INVESTIGATION OF THE EVOKED MAGNETIC ACTION FLUX OF SKELETAL MUSCLE

by

© DONALD B. MacHATTIE

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Doctor of Philosophy

McMaster University

September 1987

EVOKED MAGNETIC ACTION FLUX OF SKELETAL MUSCLE

DOCTOR OF PHILOSOPHY (1987)
(Electrical and Computer Engineering)

McMASTER UNIVERSITY
Hamilton, Ontario

TITLE: Investigation of the Evoked Magnetic Action Flux of Skeletal Muscle

AUTHOR: Donald B. MacHattie,

B.Sc. (Honours Physics, Queen's University, Kingston)

M.Sc. (Physics, University of Toronto)

SUPERVISOR: Dr. L. D. Pengelly

number of pages: xv, 268

To Barbara,

For her constant and loving knitting

of my ravelled sleeves.

ABSTRACT

The magnetomyogram (MMG) is a magnetic signal that may be measured external to an active skeletal muscle. It is generated by the transmembrane ion currents associated with the propagation of action potentials along the plasma membranes of the muscle fibres. Compared with other biomagnetic signals, such as the magnetocardiogram and the magnetoencephalogram, the MMG has been studied very little.

Presented in this dissertation is the development of a strategy for investigating the potential of the MMG as a research and clinical tool. Investigation of the compound magnetic action flux (MAF), the basic component of the MMG signal, was chosen in favour of direct study of the MMG. The design of a data acquisition system and experimental procedure for investigation of the MAF is also reported. A preliminary set of measurements from an excised frog gastrocnemius under controlled physiological conditions is presented. These measurements were designed to reveal the temporal and spatial characteristics of the MAF and to demonstrate the effect of several physiological variables on these characteristics. Such measurements have never before been reported.

The experimental measurements showed that the MAF from a frog gastrocnemius is a biphasic signal with an amplitude of 30 pT and a duration of 4.2 ms. The MAF was found to occur at the same time as the whole muscle action potential and well before the mechanical response of the muscle (generation of twitch force). The MAF was found to have an

azimuthal component and a component that looped from end-to-end of the muscle. For submaximal stimulation, the MAF amplitude was found to increase nonlinearly with the twitch force amplitude. For supramaximal stimulation, the MAF amplitude was found to decrease with increasing muscle length. Finally, it was found that the electrical conductivity of the bath medium strongly affected the amplitude of the MAF signal.

This research has demonstrated the feasibility of performing MAF measurements in a typically "noisy" laboratory with a commercially available SQUID magnetometer. It has also shown that the MAF signal is reproducible and sensitive to some physiological variables. With these results, a greater knowledge and better understanding of the MAF has developed. Further evaluation of the potential of the MMG as a research and diagnostic tool is justified.

ACKNOWLEDGEMENTS

This research was supported by a grant from the Natural Sciences and Engineering Research Council of Canada (grant #A4513). Access to the SQUID magnetometer was made possible by Dr. D. W. Strangway and Dr. N. Sugiara of the University of Toronto. Their generosity is much appreciated. I would also like to thank Dr. D. G. McDonald for tutoring in dissection and muscle preparation techniques and for supplying chemicals for the MacKenzie's saline solution.

I am grateful to the members of my supervisory committee for their guidance and encouragement. Dr. H. deBruin and Dr. A. J. McComas offered extensive bodies of knowledge and experience from which to draw.

I am most grateful to my supervisor, Dr. L. D. Pengelly, for his insatiable curiosity and his openness of mind. For without these qualities, this project would never have been tackled. His enthusiasm and support have carried me through some rough times and his criticisms have been kind. Dr. Pengelly's ability to maintain "the overall picture" and communicate the essence of a project have hopefully rubbed off on me.

My friend and colleague, Gordon Jasechko, provided many stimulating discussions and helped me debug some of my circuitry and software. The fresh perspective he introduced was often very productive.

