many different areas.

Calling format: PLOTMTA FILENAME.MOV DESCRIPTION_TEXT

Where FILENAME.MOV is the file created by the program PROCESS.EXE and contains four signals: Flow, MTA EMG diaphragm, MTA EMG Genioglossus, and MTA EMG PCA. DESCRIPTION_TEXT is a character string for user information.

This program presents a graphic display of the files with extension .MOV that contains the flow signal (channel 1), MTA of diaphragm muscle (channel 2), MTA of PCA muscle (channel 3), and MTA of Genioglossus muscle (channel 4). It displays 2.5 seconds intervals of the four signals on the screen, which is approximately two complete breaths. The user can scan the signals in both directions, and select the breaths that contain the information that is considered important. The information per each individual breath is averaged with all the information from the selected breaths and presented as a final report.

Every time the user scrolls the signals (left or right), new information is displayed on the screen about the parameters mentioned before.

Program Execution:

Some of the subroutines executed by the program are included in appendix C. This program written in C is linked with the library MCLGCI.LIB (Graphic Software Systems, Inc.) for the graphic displaying of the signals.

1. The program requests the gain setting for the bioamplifiers, optic coupler receivers, and the A/D board gain settings per channel, to calculate and output the signal's voltage levels.

2. The program opens the workstation SCREEN and inquires for its bitmap. Then it reads the first interval of data into a buffer which is plotted on the screen by the subroutine LINES, which then calls the subroutine DETECT_BREATHS.

3. The subroutine DETECT_BREATHS calls another subroutine, CALC_LINES_VOLTS, which obtains the baseline activity level for each of the three muscles. Then it obtains the flow signal timing information such as Ttot, Ti, and Te, mentioned before. With this timing reference the subroutine finds the peak activity on the MTA EMG signals, and the points of on-set and off-set of activity.

4. The point of peak activity can be selected manually by the operator, as an additional option, if the subroutine PLOT_CURSOR is enabled by the user.

5. The values such as delta peak activity, total activity and the timing information is calculated for each breath shown on the screen. The operator can individual breaths when its characteristics meet preset limits, and the program will compute the mean value from the string of selected breaths.

6. When the flow signal is contaminated by a valve artifact caused by the mechanic valve

described in the Chapter Five, the user can exit the program and request signal smoothing, and then return to the program. To do this, the program spawns the program SMOOTH.EXE also described in the Chapter Five.

The program also makes some special key assignments to provide the user features such as:

. Report generation
. Include values for calculation of the mean
. Signal magnification on screen
. Scrolling of signals
. Smoothing of the flow signal

The program outputs the following information on the screen for each complete breath shown (see figure 6.3.1):

. Total activity
. Peak activity value
. Delta activity
. Baseline level
. Time from the beginning of the file in seconds
. Graphics display of the points of occurrence and magnitude of these parameters.

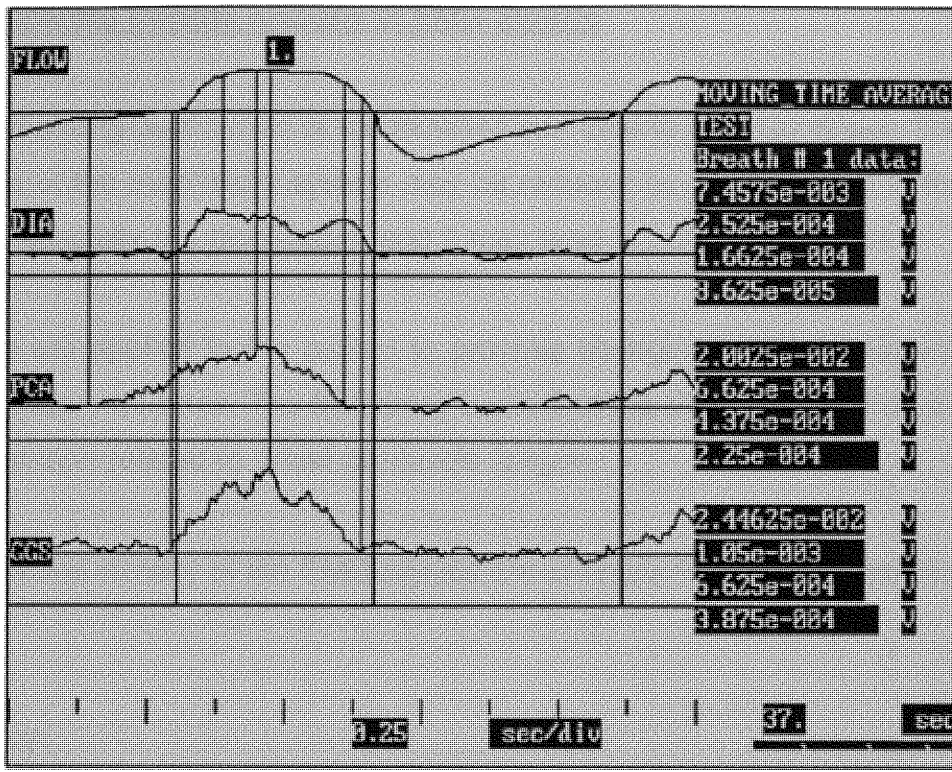


Figure 6.3.1

Screen output of the program PLOTMTA.EXE for the analysis of the EMG activity on three muscles. The vertical lines mark points of beginning of inspiration and expiration, points of maximum activity, and activation/deactivation times of the muscles. The horizontal lines represent the baseline activity levels.

Hardcopy utility:

The display output of the program can be redirected to the hardcopy unit. This is done by using the SET DISPLAY=IBMGPR instruction, and executing the program PRINTMTA.EXE which is a variation of the program PLOTMTA.EXE, and performs the hardcopy of the signals and calculates the parameters mentioned above automatically.

6.3.2 Power Spectral Density Analysis of EMG Signals

Prolonged and strong contraction of the muscles can lead to muscle fatigue. The mechanical response of the respiratory system may be reduced by this condition in the respiratory muscles, specially the diaphragm, and it may lead to respiratory failure.

Different tests are applied to the patient to provoke changes in the mechanical response of the respiratory muscles. The frequency content of the EMG signals contain some valuable information related to onset of muscle fatigue.

The power spectral density of the EMG signals are characterized by the following parameters:

1. Centroid Frequency:

The centroid frequency represents the arithmetic mean of the signals during the frequency interval in which they are analyzed. It is defined as fc[Hz]:

$$fc = \frac{Sum^i \, fi * P(i)}{Sum^i \, P(i)}$$

for i = 0 to 500 Hz (for the respiratory EMG signals).

where fi represents the ith frequency interval in Hz, and P(i) is the power magnitude at the ith frequency interval.

Some studies have proven that there is a shift in the centroid frequency during muscle fatigue states as compared to normal muscle activity[ref].

2. Total Power:

The total power is represented by:

$$P = sum^i \, P(i) \quad \text{for } i = 0 \text{ to } 500$$

3. High to low ratio:

The high to low ratio of the frequency spectrum is defined as the ratio between the power of two frequency bands. For the analysis of the respiratory muscles spectrum this frequency bands were selected as High: 150 to 350 Hz, and Low: 20 to 50 Hz.

Discrete and Fast Fourier Transform:

The Discrete Fourier Transform (DFT) is used in many applications, to transform an ordered sequence of data samples from the time domain to the frequency domain, to obtain spectral information about the sequence. The discrete Fourier transform is the representation of a finite-duration sequence as a finite sum of exponentials.

The discrete Fourier transform pair can be defined as:

$$x(nT) = 1/N \sum_{K=0}^{N-1} X(kDf)\, e^{j(2Pi/N)nK}$$

and $$X(kDf) = \sum_{n=0}^{N-1} x(nT)\, e^{-j(2Pi/N)nk}$$

where $Df = fs/N$ represents the frequency spacing between coefficients, T represents the sampling interval,and N is the period of the sequence [1].

The Fast Fourier Transform (FFT) is a fast algorithm for the computation of the DFT, and its output is the same set of complex values. The FFT algorithm eliminates most of the repeated complex products in the DFT. The ratio of computing is

approximately [3]: $$\frac{\text{FFT computing time}}{\text{DFT computing time}} = \frac{\log(\text{base}2)\ N}{2\ N}$$

Another advantage of the FFT is that requires less amount of storage space for the computation, because it overwrites the previous values. The FFT performs operation over sequences of size N, where N is an integer power of two.

Real Radix-Two Fast Fourier Transform Utility:

The program implemented for the spectrum analysis of the EMG signals (SPECTRUM.EXE described below) is based on the Real Radix-two Fast Fourier Transform (RFFT) subroutine written in assembly language included in the signal processing package 87FFT (Microway Inc.) [10].

RFFT performs an in-place 1024 points radix-two FFT on real data and returns the first N/2 + 1 complex numbers of the DFT.

Calling format: RFFT (array,exponent,norm,scale)

where array is a real array containing the data, the returned N/2 + 1 complex numbers are written in the same array, exponent is a positive integer power of two equal to the number of data points, norm is the normalization mode, and scale is a scaling factor.

Windows in Spectral Analysis:

When a data record is truncated, the single frequency component is "smeared" to the sides of this frequency. This smearing effect is known as leakage.

A primary source of this leakage is the discontinuity introduced in the periodic extensions by truncating the sequence, and it is common to use window functions to avoid this phenomena. The fundamental idea of the window functions is to gradually taper the data near the ends of the record, to avoid abrupt truncations.

HAMMING WINDOW UTILITY:

In the program SPECTRUM.EXE, before the RFFT function obtains the 1024 points FFT, the sequence is multiplied by the HAMM window function utility, which smooth and symmetrically tapers the real data sequence [10].

Calling format: HAMM(source, destination, exponent)

where source is a real array containing the 1024 points of the real data, destination is the real array containing the windowed sequence, and exponent is equals to 10.

This window function is described as:

HAMM(n) = 0.54 - 0.46 cos[2 Pi n/(N-1)] n= 0,1..N-1 and N = 1024 in this case.

Software Implementation: SPECTRUM.EXE (appendix A.13)

This program calculates the Fast Fourier Transform of K non-overlapping intervals of the EMG signals. Each interval is 1024 points long (1.024 secs.) and calculates the average of the K frequency spectral densities, each of them 512 points long. The frequency resolution is 1 Hz, and the maximum frequency represented is 500 Hz.

Calling format: SPECTRUM DATA.EMG DATA.FFT TEXT_INFO

Where DATA.EMG is the file containing the processed EMG signal, DATA.FFT is the file containing the average of K PSD's (512 integer numbers long), and TEXT_INFO is a string for user's reference.

Program Execution:

To obtain the average PSD and the parameters mentioned above the program executes the following steps:

1. The main function opens source and destination files, and calls subroutine CALC_FFT.

2. The subroutine CALC_FFT calculates the number of K intervals (number of FFT's) to be calculated and averaged.

3. CALC_FFT reads a 1024 integer points sequence, it is converted to floating point format (INT2FP) and written in the complex structure array BUFF_FFT.

4. CALC_FFT modifies the sequence with the HAMM window function.

5. Then CALC_FFT performs a 1024 points FFT with the function RFFT, that outputs a sequence of complex numbers into the BUFF_FFT array.

6. This sequence is converted to polar form by the function POLAR (87FFT Microway Inc.) and the real components are converted to integer format and written into a temporary file.

7. After the EOF is reached, the programs calculates the average of sequences 512 points long from the temporary file and outputs the averaged sequence into the DATA.FFT file.

8. Once the averaged PSD is calculated, the program calculates the centroid frequency, Hi/Lo ratio, and total power.

Software Implementation: PLOTFFT.EXE (appendix A.14)

The 512 points integer sequence representing the normalized FFT of the EMG signals is the power spectrum of the signals from 0 to 500 Hz. To obtain visual information that is very useful for the researcher, this system is provided with the program PLOTFFT.EXE that displays the periodogram of the PSD. Figures 6.3.2 a,b, and c show the output display of this program for the PSD analysis of three signals.