I am very thankful for the love and care of my family. This support was very important. I would like also to thank my father for

his wiring of a circuit board, suggestion of the copper ring for noise reduction and proof reading.

Finally, it is my wife, Barbara, to whom I am most indebted. This research has been a major undertaking for both of us. Barbara has shared in the victories and the defeats, provided many useful suggestions, and even gone frog-catching when we ran out of specimens. But, most of all, she has kept loving and caring for me even when I wasn't very lovable.

TABLE OF CONTENTS

PAGE

ABSTRACT.....iv

ACKNOWLEDGEMENTS.....vi

LIST OF FIGURES.....xii

LIST OF TABLES.....xv

CHAPTER 1 : INTRODUCTION.....1

1.1 THE MAGNETOMYOGRAM.....1

1.2 THE MMG LITERATURE.....3

1.3 MOTIVATION.....5

1.4 RESEARCH OBJECTIVES AND DOCUMENT STRUCTURE 9

CHAPTER 2 : BIOMAGNETIC SOURCES.....13

2.1 EXCITABLE MEMBRANES.....13

2.1.1 Cell Membrane Structure.....13
2.1.2 The Resting Membrane.....14
2.1.3 The Action Potential.....17
2.1.4 Membrane Ion Currents.....18

2.2 MAXWELL'S EQUATIONS.....19

2.2.1 The Origins of Maxwell's Equations 19
2.2.2 Source Terms in Maxwell's Equations.....25

2.3 SOURCE MODELS.....28

2.3.1 Homogeneous Media.....28
2.3.2 Inhomogeneous Media.....30
2.3.3 Secondary Sources.....31
2.3.4 The Cylindrical Cell.....32
2.3.5 Muscle as a Magnetic Source.....35

CHAPTER 3 : DATA ACQUISITION HARDWARE.....39

3.1 MAGNETOMETER.....39

3.1.1 Choice Of A Magnetometer.....39
3.1.2 CTF Systems SQUID Magnetometer.....44
3.1.3 Magnetic Noise Management.....47

3.2 ANALOGUE HARDWARE.....49

3.2.1 Force Transducer Circuitry.....49
3.2.2 Action Potential Circuitry.....51
3.2.3 Calibration Circuits.....54

3.3 THE COMPUTER 54

3.3.1 Computer Requirements.....54
3.3.2 The Hewlett-Packard
Integral Personal Computer (9807).....56
3.3.3 Limitations.....57

3.4 INTERFACING INSTRUMENT.....59

3.4.1 Purpose and General Description.....59
3.4.2 Computer Interface.....62
3.4.3 Magnetometer Interface.....67
3.4.4 Analogue-to-Digital Conversion.....72
3.4.5 Data Acquisition Control and Timing.....77
3.4.6 Sample Counter.....80
3.4.7 Ground Loops.....82

CHAPTER 4 : SOFTWARE DESIGN.....84

4.1 EXPERIMENT CONTROL.....85

4.1.1 The Problem.....85
4.1.2 Calibration Procedures.....88
4.1.3 Anatomical Data Entry
and Mounting of the Muscle.....90
4.1.4 Three Experiments.....93

4.2 DATA ACQUISITION SOFTWARE.....94

4.2.1 Requirements.....94
4.2.2 Implementation.....96
4.2.3 The Data_check Programme.....100

4.3 DATA RETRIEVAL AND ANALYSIS SOFTWARE.....104

4.3.1 Data Retrieval.....104
4.3.2 Plotting Software.....105
4.3.3 Muscle Fatigue Corrections.....107
4.3.4 Power Spectra.....108
4.3.5 Experiment Analysis 113

CHAPTER 5 : IN VITRO MEASUREMENTS OF THE EVOKED MAGNETIC ACTION FLUX.....114

5.1 EXPERIMENTAL DESIGN.....114

5.1.1 Objectives.....114
5.1.2 The Specimens.....115

5.2 MUSCLE ENVIRONMENT.....117

5.2.1 Muscle Bath.....117
5.2.2 Muscle Manipulator.....119
5.2.3 Force Transducer.....124
5.2.4 Action Potential Electrodes.....126
5.2.5 Muscle Stimulation.....129

5.3 EXPERIMENTAL PROCEDURE.....130

5.3.1 Set Up.....130
5.3.2 Calibration.....131
5.3.3 Dissection.....132
5.3.4 Mounting the Muscle.....134
5.3.5 Position Experiment.....135
5.3.6 Length Experiment.....140
5.3.7 Stimulus Experiment.....140
5.3.8 Muscle Bath Medium Experiment.....141

CHAPTER 6 : RESULTS: MAGNETIC ACTION FLUX OF THE FROG GASTROCNEMIUS.....143

6.1 RESPONSE TO A SINGLE PULSE STIMULUS.....143

6.2 EFFECT OF THE MUSCLE BATH MEDIUM.....154

6.3 SPATIAL ASPECTS OF THE MAF.....158

6.4 MUSCLE LENGTH EXPERIMENT.....162

6.5 STIMULUS AMPLITUDE EXPERIMENT.....163

CHAPTER 7 : DISCUSSION.....166

7.1 THE MAF UNDER OPTIMAL CONDITIONS.....166

7.2 EFFECT OF THE MUSCLE BATH MEDIUM.....170

7.3 SPATIAL ASPECTS OF THE MAF.....171

7.4 MUSCLE LENGTH EXPERIMENT.....174

7.5 STIMULUS AMPLITUDE EXPERIMENT.....176

7.6 RESEARCH OBJECTIVES.....179

CHAPTER 8 : CONCLUSION.....181

APPENDIX A: DATA ACQUISITION SOURCE CODE.....185

APPENDIX B: DATA CHECK SOURCE CODE.....213

APPENDIX C: DATA ANALYSIS SOURCE CODE.....234

REFERENCES.....265

LIST OF FIGURES

FIGURE		PAGE
1.1	Development of Research Strategy	12
2.1	Current Dipole Model of an Action Potential	33
3.1	Data Acquisition System	40
3.2	Detector Coils of Magnetometer	45
3.3	Typical Magnetic Noise Spectrum	50
3.4	Force Transducer Circuitry	52
3.5	Whole Muscle Action Potential Circuitry	53
3.6	Action Potential Calibration Circuit	55
3.7	Interfacing Instrument	61
3.8	Talker Address Identification Circuit	63
3.9	Listener Handshake Circuit	64
3.10	Talker Handshake Circuit	65
3.11	Computer Transceiver Circuit	66
3.12	Digital Multiplexer	68
3.13	Magnetometer Handshake Circuit	69
3.14	Magnetometer Transceiver Circuit	70
3.15	Magnetometer Simulator Circuit	71
3.16	Sample-and-Hold Circuits	73
3.17	Analogue-to-Digital Converter (ADC)	75
3.18	DC Logic Support Circuits	76
3.19	Timing of Data Acquisition Control Signals	78
3.20	Sample Counter Circuit	81