Calling format: PLOTFFT DATA.FFT TEXT_INFO

The passing parameters for this program are the same as for the program SPECTRUM.EXE that calculates the PSD.

The software uses the GSSCGI functions for the graphics, and it provides features as change in display gain, and frequency information.

Hardcopy utility:

A modified version of PLOTFFT.EXE is PRINTFFT.EXE that is executed after the display output is redirected to the IBMGPR printer device, and prints the PSD of the EMG signals.

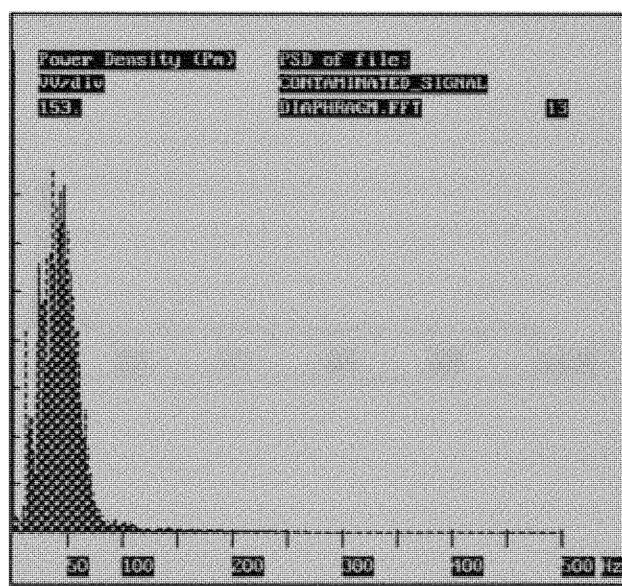


Figure 6.3.2-a
Power spectral density of the diaphragm EMG signal contaminated by EKG artifact and 60 Hz noise. The EMG signal spectrum is overlapped by the spectrum of the EKG artifact higher in amplitude.

Figure 6.3.2-b

PSD of diaphragm EMG signal after EKG artifact removal, low frequency and 60 Hz filtering.

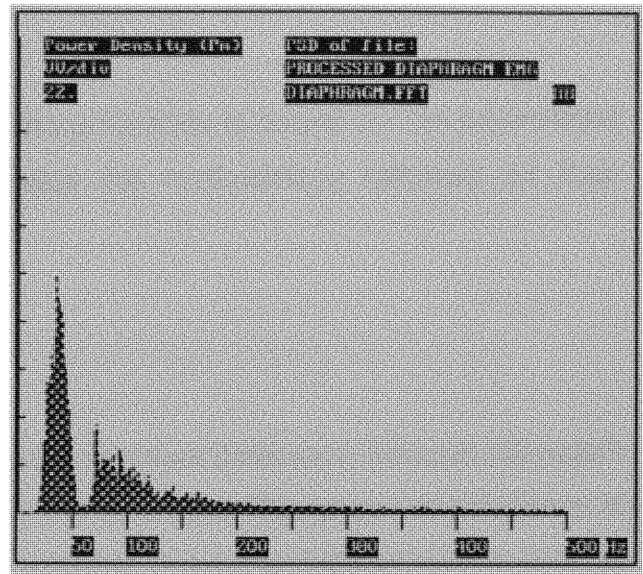
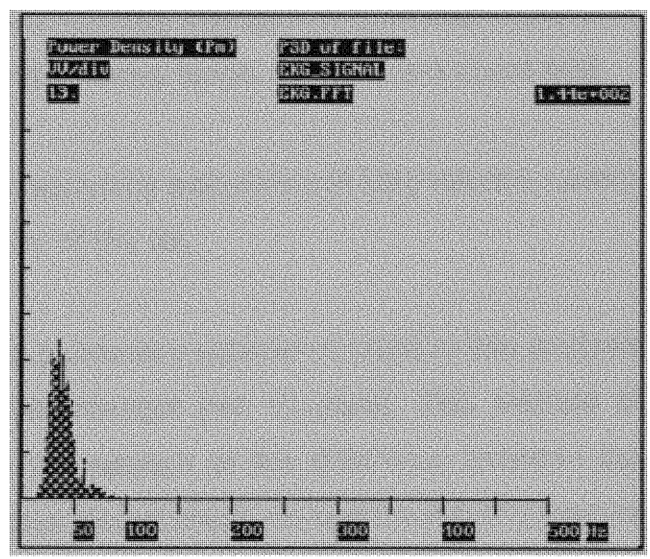


FIgure 6.3.2-c

Spectrum of the EKG signal after it was bandpass filtered. This signal has similar distribution as the contaminated EMG signal.



6.4 LUNG MECHANICS ANALYSIS

The analysis of the lung mechanical characteristics is based on the parameters described next. This description also includes the methodology followed for their calculation.

Total Pulmonary Resistance: [$cmH_2O/L/s$]

The pulmonary resistance is defined as the pressure difference between the mouth pressure and the pleural pressure divided by the flow rate. This pressure difference is required to overcome the resistance of the conducting airways and the viscous properties of the lungs. It is measured in centimeters of water per liter per second. It is called total pulmonary resistance because its value encompasses changes in both inspiratory and expiratory phases of the breathing cycle, and includes lungs, airways and chest wall.

Total pulmonary resistance can be increased by a decrease in lung volume, airway narrowing, or a change in density or viscosity of the air.

The method of calculation of the total pulmonary resistance used in this analysis is the two-point method of Mead and Whittenberger [ref 16-grant]. This method of obtaining the total resistance is based on the relation of the difference between the esophageal pressure (PE), which approximates pleural pressure, and the mouth pressure (PM) with the amount of air flowing in and out of the lungs at points of half tidal volume (see sections 2.2.1 and 5.2.3) since it is important to compare values at equivalent phases of the breathing cycle. As it is shown:

$$\text{Tot R} = \frac{(\text{PEinsp} - \text{PEexp}) - (\text{PMinsp} - \text{PMexp})}{\text{FLOWinsp} - \text{FLOWexp}}$$

(these parameters were measured at points of half tidal volume)

Inspiratory and Expiratory Resistance: [cmH20/L/s]

The airway resistance is calculated separately during inspiration and expiration. The method of calculation is based on the same pressure-flow relation explained previously, and considers separately the pressure and flow points at half inspiratory and expiratory tidal volume points [ref 16-grant].

$$\text{Insp R} = \frac{\text{PEinsp - PMinsp}}{\text{FLOWinsp}} \qquad \text{Exp R} = \frac{\text{PEexp - PMexp}}{\text{FLOWexp}}$$

(measured at point of half tidal volume)

Compliance: [ml/cmH20]

Compliance is defined as the volume change per unit pressure change. The compliance or elasticity of the lung can be affected by various diseases, when the lung remains unventilated during periods of time (apneas), and when the pulmonary venous pressure increases engorging the lung with blood.

The method used for the calculation of compliance is based on the relationship between the maximum tidal volume with the esophageal pressure at the point of maximum tidal volume during each breath.

$$C = \frac{\text{VT max (delta)}}{\text{PE (delta) at max VT}}$$

It is measured as milliliters per centimeter of water.

Tidal Volume: [ml]

Tidal volume (VT) is the volume of gas inspired or expired during each respiratory cycle. It is obtained by the integration of the air flow measured with spirometry and is counted in milliliters. This integration uses the trapezoid area method described earlier.

Timing Information: [msec]

The timing information about each breath is based on the intervals mentioned earlier: Ttot, Ti, and Te.

Breathing Frequency: [breaths/min]

The breathing frequency (f) gives information about the respiratory rate of the subject. It is the reciprocal of the total time (Ttot), measured in breaths per minute.

Minute Ventilation: [ml/min]

Defined as the volume of air breathed in and out during one minute time. It is the product of the tidal volume (VT) and the breathing frequency (f), averaged from several breaths.

Air Flow: [L/s]

The maximum volume displaced in and out the lungs during one second, both in inspiration and expiration, is given to the user as instantaneous values at half tidal volume by the system.

Esophageal and Mouth Delta Pressures: [cmH20]

The delta pressure is calculated as the difference between the maximum negative pressure and the maximum positive pressure for both PE and PM. This value is also given in centimeters of water.

Software Implementation: PLOTMEC.EXE (appendix A.10)

The program PLOTMEC.EXE performs the calculation of the respiratory mechanics parameters described above. This program as well as the program PLOTMTA.EXE from the previous section uses the graphics functions provided by the GSSCGI graphics development tools.

For signal display and event markers,the software uses the same functions as the previous programs. The algorithm of the program is basically the same as the program PLOTMTA, except for the software methods used for calculation and calibration of the signals.

Calling format: PLOTMEC FILE.MEC FILE.CAL TEXT_INFO

The file FILE.MEC contains the following signals: Flow, esophageal pressure, and mouth pressure. The file FILE.CAL contains the calibrating signals for each of the signals respectively, and TEXT_INFO is information that the operator gives to the program for report generation.

Signal Calibration:

Calibrating the signals is basically obtaining the calibration factor that would translate numeric to physical values.

A special calibration (.CAL) file is collected during the data acquisition procedure. This file contains the values used for each of the respiratory signals recorded (flow, PM and PM). These calibration values are obtained by applying known amounts of flow and pressure to the transducers, and conditioned by the Gould amplifiers with the same sensitivity levels used during the study.

The calibration files are divided into two files, one containing zero values (.ZER) and another containing the pre-established values (.VAL). The values used are 10 cmH20 for the pressure signals and 3 L/min for the flow signal.

Program Execution:

The following are the events which happen during program execution:

1. The program prepares the filenames used for the file handling routine MAKEFOUR.EXE that is used if the file FILE.MEC does not exist.

2. The calibration factors are obtained by the subroutine OBTAIN_CALIBRATION, that reads the files .VAL and .ZER, obtaining mean values for each of them, and the reciprocal of the difference between means for each channel is the calibration factor.

3. The program reduces the artificially the sampling frequency of the four signals from 1000 Hz to 100 Hz executing the program FREQS100.EXE as a child process. THis program extract one out of every ten points in time from the original sequences reducing the sampling interval of the resultant signal.

4. Sets up the graphics displaying workstation and functions.

5. The subroutine LINES is called by the main function to display the signals on the screen. This subroutine calls then the subroutine DETECT_BREATHS.

6. DETECT_BREATHS obtains first the points of beginning of inspiration and beginning of expiration in the flow signal, as it was explained in the previous section (program PLOTMTA.EXE). It also obtains the points of maximum inspiratory and expiratory flow.

7. Once M points of onset of inspiration have been detected, the number of breaths is computed as N = M - 1. These points are used as zero reference for the esophageal and mouth pressure signals, and used for offset extraction.

8. This subroutine then integrates the flow signal and obtains the tidal volume for each breath. After the tidal volume is obtained, it finds the maximum value in the tidal volume, and the points of half tidal volume occurrence during inspiration and expiration. With the information available, compliance and tidal volume are now calculated.

9. The points of half tidal volume are used for the calculation of the airway resistance parameters.

The program monitors the keyboard for operator commands such as:

. Select breaths to be included in the mean results
. Increase displaying gains
. Signal scrolling
. Flow signal smoothing function

During program execution the information about the breath marked in the screen as #1 is displayed in the screen (see figure 6.4):

. Total, inspiratory and expiratory resistance
. Compliance
. Tidal volume
. Timing information from beginning of file
. Information passed by the operator
. Graphics display of timing and magnitude of the parameters mentioned

The rest of the parameters are not displayed on the screen, but the user is provided with a written report for each selected breath and a report with the mean values for all the breaths selected.

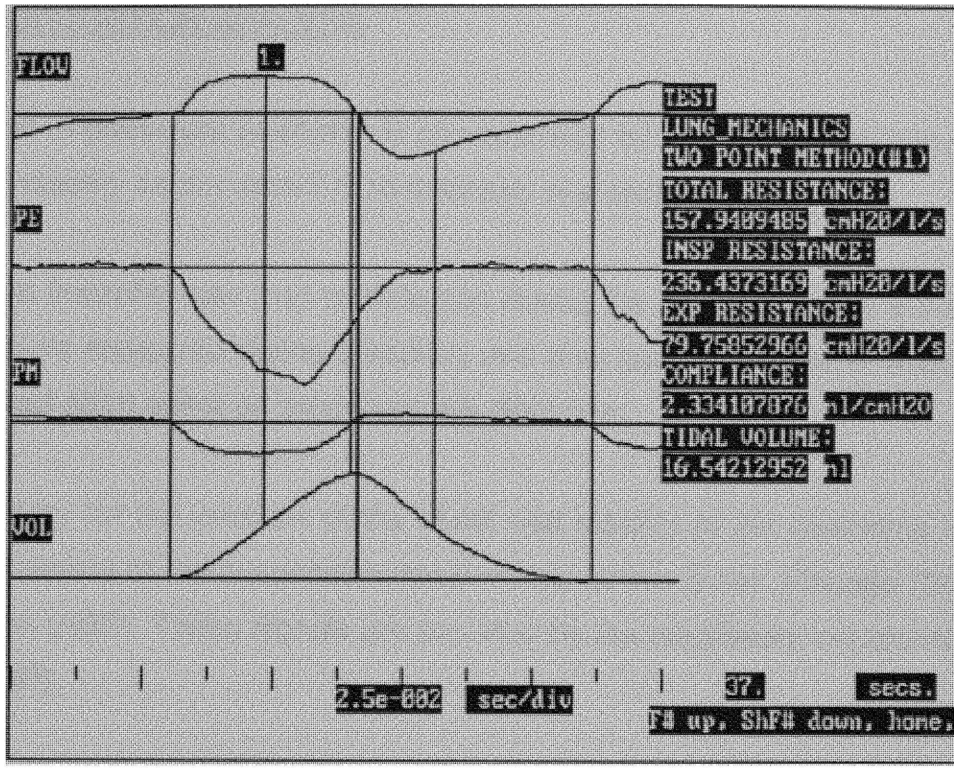


Figure 6.4

Program PLOTMEC.EXE output screen. The vertical lines mark breath timing, and points of half tidal volume for each breath. The results displayed are calculated for the breath marked as "1".

6.5 CHEST WALL DISTORTION ANALYSIS

The chest wall of premature newborns is characterized by excessive compliance plus relative lack of stability as compared to term infants and older children.

The chest wall instability can lead to distortion during the breathing cycle which may affect the pressure measured during breathing and also lead to inspiratory effort, because of the wasted effort spent on producing this distortion.

The methodology for the analysis of the chest wall stability in the newborns is based on the measurement of the thoracoabdominal motion at two levels: Ribcage (RC or RIB) and abdomen (ABD). These signals are provided by the respiratory inductance plethysmography (RIP).

The ribcage and abdomen motion signals were previously calibrated by the RIP, using its Qualitative Diagnostic method [12] estimating the relative contribution of the RC and ABD to the tidal volume.

To provide clinical research utility to this measurements it is necessary to analyze the waveforms and obtain numerical means that would represent levels of distortion in the chestwall.

To measure the levels of chestwall distortion, the system uses two methods of waveform analysis: Total compartmental displacement / Tidal volume ratio, and the difference in phase shift between the two waveforms (RIB and ABD).

Total Compartmental Displacement / Tidal Volume Ratio: (TCD/VT)

The ribcage and abdomen excursions during breathing, represent outward (positive) or inward (negative) displacements.

During regular breathing and under no chest wall distortion condition, the individual displacement of the ribcage and abdomen are synchronized, that is, they have outward and inward displacement simultaneously. If the newborn's chestwall is distorting, the individual displacements of the two compartments are opposite in direction(abdomen has outward movement, while ribcage has inward movement).

The sequence resulting of the sum point by point of the two waveforms is a representation of the tidal volume (VT) during each breathing cycle is called SUM, representing the volume of air inspired or expired. The tidal volume VT parameter is obtained from the summation of the SUM sequence over the inspiratory interval.

The total compartmental displacement (TCD) of the chestwall is defined total movement inward and outward of both compartments. this parameter is the summation of the absolute values of the SUM sequence over the same interval [12].

In the absence of chestwall distortion, the ratio TCD/VT is the unity, and it increases if the distortion level is greater.

Phase Shift Between Ribcage and Abdomen Displacements:

The difference in timing between the two waveforms is accentuated in the presence of chestwall distortion, that is, the abdomen displacement motion is followed by the ribcage displacement. The waveforms show that the ribcage signal (RIB) is still decreasing while the abdomen signal (ABD) is already increasing.

This difference in phase is measured in degrees of distortion, which is the relation between the lag time in the ribcage signal, measured from the beginning of the breathing cycle (when it becomes negative) to the point it starts rising above the zero level, with the total duration of the cycle. Where the lag time represents a fraction of the 360 degrees cycle.

This relationship is based on the assumption that the ribcage compartment is filled with air only after the displacement is outward, that is the RIB signal becomes positive after a negative period.

Software Implementation: PLOTDIS.EXE (appendix A.11)

The calculation of the phase shift and TCD/VT ratio on the chest wall signals is performed by the program PLOTDIS.EXE. This program is based on the methodology described above, and provides graphics displaying of the events happening on the chestwall, as well as obtaining numerical information of those parameters.

This program was implemented using the same graphics functions used in the programs described in previous sections.

Calling format: PLOTDIS DATA.DIS TEXT_INFO

DATA.DIS is a file created by the file handling routine MAKEFOUR.EXE that includes the flow, ribcage and abdomen signals. TEXT_INFO is information the user passes to the program identification.

Program execution and procedures:

The program calculates the chestwall distortion parameters with the following steps:

1. The main function of the program sets up the graphics workstations, and reduces the sampling interval (FREQS100.EXE).

2. The subroutine LINES is called and plots the signal on the screen, then as in the previous programs, the subroutine DETECT_BREATHS obtains timing information about the breathing cycle (Ttot, Ti, and Te).

3. The points in time where inspiration begins are taken as zero reference points for the ribcage (RIB) and abdomen (ABD) signals, eliminating any offset level.

4. Once the offsets are eliminated the SUM signal is computed as the addition point by point of the RIB and ABD signals.

5. The subroutine DETECT_BREATHS detects the points of occurrence of the maximum positive (array MAX_RC_POINTS) and negative (array MIN_RC_POINTS) values in the RIB signal, as well as the point where it returns to zero baseline (array ZERO_RC_POINTS) after the negative incursion (distortion).

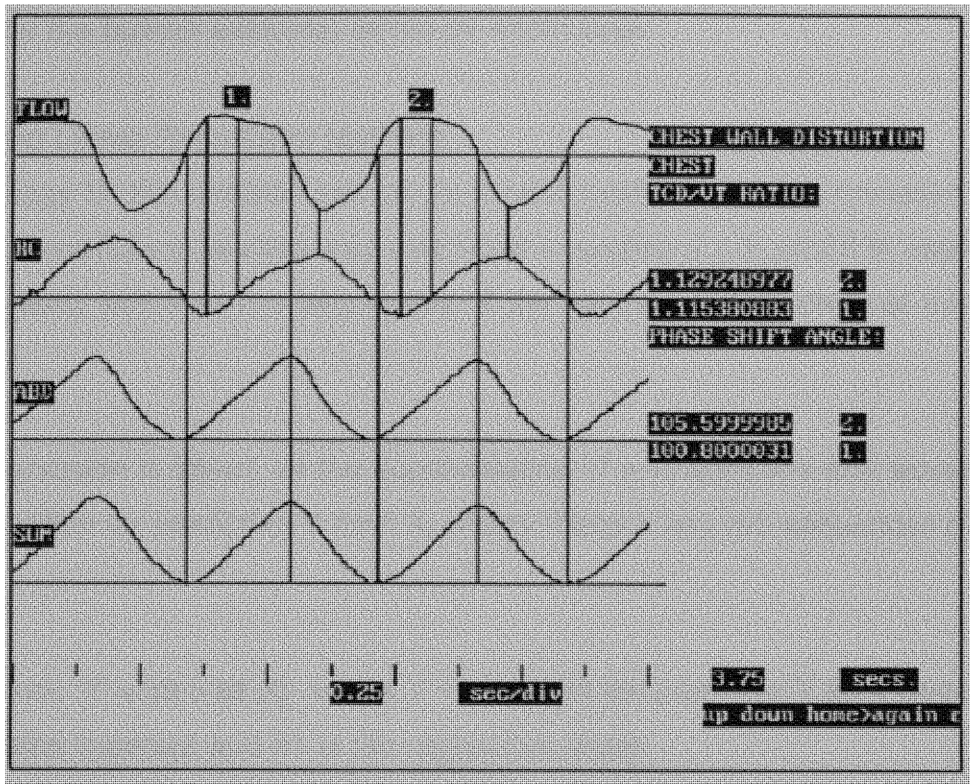
6. This subroutine calculates the summation of the absolute values of the SUM signal over the inspiratory interval to obtain TCD, and obtains the VT as the summation of the magnitude and sign values of the same sequence.

7. The phase shift is calculated as the division of the lag time calculated as ZERO_RC_POINTS - INIT_INSP by the total breath time Ttot, and multiplied by 360 degrees.

The user selects specific breaths to obtain average distortion information. And the distortion parameters per breath are shown in the screen (see figure 6.5).

Hardcopy Utility:

The program PRINTDIS.EXE is a modified version of the program PLOTDIS.EXE, that outputs a hardcopy of the four signals, with the distortion parameters information. It is accomplished by the redirection of the device graphics output to the IBMGPR driver.



^Figure 6.5

Chest wall distortion analysis program output screen. The vertical lines mark the points of maximum inward motion of the ribcage, and the point where it returns to its normal position.

6.5 ANALYSIS OF TWO RESPIRATORY PRESSURES SIGNALS

The clinical research of the pulmonary functions on the newborn studies the difference in magnitude and waveform shape of pressure signals measured at different levels of the esophagus and the upper airway.

The comparison of two waveforms that are very similar between them in timing, magnitude and shape, which is the case of pressure measurements of the same source, at different points, require the application of special methods to detect these similarities or differences that for the human examiner would be impossible.

Cross Correlation and Linear Regression Methods:

The methodology for this analysis include the Linear regression, and Correlation. The cross correlation coefficient function was described in section 5.1.3, and it is used to measure the similarity between two sequences. If the sequences are identical this coefficient is unity.

The linear regression is based on the approximation of a straight line that would fit as a first approximation for predicting Y from X, where Y and X are two sequences of size n.

In equation form: $Y' = b0 + b1\ Xj$ where Y' represents the predicted Y for a given X , b0 is the regression intercept and b1 is the regression slope.

The method to obtain b0 and b1 is the known as the least squares method. The values for b0 and b1 are determined from the data sequences in such a way as to minimize the sum of squares of the deviations of each Y from its predicted value Y'.

The slope is defined as:

$$b1 = \frac{\text{Covariance } XY}{\text{Var } X} = \frac{\text{Sum } XY - \text{Sum } X * \text{Sum } Y / n}{\text{Sum } XX - \text{Sum } X * \text{Sum } X / n}$$

and the intercept is defined as:

$$b0 = \text{Mean } Y - b1 * \text{Mean } X$$

The analysis of the two waveform pressures uses the value of the slope as a proportion factor between the two pressures, giving an idea of the difference in magnitude levels.

Software implementation: PLOTPE.EXE (appendix A.12)

This program performs the evaluation of similarities between two signals (in this case the two pressures) using a third signal (flow) as a timing reference.

This program was written in C language, and uses graphics functions of the library MCLCGI.LIB from GSSCGI tool kit.