4.1 Block Diagram of the SMDA Programme.....87

4.2 The "set_up" Subroutine.....89

4.3 The "muscle_dat" Subroutine.....91

4.4 The "DATA_CHECK" Programme.....101

4.5 Muscle Fatigue Correction Calculations.....109

4.6 MAF Power Spectrum.....111

4.7 Windowed MAF Power Spectrum.....112

5.1 The Muscle Bath.....118

5.2 Spatial Coordinate System.....119

5.3 Platform 1 of Muscle Positioning Device.....121

5.4 Platform 2 of Muscle Positioning Device.....123

5.5 Platform 3 of Muscle Positioning Device.....125

5.6 Force Transduction System.....127

5.7 Whole Muscle Action Potential Electrodes.....128

5.8 Muscle Position For Maximum MAF Amplitude.....137

5.9 Three Spatial Mappings of the MAF.....139

6.1 Two MAF Recordings.....144

6.2 Average of 9 Successive MAF's.....145

6.3 Power Spectrum of the Average of 9 MAF's.....146

6.4 A Second Power Spectrum of Averaged MAF's.....148

6.5 Windowed MAF and Its Power Spectrum.....149

6.6 A Second "Windowed" Power Spectrum.....150

6.7 Timing of MAF and Muscle Twitch.....151

6.8 Whole Muscle Action Potential.....152

6.9 Two Averaged Bullfrog MAF's.....155

6.10 MAF's From Muscle in MacKenzie's Saline.....157

6.11 MAF's From Muscle in Mineral Oil.....158

6.12 MAF's From Muscle in MacKenzie's Saline.....159

6.13 MAF Amplitude vs X-Position.....160

6.14 MAF and Artifact Amplitudes vs Z-Position.....161

6.15 MAF Amplitude vs Y-Position.....162

6.16 MAF Amplitude vs Muscle Length.....163

6.17 MAF Amplitude vs Force Amplitude.....164

6.18 MAF DC-Offset vs Force Amplitude.....165

6.19 Peak Force Delay vs Force Amplitude.....165

7.1 Azimuthal Component of the MAF.....172

7.2 End-to-End Component of the MAF.....174

7.3 Exponential Function of MAF Amplitude vs Force.....177

LIST OF TABLES

TABLE		PAGE
6.1	MAF, Stimulus Artifact, WMAP and Force Statistics	153
6.2	Frog-to-Frog Signal Statistics	156

CHAPTER 1

INTRODUCTION

1.1 THE MAGNETOMYOGRAM

The electromyogram (EMG) is a recording of the electric potential difference between two electrodes applied to the skin overlying active skeletal muscle (or between electrodes embedded in the muscle). It has been used extensively as a fundamental tool in physiological research and for clinical diagnosis of neuromuscular diseases. The electric potential of the skin overlying an active muscle is caused to fluctuate by the propagation of action potentials along the muscle fibres. The flow of ions associated with the propagation of an action potential constitutes an electrical current. As with any electrical current, these ion currents will generate magnetic fields. It is the fluctuating magnetic field due to active skeletal muscle that is the object of study in this research.

The magnetomyogram (MMG) has been defined as "a recording of one component of the magnetic field vector vs time, where the magnetic field at the point of measurement is due to currents generated by skeletal muscle" (Cohen & Givler, 1972). Because the MMG and the EMG both arise from the propagation of action potentials, they may present similar characteristics (Plonsey, 1981; Tripp, 1981; Gielen & Wikswo, 1985). MMG recordings from near the human elbow "have the same general form as

EMG's recorded on the skin over contracted skeletal muscles, where the voltage signals are known to be due to the action potentials of the muscle motor units" (Cohen & Givler, 1972). Measurements from the human thenar muscles yielded MMG peak-to-peak amplitudes of approximately 20 picotesla (pT) and a power spectrum that peaked at 80 Hz and showed little power over 300 Hz. Measurements from the region of the human elbow yielded larger amplitude and lower frequency magnetic signals (Cohen & Givler, 1972).

In general, Duret & Darp (1983) reported that the MMG has an amplitude of 10 pT and a power spectrum that ranges from DC to 2 kHz. This, and the paper of Cohen & Givler define the basic characteristics of the magnetic signal measured from human skeletal muscles under voluntary control. The electrophysiological processes involved in the generation of the MMG are reviewed in Chapter 2 of this dissertation.

It should be noted that although the MMG power spectrum drops off at low frequencies, a DC field has been observed and studied (Cohen & Givler, 1972; Cohen, et al., 1980). This DC field was observed after a muscle had been actively contracting. It was found to persist for several minutes while the muscle rested. Since the DC magnetic field did not appear to be directly associated with activation of the muscle, it was not considered to be part of the MMG and was not considered in this research.