Calling format: PLOTPE DATA.PES TEXT_INFO

The file DATA.PES is a file containing three signals: Flow, signal pressure 1, and signal pressure 2.

Program execution:

1. The main function of the program opens the file for binary buffered reading, sets up the graphic workstation.

2. Subroutine LINES plots the signals, and calls DETECT_BREATHS.

3. DETECT_BREATHS obtains breathing timing information, and uses it to find zero pressure baseline levels.

4. DETECT_BREATHS also plots the two pressure signals in XY coordinates (PE1>X and PE2>Y). It also calculates maximum positive and negative pressure values and the calls the subroutine SLOPE_REGRESSION.

5. SLOPE_REGRESSION calculates the linear regression slope interval and the cross correlation coefficient of the pressure sequences for each breath and displays the information of breath #1 in screen (see figure 6.6).

The program displays the following information:

. Slope of the linear regression

. Cross correlation coefficient

. Maximum positive and negative pressures

. Timing information

Hardcopy utility:

As explained in previous sections, the hardcopy utility is provided by the redirection of the graphics output to the printer when the program PRINTPE.EXE is executed. The program PRINTPE.EXE is a modification of PLOTPE.EXE.

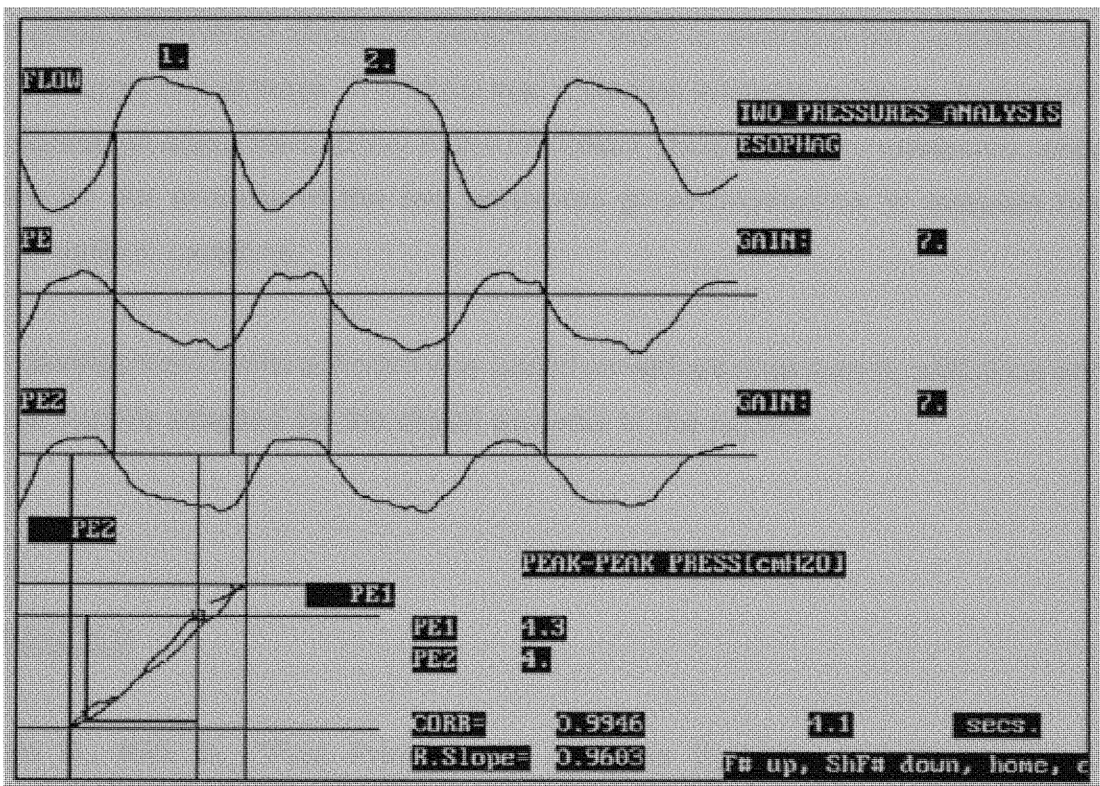


Figure 6.6

This figure is the screen display of the program PLOTPE.EXE. At the bottom left, the xy plot of the two pressure signals is shown, the lines mark the points 6 maximum pressures (positive and negative), beginning of inspiration (square), and beginning of expiration.

6.7 SUMMARY

This microcomputer based system for the study of the respiratory function in premature infants provides independent analysis software for each of the components of the respiratory system.

The muscle activity analysis provides information about magnitude, timing and frequency contents of the muscles during the breathing cycle. The analysis of the lung mechanics gives information about the parameters used to evaluate the condition and function of the lungs and airways. The chest wall distortion analysis software detects and presents information about the stability of the compliant chestwall of the newborns, while the software for the analysis of the two pressures gives information to evaluate this parameter in various locations.

The development of this software was based on various engineering, statistical and physiological models adapted for the digital analysis of the signals.

6.8 REFERENCES

[1] Strum R., Kirk D., "First Principles of Discrete Systems and Digital Signal Processing", Addison-Wesley Publishing Company, Inc., 1988.

[2] DeFatta D., Lucas J., Hodgkiss W., "Digital Signal Processing: A System Design Approach", John Wiley & Sons, Inc., 1988.

[3] Stearns S., David R., "Signal Processing Algorithms",Prentice-Hall, Inc., 1988.

[5] Bruce E., Goldman M., Mead J., "A digital Computer Technique for Analyzing

Respiratory Muscle EMG's", Journal of Applied Physiology, 1984.

[6] Bloch R., "Subtraction of Electrocardiographic Signal from Respiratory Electromyogram", Journal of Applied Physiology, 1983.

[7] Cobbold R., "transducers for Biomedical Measurements: Principles and Applications", John Wiley, New York, 1972.

[8] Levine S., Gillen J., Weiser P., Gillen M., Kwatny E., "Description and Validation of an ECG removal procedure for EMGdi power spectrum analysis", Journal of Applied Physiology, 1986.

[9] Gottlieb G., Agarwal C., "Filtering of Electromyographic Signals", American Journal of Physical Medicine, 1970.

[10] 87FFT Reference Manual, Micro Way Inc., 1985.

[11] West J., "Respiratory Physiology", Williams & Wilkins, 1990.

[12] Sackner M., Krieger B., "Noninvasive Respiratory Monitoring", Marcel Dekker, Inc., 1989.

[13] Ruppel G., "Manual of Pulmonary Function Testing", The C.V. Mosby Company, 1979.

[14] Gross D., Grassino A., Ross D., Macklem P., "Electromyogram Pattern of Diaphragmatic Fatigue", Journal of Applied Physiology, 1979.

[15] Scheitzer T., Fitzgerald J., Bowden J., Lynne Davies P., "Spectral Analysis of Human Respiratory Muscle Electromyograms", Journal of Applied Physiology,

1979.

[16] Hicks C., "Fundamental Concepts in the Design of Experiments", Saunders College Publishing, 1982.

[17] GSSCGI Device Driver Supplement & GSSCGI Programers Guide, Graphics Software Systems, Inc., 1987.

CHAPTER VII

SYSTEM ENHANCEMENT AND DISCUSSION

7.1 INTRODUCTION AND DISCUSSION

The development of this system combined areas of analog and digital electronics, computer and software engineering, and human physiology.

The main effort in the development of the project was devoted to the implementation of the software for data acquisition, signal processing, and analysis. The result is a portable system, meaning it is a hardware independent system. Figure 7.1 shows a block diagram containing the software components of this system according to its function.

The software for signal processing and analysis, even though it was originally created for this specific application, can be applied in many different areas with further custome procedures.

The algorithms and techniques used for signal processing were specifically designed or adapted for this purpose. The analysis algorithms and computations are adaptations of pre-established physiological relations, and they are based on specific research aims of the medical counterpart.

An intentional feature of the system is that it is an open ended system, that is, its development and applicability do not end with the material presented in this document.

The use of this system provides the clinical researcher the ability to accomplish specific study aims that in time should provide a better knowledge of the respiratory system in the premature infant.

7.2 CARDIOVASCULAR SIGNAL ANALYSIS IN ANIMAL RESEARCH

As part of the system enhancements, and as result of the portability of the system. There is a parallel system being developed for the study of the respiratory and cardiovascular functions in newborn animals.

The software developed for the analysis of the pulmonary functions (PLOTMEC.EXE), chest wall distortion (PLOTDIS.EXE), and comparison of two pressure signals (PLOTPE.EXE) is actually being used for the study of these parameters in piglets and rabbits. These programs were customized to the specific breathing patterns of these animals, with some modifications in the source codes.

Application software being developed in present time for the analysis of cardiovascular and ventilatory parameters, such as:

. Minute ventilation

. Pulmonary artery pressure measurements

. Blood pressure measurements

7.3 COMPRESSED SPECTRAL ARRAY OF EMG SIGNALS

The power spectral density analysis of the EMG signals have been analyzed using only the average of several intervals.

A new technique for the analysis of spectral densities is known as Compressed Spectral Density Array, which is a tri-dimensional array of the frequency spectrum changes in time. This technique is useful to measure the variation in time of the individual frequency contents of the signal.

The graphics software required for this purpose would show a tri-dimensional view of several PSD's, with time as the third variable.

7.4 CARDIAC SINUS ARRHYTHMIA

The heart rate is monitored as the reciprocal of the time between two QRS complexes in the EKG signal. The heart rate increases and decreases during the various phases of the respiratory cycle, and during deep respiration this changes are greater.

During each respiratory cycle, the negative intrapleural pressure increases and decreases, increasing and decreasing the effective pressure in the veins of the chest. The relationship between the heart rate variability and the breathing pattern will be correlated by the algorithm implemented.

7.5 INTERFACING WITH ULTRASOUND EQUIPMENT

One of the most interesting future enhancements of this system is the analysis of signals obtained with ultrasound equipment based on the Doppler effect.

The ultrasound equipment is actually used for different non-invasive measurements such as retinal blood flow, and cardiovascular flow.

The interfacing of the computer system with this sophisticated devices will be part of the work planned for development and implementation with the software necessary for the analysis of this information.

7.6 SUMMARY

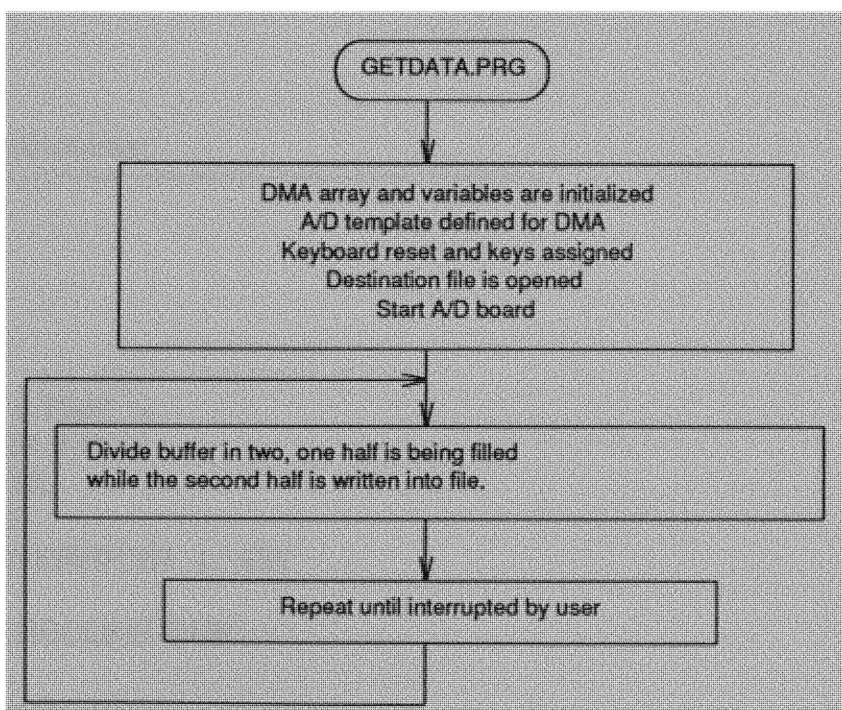
The application of this system to different areas of clinical research requires some enhancement and modification. The immediate enhancement and applicability of this system is focused on the development of software for animal research, new techniques for PSD analysis, cardiac sinus arrhythmia, and interfacing of the system with ultrasound equipment for non-invasive measurements.

APPENDIX A

SOFTWARE ROUTINES

A.1 Data Acquisition: Program GETDATA.PRG

The program GETDATA.PRG is running under ASYST Scientific System Environment. It collects 7 channels DMA, with 1000 Hz frequency sampling per channel. The following are some of the most important subroutines of this program, preceded by figure A.1 that shows its flow diagram.



Definition of DMA buffer array and A/D template for DMA:

```
*******************************************************************
DIM[ 14000 ] DMA.ARRAY DATA.BUF
DT2821
0 6 A/D.TEMPLATE ad7in
DATA.BUF CYCLIC DMA.TEMPLATE.BUFFER
A/D.INIT
*******************************************************************
```

Start Acquisition Procedure:

```
*******************************************************************
: START.ACQ
AD7IN
1000. 7000. / CONVERSION.DELAY
A/D.INIT
A/D.IN>ARRAY(DMA)
;
*******************************************************************
```

Data collection subroutine:

```
*******************************************************************
: COLLECT
   INTEGER
   BEGIN
      0 TIME :=
      BEGIN
         1 TIME + TIME :=
         ?BUFFER.INDEX
      7000 > UNTIL
      TIME 1 = IF
         ." A/D OVERRRUN  1." CR
      THEN
      CURRENT.RECORD DUP
      DATA.BUF
      SUB[ 1 , 7000 , 1 ]
      RANDOM.PUT                    \ WRITES DATA
      DUP 30 - 60 / DUP . 60 * - ." >>> " .
```

```
\ TIME . ?BUFFER.INDEX .
CR
1 + CURRENT.RECORD :=
0 TIME :=
BEGIN
   1 TIME + TIME :=
   ?BUFFER.INDEX
7000 < UNTIL
TIME 1 = IF
   ." A/D OVERRRUN  2." BELL CR
THEN
CURRENT.RECORD DUP
DATA.BUF SUB[ 7001 , 7000 , 1 ]
RANDOM.PUT                  \ WRITES DATA
DUP 30 - 60 / DUP . 60 * - ." >>> " .
\ TIME . ?BUFFER.INDEX .
CR          \ PRINT RECORD TIME
1 + CURRENT.RECORD :=
?KEY
UNTIL
   RANDOM.CLOSE
;
*****************************************************************
```

A.2 File Handling Program. IMPORT.EXE

IMPORT.EXE translates files from CODAS format to binary format. The following is the subroutine that performs the data translation.

Disk space availability and file opening routine:

```
*****************************************************************
unsigned long total_space, free_space, bytes_per_cluster;
        struct diskfree_t dfinfo;
        struct stat info;
        if(_dos_getdiskfree (4, &dfinfo) !=0)
        {
                printf("error disk space");
                exit(1);
        }
        bytes_per_cluster = dfinfo.sectors_per_cluster * dfinfo.bytes_per_sector;
        free_space = dfinfo.avail_clusters * bytes_per_cluster;
```

```
if ((dat_file_hdl = fopen(argv[argc-2],"rb")) == NULL)  /* file is */
                                                   /* opened in binary mode */
{
        printf(stderr," couldn't open file %s\n",argv[argc-1]);
        exit(1);
}
if ((dat_file_out = fopen(argv[argc-1],"wb")) == NULL)  /* file is */
                                                   /* opened in binary mode */
{
        printf(stderr,"%s couldn't open file %s\n",argv[0],
          argv[argc-1]);
        exit(1);
}
if(fstat(fileno(dat_file_hdl), &info) !=0)
{
        printf("error file size");
        exit(1);
}
if(info.st_size > free_space)
{
        printf("File size exceeds free space");
        exit(1);
}
```
```
**************************************************************
```

Data translation routine:

```
**************************************************************
        _clearscreen(_GCLEARSCREEN);
        printf("\n    Please wait....... ");
        fseek(dat_file_hdl,+1156,SEEK_CUR);
        ch = 0;
        while(feof(dat_file_hdl)==0)
        {
        ch +=1;
        numread = fread(buff_first_get,sizeof(int),8000,dat_file_hdl);
        for (i = 0; i < numread; i++)
                buff_write[i] = buff_first_get[i] >> 4;
        numwritten = fwrite(buff_write,sizeof(int),numread,dat_file_out);
        }
        printf("\n %d  data buffers ",ch);
        printf("\n information  converted ");
**************************************************************
```

Obtaining tidal volume signal by executing two child processes:

```
*****************************************************************
   printf("\nFile %s   FLOW signal in ",argv[argc-1]);
  spawnl(P_WAIT,"one-ch8","one-ch8",argv[argc-1],"x1.vol",NULL);
  spawnl(P_WAIT,"integ","integ","x1.vol","x2.vol",NULL);
  spawnl(P_WAIT,"oneto8","oneto8",argv[argc-1],"x2.vol","x1.vol",NULL);
  remove(argv[argc-1]);
  rename("x1.vol",argv[argc-1]);
*****************************************************************
```

A.3 Signal Displaying: SELECT.EXE

SELECT.EXE drives the WFS200PC waveform scroller card for signal displaying of eight channels.

Arrays for WFS200PC card setup:

```
*****************************************************************
static int setary[24] =   {8,        /* channels enabled      */
                    4,4,4,5,3,3,3,8,    /* scale factors      */
                    0,0,0,0,0,0,0,0,    /* offsets            */
                    1,                  /* trigger source     */
                    0,                  /* trigger level      */
                    0,                  /* trigger slope      */
                    10,                 /* display format     */
                    1,                  /* scroll mode        */
                    0,                  /* horiz res          */
                    2};                 /* vertical res       */

static int    color[8] =    {4,6,5,7,4,6,5,7};    /* color */
*****************************************************************
```

Setup WFS200PC card and video graphics mode):

```
*****************************************************************
   _clearscreen(_GCLEARSCREEN);
  _setvideomode(_ERESCOLOR);
  setadd(0x308);  /* card address is HEX 308  */
```

```
    setup(setary);    /* configure card function  */
       tcolor(color);    /* colors per channel       */
       init();          /* initialize card          */
       blank();          /* clear                    */
*****************************************************************
```

Subroutine for signal scrolling:

```
*****************************************************************
     int i,span;
       ldiv_t clock;
       direct(di); /* direction right to left*/
       print_options();
       while(tstkey())
       {
              current_pos = ftell(dat_file_hdl);
              if (current_pos < 100)
              {
                     _settextposition(1,1);
                     printf("Begining of data, hit  RIGHT >      ");
                     goto noscroll;
              }
              if (current_pos >= (filesize - 100))
              {
                     _settextposition(1,1);
                     printf("End of data, hit HOME or < LEFT      ");
                     goto noscroll;
              }
              numread = fread(buff_first_get,sizeof(int),8,dat_file_hdl);
              tot_numread= ftell(dat_file_hdl);
              clock = ldiv(tot_numread,16000);
              _settextposition(22,70);
              printf("%ld sec",clock.quot);
              fseek(dat_file_hdl,+num_bytes,SEEK_CUR);
              for (i = 0; i < 8; i++)
                     {
                            plot(buff_first_get[i]); /* send infor to screen*/
                            delay();/* delay counter*/
                     }
     }
              noscroll: ;
```

```
}
*****************************************************************
```

A.4 Signal Processing: FIREMG.EXE

This program is the digital implementation of a FIR filter for EMG signal processing.

Filtering subroutine:

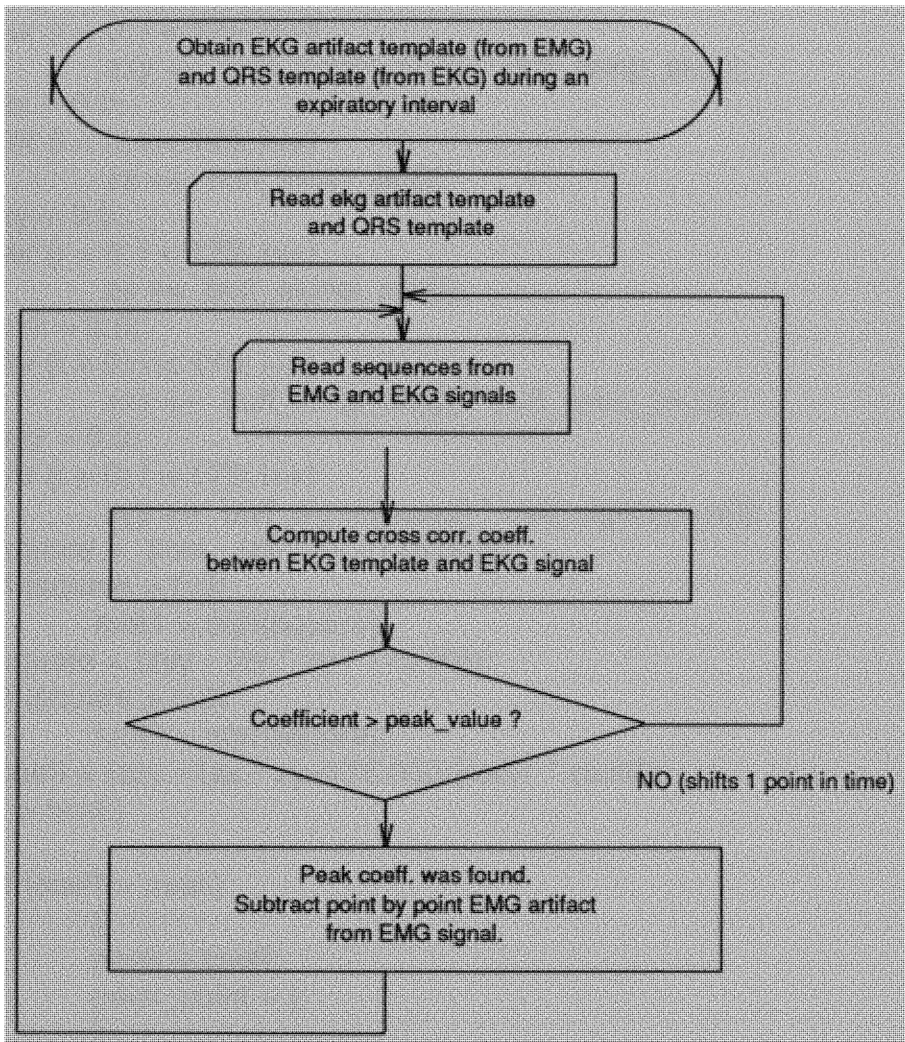
```
*****************************************************************
int n, i, j, norm, bits,p, stop;
        long m,filesize,k;
        double h,angle;
        static float win[129];
        static float fir[129];
        static float aux[129];
        static float xxx[129];
        static float yyy[129];
        static int buff_get_int[129];
        static int buff_out_int[129];
/* Define a pointer (r) to the real array.
  Note that the compiler will issue a warning message which can be
  safely ignored. */
        int *source,*dest;
        float *r,*rr;
        r = &xxx;
        dest = &buff_out_int;
        rr = &yyy;
        source  = &buff_get_int;
        bits = 16;
        n = 10;
        m = 129;
        filesize = fileno(dat_file_hdl);
        /* calc k intervals  */
```

```
        k = filesize / 258;
        printf("\n k = %ld intervals",k);
/* calc coefficients and apply Hamming window*/
        for(j = 0,h = 0.0; j < 129; j++,h++)
                {
                angle = 3.141592654 * (h - 64.00)/64.00;
        win[j] = 0.54 + .46*cos(angle);
                if ((j-64) != 0)
fir[j]=win[j] * (sin((h-64.0)* 0.345575191)-sin((h-64.0)*0.408407045)
                -sin((h-64.0)*0.125663706))/((h-64.00)*3.141592654);
                if ((j-64) == 0)
                fir[64] = 0.94* win[64];
                }
/* perform k reads, filter and send to file*/
for ( p = 0; p < k; p++)
{
        _settextposition(15,15);
        printf("Wait .. %d       ",p);
        gettime();
        /* read sequence and convert int to fp*/
        numread= fread(buff_get_int,sizeof(int),129,dat_file_hdl);
        for(j = 0; j< 129; j++)
            xxx[j] = buff_get_int[j];
        for(j = 0; j< 129; j++)
        {

                yyy[j] = 0.0;
                for(i = 0; i< 129; i++)
                        {
                        if ((j - i) >= 0)
                                {
                                yyy[j] += (fir[i] * xxx[j -i]);
                                }
                        if ((j - i) < 0)
                                {
                        yyy[j] += (fir[i] *  aux[129+(j-i)]);
                                }
                        }
        }
        for(j = 0; j< 129; j++)
                aux[j] = xxx[j];
        fp2int(rr,dest,&m,&bits);
        numwritten = fwrite(buff_out_int,sizeof(int),129,dat_file_flt);
*****************************************************************
```

A.5 Signal Processing: CLEANEMG.EXE

This program removes the EKG contaminant from the EMG signal. The following is the subroutine that performs the cross correlation between two sequences, keeps track of previous coefficients and determines the occurrence of a QRS complex when the peak coefficient is reached. Figure A5 shows the flow diagram of this program.