1.2 THE MMG LITERATURE

The first biomagnetic signal to be measured was from the heart (Baule & McFee, 1963). The magnetometer consisted of two large induction coils, each of 2 million turns. A voltage of only 30 microvolts was induced in these coils. No other human biomagnetic signals were detected until a specially shielded room was employed (Cohen, 1968). With the environmental magnetic noise significantly reduced, the alpha rhythm of the magnetoencephalogram was detected (with the aid of signal averaging).

It was not until the SQUID (Superconducting Quantum Interference Device) magnetometer was used by Cohen, Edelsack and Zimmerman (1970) that interest in biomagnetism really took off. Based on the quantum mechanics of the Josephson junction, this detector provided several orders of magnitude greater sensitivity than the large coils of the earlier instrumentation. Magnetic fields from skeletal muscle, the lungs and the eye were soon discovered, and work on the magnetocardiogram and magnetoencephalogram progressed rapidly.

Relatively little research has been done on the magnetic field generated by skeletal muscle. The magnetic fields generated by the heart and brain have attracted most researchers entering the field of biomagnetism. Magnetic measurements from the brain have been of great interest largely due to the failure of the EEG to "see" into the brain and localize centers of activity. The skull has proven to be much more transparent to magnetic fields than to electric fields. Because the heart is a life-critical organ and produces the largest amplitude

magnetic signal of the human body, it too has been well researched. Other areas that have developed significantly involve DC fields from the abdomen and limbs, and magnetic fields from the eye. The great success of the EMG and the relative difficulty of measuring weak magnetic fields are probably the main reasons for the lack of interest in the MMG.

The MMG literature is very small. Since the MMG was first reported by Cohen & Givler in 1972, only 4 papers have presented MMG recordings. As described in the previous section, Cohen & Givler recorded magnetic signals from close to the human elbow (during voluntary flexion of the elbow) and from the palm of the hand. The next paper to present an MMG recording, discussed it only as a potential source of interference in magnetoencephalography recordings (Reite et al., 1976). Study of MMG signals from voluntary contractions of the human tibialis anterior was reported at the 1982 Biomagnetism conference in Rome (Koga & Nakamura, 1983). Level-triggered signal averaging techniques were used to extract single action potentials from the MMG pulse trains. Single action potential amplitudes and durations were in the neighbourhoods of 50 pT and 10 ms respectively.

Of greatest interest is the recent recording of magnetic signals from single motor units from the extensor digitorum longus in rats (Gielen & Wikswo, 1985; Gielen, Roth & Wikswo, 1986). Signals from the firing of single motor units (of approximately 75 fibres) were obtained with a signal-to-noise ratio of 14 dB. The magnetic recording was found to be less sensitive to magnetometer position than the electric potential recording was to electrode placement. Work to be published by Roth & Wikswo (Gielen & Wikswo, 1985), suggests that the magnetic field

associated with action currents in cylindrical fibers is less sensitive than the surface electric potential to the conductivities around the fibre and may give a more reliable signal in terms of revealing the physiology of active muscle fibers.

1.3 MOTIVATION

There are several reasons why the magnetomyogram (MMG) has not been studied. As stated in the previous section, most researchers entering the field of biomagnetism have been attracted to the heart and the brain. The greatest reason is that the EMG has been a successful research and diagnostic tool for neuromuscular diseases and the MMG is technically more difficult to measure. The Earth's static magnetic field is approximately 70 microtesla (Williamson & Kaufman, 1981). Magnetic oscillations from the earth's ionosphere and urban sources (motors, transformers, automobile ignitions, etc.) are in the order of 500 nanotesla (Cohen, 1975). As stated earlier, the MMG typically has an amplitude of 10 picotesla (pT). Thus, the signal is several orders of magnitude smaller than the environmental noise. The use of SQUID magnetometers with detection coils in gradiometer configurations has allowed measurements to be made without the great expense of a magnetically shielded room. However, even the cost of a SQUID magnetometer is approximately forty thousand Canadian Dollars. The requirement for liquid helium to cool the superconducting components is a further technical impediment. This can add a running cost of approximately four

thousand Canadian dollars per year.