Extraction subroutine (implementation of cross correlation):

```
*****************************************************************
        float square;
        long m;
        int *source_signal,*dest_clean;
        float *dest_signal,*source_clean;
        int i,j,bits;
        dest_clean = &buff_signal_int;
```

```
source_clean = &buff_signal_out;
bits = 16;
m=16000;
for (j = 0; j < ((numread/2) - 175); j++)
{
        for (i = 0; i < 175; i++)
                        buff_signal_aux[i] = buff_signal_ekg[j+i];
        zxy = 0.0;
        for (i = 0; i < 175; i++)
                zxy +=  buff_signal_aux[i] * buff_temp_ekg[i];
        zx = 0.0;
        for (i = 0; i < 175; i++)
                zx +=  buff_temp_ekg[i];
        zy = 0.0;
        for (i = 0; i < 175; i++)
                zy +=  buff_signal_aux[i];
        zxx = 0.0;
        for (i = 0; i < 175; i++)
                zxx +=  buff_temp_ekg[i] * buff_temp_ekg[i];
        zyy = 0.0;
        for (i = 0; i < 175; i++)
                zyy +=  buff_signal_aux[i] * buff_signal_aux[i];
        square =  (zxx - (zx * zx / 175.00));
        if (square > 0.0)
                root1 = sqrt(square);
        square = (zyy - (zy * zy / 175.00));
        if (square > 0.0)
                root2 = sqrt(square);
        coeff = (zxy - (zx * zy /175.00))/(root1 * root2);
if (coeff > level)
        {
        if (coeff > peak)
                {
                peak = coeff;
                peak_point = j;
                }
        if (coeff < peak)
                {
                _settextposition(7,5);
                printf("\n Extract artifact coeff = %f  at
              point = %d   peak # %f",peak,peak_point,counter);
                check_artifact(peak_point);
                if(coeff_check>0.5)
                {
```

```
                    printf("\n Extracted at point %d ",peak_point);
                    counter +=1.0;

                            for (i = 0; i < 175; i++)
                            {
                buff_signal_emg_out[peak_point+i] =
                (buff_signal_emg[peak_point+i] - buff_temp_emg[i]);
                            }
                    }
                    }

            }
        if (coeff < level)
                peak = 0.0;

        }
        joint();
        fp2int(source_clean,dest_clean,&m,&bits);
}
****************************************************************
```

A.6 Signal Processing: MOVING.EXE

This program is the digital implementation of a digital Paynter filter for moving time averaging.

Calculating filter coefficients:

```
****************************************************************
int i,j;
        float denominator,interval,wp,w3,w2;
        /* averaging interval*/
        interval = 100;
        wp = 3.141592654 / interval;
        w3=wp*wp*wp;
        w2=wp*wp;
        denominator = (3.2 + 4.0*wp + 3.2*w2 + w3);
        aaa[0] = (wp + w3)/denominator;
        aaa[1] = (3.0*w3 - wp)/denominator;
```

```
        aaa[2] = aaa[1];
        aaa[3] = aaa[0];
        bbb[0] = 1.0;
        bbb[1] = (3.0*w3 + 3.2*w2 - 4.0*wp - 9.6)/denominator;
        bbb[2] = (3.0*w3 - 3.2*w2 - 4.0*wp + 9.6)/denominator;
        bbb[3] = (w3 - 3.2*w2 + 4.0*wp - 3.2)/denominator;
}
*********************************************************************
```

Filtering the signals:

```
*********************************************************************
    int i, j,norm, bits,stop;
        float aux,gain;
        long m;
        static float xxx[100];
        static float yyy[100];
        static int buff_get_int[100];
        static int buff_out_int[100];
/* Define a pointer (r) to the real array.
  Note that the compiler will issue a warning message which can be
  safely ignored. */
        int *source,*dest;
        float *r,*rr;
        r = &xxx;
        dest = &buff_out_int;
        rr = &yyy;
        source  = &buff_get_int;
        bits = 16;
        m = 100;
        for(j = 0; j < 2; j++)
                {
                initxxx[j] = 0.0;
                inityyy[j] = 0.0;
                }
        while (feof(dat_file_hdl) == 0)
        {
                numread= fread(buff_get_int,sizeof(int),100,dat_file_hdl);
                int2fp(source,r,&m,&bits);
                for(j = 0; j < 100; j++)
                {
                        yyy[j] = 0.0;
                        for(i = 0; i< 4; i++)
                        {
                        if ((j - i) >= 0)
```

```
                yyy[j] += (aaa[i] * xxx[j -i]);
            if ((j - i) < 0)
                yyy[j] += (aaa[i] * initxxx[3+j-i]);
            }
            for(i = 1; i< 4; i++)
            {
                if ((j - i) >= 0)
                yyy[j] -= (bbb[i] * yyy[j -i]);
                if ((j - i) < 0)
                yyy[j] -= (bbb[i] * inityyy[3 +j-i]);
            }
        }
        for(j = 97; j < 100; j++)
            {
            initxxx[j-97] = xxx[j];
            inityyy[j-97] = yyy[j];
            }
        fp2int(rr,dest,&m,&bits);
        numwritten = fwrite(buff_out_int,sizeof(int),100,dat_file_flt);
    }
}
****************************************************************
```

A.7 Signal Processing: INTEG.EXE

Digital integrator, integrates signals with 1 ms sampling period. Obtains the tidal volume from the flow signal. INTEG.EXE is a process executed by program IMPORT.EXE.

Integration subroutine:

```
****************************************************************
{
    int i, j,norm, bits,stop,k;
    float aux,gain;
    long m;
    static float xxx[100];
    static float yyy[100];
    static int buff_get_int[100];
    static int buff_out_int[100];
```

```
/* Define a pointer (r) to the real array.
   Note that the compiler will issue a warning message which can be
   safely ignored. */
        int *source,*dest;
        float *r,*rr;
        r = &xxx;
        dest = &buff_out_int;
        rr = &yyy;
        source  = &buff_get_int;
        gain = 1;
        bits = 16;
        m = 100;
        for(j = 0; j < 1; j++)
                {
                initxxx[j] = 0.0;
                inityyy[j] = 0.0;
                }

        while (feof(dat_file_hdl) == 0)
        {
                numread= fread(buff_get_int,sizeof(int),100,dat_file_hdl);
                int2fp(source,r,&m,&bits);
                for(j = 0; j < 100; j++)
                {
                        if ((j - 1) >= 0)
                                yyy[j] = (0.0005 * (xxx[j]+xxx[j-1])) + yyy[j-1];
                        if ((j - 1) < 0)
                                yyy[j] = (0.0005 * (xxx[j]+initxxx[0])) + inityyy[0];

                }
                inityyy[0] = yyy[99];
                initxxx[0] = xxx[99];
                fp2int(rr,dest,&m,&bits);
                numwritten = fwrite(buff_out_int,sizeof(int),100,dat_file_flt);
        }
}
****************************************************************
```

A.8 Signal Processing: SMOOTH.EXE

SMOOTH.EXE is a digital first order lowpass filter. The following subroutine calculates the filter coefficients, and the filtering itself is a variation of the filter implemented in the program MOVING.EXE section A.6.

Calculation of filter coefficients:

```
*******************************************************************
{
        int i,j;
        float wc,fs;
        fs = 10;
        wc=2* 3.141592654*fs;
        aaa[0] = wc/(wc+1.0);
        aaa[1] = wc/(wc+1.0);
        bbb[0] = 1.0;
        bbb[1] = (wc - 1.0)/(wc + 1.0);
}
*******************************************************************
```

A.9 Signal Analysis: PLOTMTA.EXE

This program is used for the analysis of the moving time averaged EMG signals. It handles software device drivers for graphics display. It plots four channels, with 2000 points per channel (2 seconds of activity). The following are some of the subroutines implemented in this program.

The signals are stored in the following array:

Flow : buffch[0]
Diaphragm EMG : buffch[1]
PCA EMG : buffch[2]
Genioglossus EMG : buffch[3]

Procedure to open graphics workstation:

```
*******************************************************************
        static int display[] = {1,1,1,1,1,1,1,1,1,1,1,
                                  'D','I','S','P','L','A','Y',' '};
        /* OPEN THE WORKSTATIONS */
        error = v_opnwk (display,&screen,screen_out);
        if (error == -1)
        {
                printf ("Error %d in  display  Open Ws",vq_error());
                exit (-1);
        }
        box[0] = 0;
        box[1] = 0;
```

```
        box[2] = screen_out[51];
        box[3] = screen_out[52];
        /* inquire the screen bitmap and bitmap size */
        vqd_bitmap (screen,&scrmap,box);
        v_clrwk (screen);
*****************************************************************
```

Plotting signals on the screen:

```
*****************************************************************

{
double value;
char string[20];
        int stop,i,j,width;
    vsl_type (screen,1);                    /*solid line*/
        for (i = 0; i<4000;i++)
        {
                xy[i] = 5*i;  /* Horizontal compression 5 pel's*/
                i++;
        }
        for (i = 1,j=0; i<4000;i++,j++)
        {
                        xy[i]=(gain[0] * buffch[0][j])+20000;
                        i++;
        }
                        vsl_color (screen,colors[0]);
                        stop = v_pline (screen,2000,xy);
        for (i = 1,j=0; i<4000;i++,j++)
        {
                        xy[i]=(gain[1] * buffch[1][j])+15000;
                        i++;
        }
                        vsl_color (screen,colors[1]);
                        stop = v_pline (screen,2000,xy); /*plot 2000 pts*/
        for (i = 1,j=0; i<4000;i++,j++)
        {
                        xy[i]=(gain[2] * buffch[2][j])+10000;
                        i++;
        }
                        vsl_color (screen,colors[2]);
                        stop = v_pline (screen,2000,xy);
        for (i = 1,j=0; i<4000;i++,j++)
        {
```

```
                    xy[i]=(gain[3] * buffch[3][j])+5000;
                    i++;
        }
                    vsl_color (screen,colors[3]);
                    stop = v_pline (screen,2000,xy);
                    if (stop == -1)
                    {
                    printf ("Error %d in  plotting  Open Ws",vq_error());
                    exit (-1);
                    }
        value = (ftell(dat_file_hdl)/(8*fs));
        gcvt(value,10,string);
        v_gtext (screen,22000,2000,string);
        v_gtext (screen,26000,2000," secs."); /* time from beg. of file*/
        value = 25/fs;
        gcvt(value,10,string);
        v_gtext (screen,10000,1500,string);
        v_gtext (screen,14000,1500," sec/div"); /*time per division*/
}
******************************************************************
```