Fortunately, there are some new technologies on the horizon that promise to bring down the cost of making biomagnetic measurements (see section 3.1.1). First, there is the recent development of superconducting ceramic materials that operate at liquid nitrogen temperatures. As well as removing the considerable expense of the liquid helium, the higher temperature greatly simplifies the dewar design. The other technology involves the use of laser light in optical fibers with magnetostrictive coatings. Interference between laser light from the coated fibre and from a reference fibre provides a signal that is very sensitive to magnetic fluctuations (though the frequency response may be limited).

Despite the problems cited above, there are several compelling reasons for investigating the MMG. First, the human body is more "transparent" to magnetic fields than electric fields. Biomagnetic measurements do not involve physical contact with the subject as required for electric potential measurements. The magnetic field is a vector function of space and time while the EMG can usually only provide components of the electric field vector in the plane of the skin surface. Finally, there is the possibility that the MMG may contain information about the physiological processes of muscle tissue that is not present in the EMG.

Measurement of electric field strength depends greatly on the conductivity of the medium between the source and the point of measurement. For magnetic measurements, it is the magnetic permeability of the medium. Biological tissues have been found to have electric permittivities and magnetic permeabilities close to the values of free space.

These values vary little from one tissue to the next (Tripp, 1983; Plonsey, 1981; Williamson & Kaufman, 1981). Contrasting this is the variation of electrical conductivity over two orders of magnitude for human tissues (Williamson & Kaufman, 1981). This means that deep muscle sources might be "seen" with magnetic measurements more clearly than with skin electric potential measurements. Support of this reasoning is given by the report that the centre of a somatically evoked MEG field pattern could be located 3 cm under the scalp with an accuracy of 1 cm (Williamson & Kaufman, 1981).

Magnetic measurements involve placing a detector close to the active tissue. There is no physical contact. No electrodes have to be placed on "prepared" areas of a subject's skin as with EMG measurements. It is not even necessary to remove the subject's clothing. This could be an advantage in a busy clinic. Also, because there are no electrodes, there is no electrical hazard. Not being constrained to the body surface as with EMG electrodes provides the opportunity for performing spatial scans of the magnetic field. This, in combination with the transparency of the body to magnetic fields provides the potential for tomography and imaging techniques.

The vector information of the MMG is more easily obtained than that of the EMG. A set of three orthogonal coils can be easily used to completely determine the magnetic field vector. EMG vector measurements are difficult because the electrodes are restricted to the skin surface. In general, vector measurements provide more information than scalar measurements. Magnetic vector measurements from the heart have already been reported (Barry et al, 1977; Lekkala & Malmivuo, 1981).

Finally, there is the possibility that the MMG arises from slightly different aspects of action potential propagation or internal muscle structure and may therefore contain some information not available in the EMG. There has been much speculation about this, but little has been proven by experiment. The general consensus seems to be that "although in some instances there are remarkable similarities between biomagnetic and skin potential phenomena, the two measures in many cases are also at least partially independent in what they reveal about bodily functions" (Williamson & Kaufman, 1981). It appears that researchers have felt that there is not promise enough of new information in the MMG or more research would have been done by now. As will be discussed in Chapter 2, one model of a cylindrical cell in an infinite homogeneous medium has yielded magnetic and electric fields which are not independent (Tripp, 1983; Plonsey, 1981). However, as stated in the previous section, the work of Gielen & Wikswo indicates that the magnetic field associated with action currents in cylindrical fibres is less sensitive than the surface electric potential to the conductivities around the fibre and may give a more reliable signal in terms of revealing the physiology of active muscle fibres.