Calculation of baseline noise level:

```
******************************************************************
{
        int i,j;
        int buffer_aux[2000];
        int slope1,slope2,peak;
        long mean_emg;
        float root;
        /* mean activity */
        for (i=0,mean_emg = 0;i<2000;i++)
                mean_emg += buffch[ch][i];
        mean_emg /= 2000.00;
    /* find minimum points*/
        for (i=750,j=0;i<1250;i++)
        {
                slope1 = buffch[ch][i] - buffch[ch][i-750];
                slope2 = buffch[ch][i+750] - buffch[ch][i];
                peak = slope1 * slope2;
                if ((peak < 0)&&(slope1<slope2)&&(buffch[ch][i] < mean_emg))
                {
                        buffer_aux[j]=buffch[ch][i];
```

```
                    j++;
                }
        }
        for (i=0;i<j;i++)
                mean_baseline[ch-1] += buffer_aux[i];
        if (j >1)
        {
        mean_baseline[ch-1] /=(j);
        }
}
****************************************************************
```

Detect onset of inspiration from the flow signal (buffch[0]) and number of breaths:

```
****************************************************************
        zero_crossing=1;
        slope = 0;
        number_breaths = 0;
        first_point=buffch[0][0];
        for (i=1;i<2000;i++)
        {
                second_point=buffch[0][i];
                zero_crossing = first_point * second_point;
                slope = second_point - first_point;
                if ((zero_crossing <= 0)&&(slope > 0))
                {
                xy2[0] = 10*i;          /* vertical line*/
                xy2[1] = 20000;
                xy2[2] = 10*i;
                xy2[3] = 5000;
                vsl_color (screen,5);
                vsl_type (screen,2);
                v_pline (screen,2,xy2);
                init_insp[number_breaths] = i;
                if(abs(second_point) > abs(first_point))
                        init_insp[number_breaths] = i-1;
                number_breaths ++;
                }
                slope = 0;
                first_point = second_point;
        }
        number_breaths --;
****************************************************************
```

```
Detect maximal point of emg activity*/
******************************************************************
        for (j=0;j<number_breaths;j++)
        {
              max_dia[j]=buffch[1][init_insp[j]];
              for (i = init_insp[j]+1; i < init_exp[j]; i++)
              {
                      if (buffch[1][i] >= max_dia[j])
                      {
                      max_dia[j] = buffch[1][i];
                      max_dia_points[j] = i;

                      }
              }
              xy2[0] = 10*max_dia_points[j];
              xy2[1] = (gain[1] * buffch[1][max_dia_points[j]])+15000;
              xy2[2] = 10*max_dia_points[j];
              xy2[3] = (gain[0] * buffch[0][max_dia_points[j]])+20000;
              vsl_color (screen,6);
              vsl_type (screen,1);
              v_pline (screen,2,xy2);
         }
******************************************************************


Detect on set and off set of EMG activity
******************************************************************
        for (j=0;j<number_breaths;j++)
        {
              i=max_dia_points[j];
              while (mean_baseline[0] < buffch[1][i])
                      i++;
              if (i<0) i=0;
              if (i>1999) i=1999;
              time_activity_dia[0][j] = i;
              xy2[0] = 10*i;
              xy2[1] = (gain[1] * buffch[1][i])+15000;
              xy2[2] = 10*i;
              xy2[3] = (gain[0] * buffch[0][i])+20000;
              vsl_color (screen,2);
              vsl_type (screen,1);
```

```
        v_pline (screen,2,xy2);
        i=max_dia_points[j];
        while (mean_baseline[0] < buffch[1][i])
                i--;
        if (i<0) i=0;
        if (i>1999) i=1999;
        time_activity_dia[1][j] = i;
        xy2[0] = 10*i;
        xy2[1] = (gain[1] * buffch[1][i])+15000;
        xy2[2] = 10*i;
        xy2[3] = (gain[0] * buffch[0][i])+20000;
        vsl_color (screen,2);
        vsl_type (screen,1);
        v_pline (screen,2,xy2);
*****************************************************************
```

A.10 Signal Analysis: PLOTMEC.EXE

Program implemented for the lung mechanics analysys. The setup, graphics displaying, and onset of inspiration determination procedures are similar as the routines presented in section A.9. The following are the most relevant subroutines implemented for the analysis of the signals.

The signals are stored in the following array:

Flow : buffch[0]
Esophageal Pressure : buffch[1]
Mouth Pressure : buffch[2]
Tidal volume : buffch[3]

Determining calibration factors from calibration files:

```
*****************************************************************
    int   i,j;
    float count,pe,pm,fl;
    if ((dat_file_val = fopen(filename_val,"rb")) == NULL)
    {
    fprintf(stderr," couldn't open file %s for calibration, check if exists\n",
    filename_val);
            exit(1);
    }
    if ((dat_file_zer = fopen(filename_zer,"rb")) == NULL)
    {
```

```
  fprintf(stderr," couldn't open file %s for calibration, check if exists\n",
filename_zer);
          exit(1);
  }
  printf("\nCalibrating signals..................");
  pe=0;pm=0;fl=0;count=0;
while(feof(dat_file_val)==0)
{
numread = fread(buff_plot,sizeof(int),40,dat_file_val);
for (i=0;i<40;i++)
          {
           cal_flow+=buff_plot[i];
           i++;
           cal_eso+=buff_plot[i];
           i++;
           cal_pmm+=buff_plot[i];
           i++;
           }
  count += 10;
  }
  cal_flo /=count; /* calculation of max. cal. value*/
  cal_eso /=count;
  cal_pmm /=count;
  pe=0;pm=0;fl=0;count=0;
while(feof(dat_file_zer)==0)
{
numread = fread(buff_plot,sizeof(int),40,dat_file_zer);
for (i=0;i<40;i++)
          {
           fl+=buff_plot[i];
           i++;
           pe+=buff_plot[i];
           i++;
           pm+=buff_plot[i];
           i++;
           }
  count += 10;
  }
  fl /=count; /* zero values*/
  pe /=count;
  pm /=count;
  cal_eso -= pe; /* max value - zero value*/
  cal_pmm -= pm;
  cal_flow -= fl;
```

```
count = 1.0/cal_eso;
cal_eso = count*10;
count = 1.0/cal_pmm;
cal_pmm = count*10;        /* 10 cmH2O*/
count = 1.0/cal_flow;
cal_flow = count*3;     /* 3 l/min */
fclose(dat_file_val);
fclose(dat_file_zer);
```
****

Detect points of maximum tidal volume, and points of half tidal volume during inspiration and expiration:
****

```
/* detect max volume*/
  for (j=0;j<number_breaths;j++)
  {
        max_vol[j]=buffch[1][init_insp[j]];
        for (i = init_insp[j]+1; i < init_insp[j+1]; i++)
        {
                if (buffch[3][i] > max_vol[j])
                {
                max_vol[j] = buffch[3][i];
                max_vol_points[j] = i;
                }
        }
/*detect  half volumen during insp. and exp.*/

  for (j=0;j<number_breaths;j++)
  {
        half_vol_insp[j]=max_vol[j]/2;
        for (i = init_insp[j]; i < max_vol_points[j]; i++)
        {
                if (buffch[3][i] == half_vol_insp[j])
                        half_vol_insp_points[j] = i;
        }

        half_vol_exp[j]=max_vol[j]/2;
        for (i = max_vol_points[j]; i < init_insp[j+1]; i++)
        {
                if (buffch[3][i] == half_vol_exp[j])
                        half_vol_exp_points[j] = i;
        }
```

```
        }
*******************************************************************
```

Extract offsets in PE and PM:

```
*******************************************************************
        for (i = 0; i < init_insp[0]; i++)
        {
        buffch[1][i] = buffch[1][i] - buffch[1][init_insp[0]];
        buffch[2][i] = buffch[2][i] - buffch[2][init_insp[0]];
        }
        for (i = init_insp[number_breaths]+1; i < 250; i++)
        {
        buffch[1][i] = buffch[1][i] - buffch[1][init_insp[number_breaths]];
        buffch[2][i] = buffch[2][i] - buffch[2][init_insp[number_breaths]];
        }
*******************************************************************
```

Calculation of compliance and tidal volume:

```
*******************************************************************

        for (j=0;j<number_breaths;j++)
        {
              vt[j] = buffch[3][init_exp[j]];
              vt[j] *= (1000.0*cal_flow/60.0);
               compliance[j] = buffch[3][init_exp[j]];
               compliance[j] *= (cal_flow *1000.0/60.0);
               compliance[j] /= - buffch[1][init_exp[j]];
               compliance[j] /= (cal_eso);
        }
                      vst_color (screen,1);
                      v_gtext (screen,20000,12000,"COMPLIANCE:");
                      value = compliance[0];
                      gcvt(value,10,string);
                      v_gtext (screen,20000,11000,string);
```

```
                v_gtext (screen,25000,11000,"ml/cmH2O");
                vst_color (screen,1);
                v_gtext (screen,20000,10000,"TIDAL VOLUME:");
                value = vt[0];
                gcvt(value,10,string);
                v_gtext (screen,20000,9000,string);
                v_gtext (screen,25000,9000,"ml");
```

****

Calculation of total resistance:

****

```
        for (j=0;j<number_breaths;j++)
        {
            tot_r[j] = (buffch[1][half_vol_insp_points[j]]
                        -buffch[1][half_vol_exp_points[j]]);
            tot_r[j] *= cal_eso;
            dummy = (buffch[2][half_vol_insp_points[j]]
                        -buffch[2][half_vol_exp_points[j]]);
            dummy *= cal_pmm;
            tot_r[j] -= dummy;
            tot_r[j] *= -60.0;
            dummy = (buffch[0][half_vol_insp_points[j]]
                    - buffch[0][half_vol_exp_points[j]]);
            dummy *=cal_flow;
            tot_r[j] /= dummy;
        }
                vst_color (screen,1);
                v_gtext (screen,20000,18000,"TOTAL RESISTANCE:");
                value = tot_r[0];
                gcvt(value,10,string);
                v_gtext (screen,20000,17000,string);
                v_gtext (screen,25000,17000,"cmH20/l/s");
```

****

Calculation of inspiratory resistance:

****

```
        for (j=0;j<number_breaths;j++)
        {
            insp_r[j] = ((buffch[1][half_vol_insp_points[j]]
                            - (buffch[1][init_exp[j]]/2))*cal_eso);
            insp_r[j] -= (buffch[2][half_vol_insp_points[j]]*cal_pmm);
```

```
            insp_r[j] *= -60.0;
            insp_r[j] /= ((buffch[0][half_vol_insp_points[j]])*cal_flow);
        }
                    vst_color (screen,1);
                    v_gtext (screen,20000,16000,"INSP RESISTANCE:");
                    value = insp_r[0];
                    gcvt(value,10,string);
                    v_gtext (screen,20000,15000,string);
                    v_gtext (screen,25000,15000,"cmH20/l/s");
****************************************************************
```

Calculation of expiratory resistance:

```
****************************************************************
        for (j=0;j<number_breaths;j++)
        {
            exp_r[j] = ((buffch[1][half_vol_exp_points[j]]
                          - (buffch[1][init_exp[j]]/2))*cal_eso);
            exp_r[j] -= (buffch[2][half_vol_exp_points[j]]*cal_pmm);
            exp_r[j] *= -60.0;
            exp_r[j] /= (buffch[0][half_vol_exp_points[j]]*cal_flow);
        }
                    vst_color (screen,1);
                    v_gtext (screen,20000,14000,"EXP RESISTANCE:");
                    value = exp_r[0];
                    gcvt(value,10,string);
                    v_gtext (screen,20000,13000,string);
                    v_gtext (screen,25000,13000,"cmH20/l/s");
****************************************************************
```

A.11 Signal Analysis: PLOTDIS.EXE

Program implemented for the analysis of chest wall distortion. The setup, graphics displaying, and onset of inspiration determination procedures are similar as the routines presented in section A.9 (program PLOTMTA.EXE). The following are the most relevant subroutines implemented for the analysis of the signals.

The signals are stored in the following array:

Flow : buffch[0]

Ribcage : buffch[1]
Abdomen : buffch[2]
Sum of RC and ABD : buffch[3]

Offset extraction from the RC and ABD signals using the onset of inspiration point as zero reference:

```
****************************************************************
        for (j=0;j<number_breaths;j++)
        {
                for (i = init_insp[j]+1; i < init_insp[j+1]; i++)
                {
                buffch[1][i] = buffch[1][i] - buffch[1][init_insp[j]];
                buffch[2][i] = buffch[2][i] - buffch[2][init_insp[j]];
                buffch[3][i] = buffch[2][i] + buffch[1][i];
                }
                buffch[1][init_insp[j]]= 0;
                buffch[2][init_insp[j]]= 0;
                buffch[3][init_insp[j]]= 0;
        }
****************************************************************
```

Detect zero, maximum and minimum excursion of the ribcage signal:

```
****************************************************************
        for (j=0;j<number_breaths;j++)
        {
                max_rc[j]=buffch[1][init_insp[j]];
                min_rc[j]=buffch[1][init_insp[j]];
                for (i = init_insp[j]+1; i < init_insp[j+1]; i++)
                {
                        if (buffch[1][i] >= max_rc[j])
                        {
                        max_rc[j] = buffch[1][i];
                        max_rc_points[j] = i;

                        }
                }
                for (i = init_insp[j]+1; i < max_rc_points[j]; i++)
                {
                        if (buffch[1][i] <= min_rc[j])
                        {
                        min_rc[j] = buffch[1][i];
```

```
                min_rc_points[j] = i;
                }
        }
        for (i = min_rc_points[j]; i < max_rc_points[j]; i++)
        {
                if (buffch[1][i] >= 0)
                {
                zero_rc_points[j] = i;
                i = max_rc_points[j];
                }
        }
    }
```
****

Calculation of tcd/vt ratio:
****

```
    for (j=0;j<number_breaths;j++)
    {
        for (i = init_insp[j],tcd=0,vt=0; i < init_exp[j]; i++)
        {
        tcd+=(abs(buffch[1][i])+abs(buffch[2][i]));
        vt+=buffch[3][i];
        }
        tcd_f=tcd;
        vt_f = vt;
        tcdvt[j] = tcd_f/vt_f;
                vst_color (screen,1);
                value = tcdvt[j];
                gcvt(value,10,string);
                v_gtext (screen,20000,15000+(j*1000),string);
                value = j+1;
                gcvt(value,10,string);
                v_gtext (screen,26000,15000+(j*1000),string);
    }
```
****

Calculation of phase shift angle between RC and ABD signals:
****

```
    for (j=0;j<number_breaths;j++)
2   {
        angle[j] = ((zero_rc_points[j] - init_insp[j])*360);
```

```
            angle[j] /= (init_insp[j+1] - init_insp[j]);
            value = angle[j];
                    vst_color (screen,1);
                    gcvt(value,10,string);
                    v_gtext (screen,20000,10000+(j*1000),string);
                    value = j+1;
                    gcvt(value,10,string);
                    v_gtext (screen,26000,10000+(j*1000),string);
      }
****************************************************************
```

A.12 Signal Analysis: PLOTPE.EXE

This program evaluates two pressure signals for waveform shape and magnitude proportionality. The setup, graphics displaying, and onset of inspiration determination procedures are similar as the routines presented in section A.9 (program PLOTMTA.EXE). The following are the most relevant subroutines implemented for the analysis of the signals. It plots signals in time (See section A.9) and also plot the two pressures in XY axis.

The signals are stored in the following array:

Flow :	buffch[0]
Pressure #1 :	buffch[1]
Pressure #2 :	buffch[2]

Extract pressure offsets with zero flow reference and outputs
XY-plot of two pressures with timing markers:

```
****************************************************************
      /* extract offsets in PE1 and Pe2 */

      for (j=0;j<number_breaths;j++)
      {
            for (i = init_insp[j]+1; i < init_insp[j+1]; i++)
            {
            buffch[1][i] = buffch[1][i] - buffch[1][init_insp[j]];
```

```
                buffch[2][i] = buffch[2][i] - buffch[2][init_insp[j]];
                }
        }
        /* plot xy on screen... */
        vsl_color (screen,8);
        vsl_type (screen,1);
        /*   X Y AXIS*/
        xy[0] = 0;
        xy[1] = 5000;
        xy[2] = 10000;
        xy[3] = 5000;
        v_pline (screen,2,xy);
        xy[0] = 5000;
        xy[1] = 0;
        xy[2] = 5000;
        xy[3] = 10000;
        v_pline (screen,2,xy);
        /* plot xy pe1 pe2*/
        for (i = 0,j=0; i < (init_insp[1]-init_insp[0]);j++,i++)
        {
                xy[j]=(2*gain[1] * buffch[1][init_insp[0]+i])+5000;     /* x */
                j++;
                xy[j]=(2*gain[2] * buffch[2][init_insp[0]+i])+5000;     /*  y */
        }
                        vsl_color (screen,5);
                        vsl_type (screen,1);
                        v_pline (screen,(init_insp[1]-init_insp[0]),xy);
*****************************************************************
```

Calc max, min pressure values:

```
*****************************************************************
                max_pe1=buffch[1][init_insp[0]];
                min_pe1=buffch[1][init_insp[0]];
                max_pe2=buffch[2][init_insp[0]];
                min_pe2=buffch[2][init_insp[0]];
                for (i = init_insp[0]+1; i < init_insp[1]; i++)
                {
                if (max_pe1 < buffch[1][i])
                        {
                        max_pe1 = buffch[1][i];
                        max_pe1_point = i;
                        }
```

```
            if (min_pe1 > buffch[1][i])
                {
                min_pe1 = buffch[1][i];
                min_pe1_point = i;
                }
            if (max_pe2 < buffch[2][i])
                {
                max_pe2 = buffch[2][i];
                max_pe2_point = i;
                }
            if (min_pe2 > buffch[2][i])
                {
                min_pe2 = buffch[2][i];
                min_pe2_point = i;
                }
            }
```

****

Calculation of linear regression slope and cross corr. coeff. :

****

```
    /* calling sequence*/
            slope_regression(init_insp[1]-init_insp[0]);
            v_gtext (screen,11000,1000,"R.Slope=");
            value = slope_pressure;
            gcvt(value,4,string);
            v_gtext (screen,15000,1000,string);
            v_gtext (screen,11000,2000,"CORR=");
            value = corr_coeff;
            gcvt(value,4,string);
            v_gtext (screen,15000,2000,string);

    /* calculates slope regression and cross correlation coeff*/
slope_regression(points)
    int points;
    {
    int i,h,k;
    float root1,root2,square,zx,zy,zxy,zxx,zyy;
    float point_f;

            if ((points>0)&& (points<250))
            {
            zxy = 0.0;
```

```
        for (i = 0; i < points; i++)
                zxy +=  buff_signal_pe1[i] *buff_signal_pe2[i] ;
        zx = 0.0;
        for (i = 0; i < points; i++)
                zx +=  buff_signal_pe1[i];
        zy = 0.0;
        for (i = 0; i < points; i++)
                zy +=  buff_signal_pe2[i];
        zyy = 0.0;
        for (i = 0; i < points; i++)
                zyy +=  (buff_signal_pe2[i] * buff_signal_pe2[i]);
        zxx = 0.0;
        for (i = 0; i < points; i++)
                zxx +=  (buff_signal_pe1[i] * buff_signal_pe1[i]);
        point_f = points;
        square = (zxx - (zx * zx / point_f));
        slope_pressure = (zxy - (zy * zx /point_f))/(square);
    /* correlation*/
        square =  (zxx - (zx * zx / point_f));
        if (square < 0.0)
                {
                printf("error xx");
                exit(1);
                }
        root1 = sqrt(square);
        square = (zyy - (zy * zy / point_f));
        if (square < 0.0)
                {
                printf("error yy");
                exit(1);
                }
        root2 = sqrt(square);
        if ((root1*root2)>0.0)
        corr_coeff = (zxy - (zx * zy /point_f))/(root1 * root2);
         }
}
*****************************************************************
```

A.13 Signal Analysis: SPECTRUM.EXE

This program obtains the PSD periodogram of the EMG signals. The following is the routine that performs the 1024 points FFT.

Subroutine for FFT:

```
*****************************************************************
        static struct complex {
                        float real;
                        float imag;
                        };
        static float buffer_plot_float[512];     /*temporary*/
        static int buffer_plot_fft[512];
        int n, i, j,h, norm, bits,significant_digits;
        long m,mm;
        double point;
        static struct complex buff_fft[1024];
        static int buff_int[1024];
        static double buff_write[512];
/* Define a pointer (r) to the complex structure (c) to handle real numbers.
  Note that the compiler will issue a warning message which can be
  safely ignored. */
        float *source_fft;
        int *dest_fft;
        float *r, scale,kreal;
        r = &buff_fft;
        dest_fft =&buffer_plot_fft;
        source_fft =&buffer_plot_float;
        /* size of ffts      */
        bits = 16;
        n = 10;
        m = 1024;

        /* calc k number of intervals 1024 points */
        k = filesize / 2048;
        kreal=k ;
        /* perform k ffts  and send to file*/
        printf("\n\n\n\n                    Wait ..");
        for ( j = 0; j < k; j++)
        {
                /* read sequence and convert int to fp*/
                numread= fread(buff_int,sizeof(int),1024,dat_file_hdl);
```

```
        bits = 16;
        for ( i = 0,h=0; i < 1024; i++,h++)
        {
        buff_fft[h].real = buff_int[i];
        i++;
        buff_fft[h].imag = buff_int[i];
        }
    /* Perform a 1024 points real FFT and convert the complex results
    to polar (magnitude/phase) form. Note that integer m must be
    a long integer. */
        i=1;
        norm=0;
        scale=1.0;
    /* hamming window*/
        hamm(r,r,&n);
    /* fft 1024 pts*/
        rfft(r,&n,&norm,&scale);
    /*convert to polar form and send to destination file*/
        mm=512;
    polar(r,r,&mm,&i);
        for ( h = 0; h < 512; h++)
        buff_write[h] += ((buff_fft[h].real * buff_fft[h].real)/1024.00);
}
****************************************************************
```

A.14 Signal Analysis: PLOTFFT.EXE

PLOTFFT.EXE displays the 512 points average periodogram stored in binary form by the program SPECTRUM.EXE (A.13). It uses the same graphics procedure set up as explained in section A.9.

Plotting two axis and periodogram:

```
****************************************************************
    /* axis*/
    xy[0] = 0;
    xy[1] = 2000;
    xy[2] = 0;
    xy[3] = 22000;
    vsl_color (screen,15);
    vsl_type (screen,15);
    v_pline (screen,2,xy);
    xy[0] = 0;
    xy[1] = 2000;
```

```
xy[2] = 20480;
xy[3] = 2000;
v_pline (screen,2,xy);
/* read file*/
numread = fread(buff_plot,sizeof(int),512,dat_file_hdl);
/* plot psd*/
for (i = 0; i<512;i++)
{
            xy[0] = 40*i;
            xy[1] = 2000;
            xy[2] = 40*i;
            xy[3] = (gain * (buff_plot[i]))+2000;

            vsl_color (screen,5);
            vsl_type (screen,15);
            stop = v_pline (screen,2,xy);
            if (stop == -1)
            {
            printf ("Error %d in  plotting  Open Ws",vq_error());
            exit (-1);
            }
 }
```
****