Regardless of whether the MMG contains new information (independent of the electric potential), MMG vector measurements will likely provide more information than scalar EMG measurements. This, combined with the possibility of detecting the activity of deep muscles (inaccessible to surface EMG recordings) and the possibility of tomography provide compelling (though unproven) incentives to investigate the clinical potential of the MMG.

1.4 RESEARCH OBJECTIVES AND DOCUMENT STRUCTURE

From the above discussion, it is evident that the MMG is a measurement that has the potential to provide information not available to conventional EMG techniques. It is our hypothesis that the magnetomyogram (MMG) can be developed into a useful research and diagnostic tool for neuromuscular diseases. The overall objective of our research initiative was to investigate the potential of the magnetomyogram as a research and diagnostic tool.

Since very little work had been done on the MMG, there were many possible avenues to follow. It was decided that it was of greatest importance to try to determine the basic characteristics of the MMG and investigate how these characteristics depend on physiological variables. Such an investigation requires as complete control of the muscle as possible. This determined that an excised muscle preparation should be used.

The use of excised muscle tissue presented several important advantages over in situ measurements. The MMG (as for the EMG) is a superposition of asynchronous action potential spikes from the individual motor units of a muscle. Thus, it is the single motor unit action potential (SMU-AP) that is the basic building block of the MMG (and EMG). To develop an understanding of the MMG it was felt that the magnetic signal from the SMU-AP should first be thoroughly investigated.

Indirect stimulation of the muscle was chosen because it is more natural than direct stimulation and results in a smaller stimulus artifact. Single square voltage pulses were to be applied to the nerve

of the muscle. The magnetic flux density resulting from activation of the muscle (under isometric conditions) was to be measured as a function of time. We have called this signal the "magnetic action flux" (MAF). It is the magnetic equivalent of the electric action potential.

Both single or compound MAF signals may be generated by providing a very mild stimulus or a supramaximal stimulus to the nerve. A very low signal-to-noise ratio was expected. To maximize the signal-to-noise ratio, supramaximal stimulations were used to generate compound MAF signals. It was felt that although the superposition of individual MAF's in the compound signal would tend to smooth the finer details, the fundamental characteristics of the individual MAF's would not be lost.

Having determined that the research would involve a set of magnetic measurements from an excised muscle preparation, a specific muscle had to be chosen and a data acquisition system had to be designed. To allow comparison of results with the EMG literature, it was desirable to choose a standard muscle preparation. The frog gastrocnemius was chosen. Design of the data acquisition system involved electronic hardware, computer software and an apparatus to hold and maintain the muscle preparation.

Once a general purpose data acquistion system was in place, a set of specific MAF experiments had to be designed to investigate the effect of individual physiological variables on the MAF. Analysis of these measurements should then provide material for modelling the muscle as a magnetic source. From there, in situ MMG measurements could be pursued.

Figure 1.1 provides a summary of the above discussion. It outlines, in the form of a flowchart, the path of reasoning followed in determining the course of action to be followed in this research. This dissertation presents the development of a data acquisition system, experimental procedure and the results of a preliminary set of MAF measurements.

Chapter 2 of this dissertation deals with the theory of biomagnetic sources. First, the electrochemical processes of excitable membranes are considered. Then, Maxwell's equations are introduced and the electrodynamics of excitable tissues is investigated. Some modelling of specific biomagnetic sources is reviewed and its applicability to skeletal muscle is considered.

Chapters 3, 4 and 5 deal with the design of the experimental apparatus and procedures. In Chapter 3, the data acquisition hardware is discussed. The design of software is presented in Chapter 4. Extensive software listings are presented in the appendices. The apparatus for holding and maintaining the muscle preparation and the experimental procedures are described in Chapter 5.

The results of the experiments described in section 5.3 are presented in Chapter 6. A discussion of these experimental results and their implications is presented in Chapter 7. Chapter 8 provides a brief summary and conclusion of this dissertation.

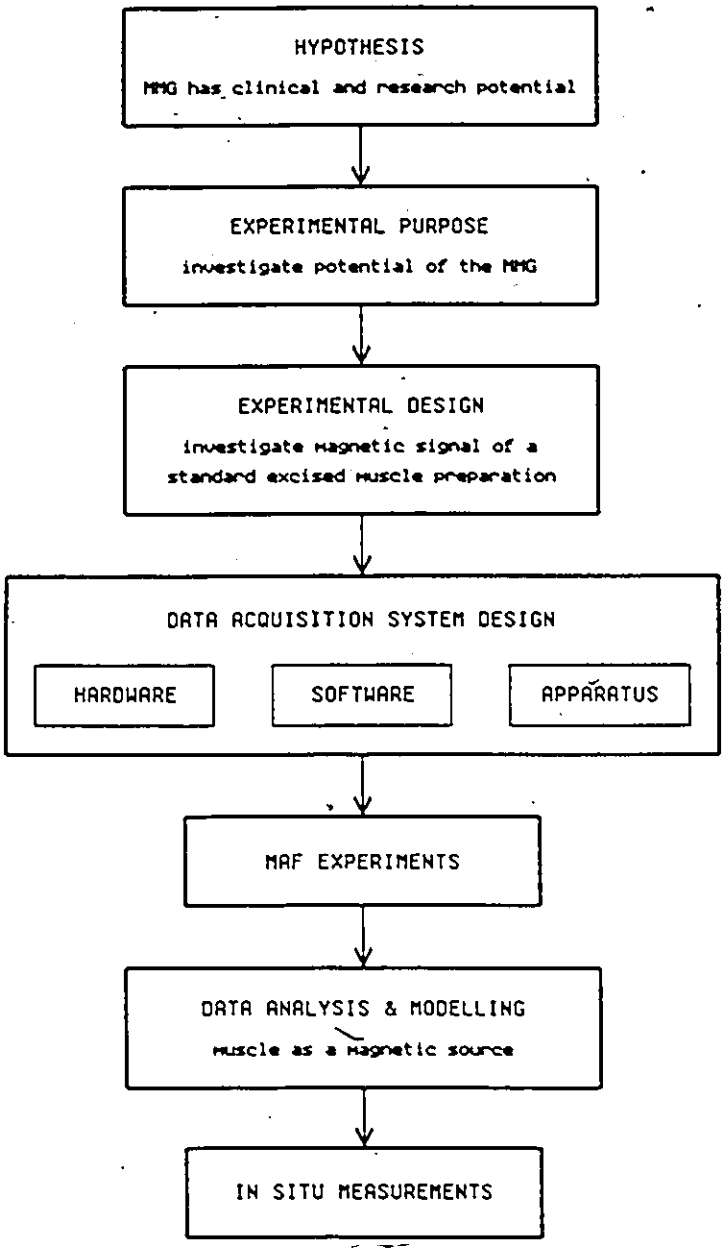


Figure 1.1 : Flowchart summarizing the steps in determining the course of this research.

CHAPTER 2

BIOMAGNETIC SOURCES

The purpose of this chapter is to briefly review the physiology of the excitable membrane and consider some of the basic electromagnetic theory that will lead to an elementary understanding of how magnetic fields may be generated by skeletal muscle. The membrane physiology presented in this chapter has been derived from the physiology texts by Guyton (1981) and Vander, Sherman & Luciano (1980) and the monograph by Carlson & Wilkie (1974). The basic discussion of Maxwell's equations comes from the physics texts of Halliday & Resnick (1962) and Jackson (1975). Discussion of the application of basic electromagnetic theory to biomagnetic sources was derived from the Biomagnetism text edited by Williamson, Romani, Kaufman & Modena (1983) and papers by Plonsey (1981) and Tripp (1981).

2.1 EXCITABLE MEMBRANES

2.1.1 Cell Membrane Structure

The cytoplasm of living cells is enclosed by a limiting membrane, called the plasma membrane. This membrane is primarily composed of a double layer of phospholipid molecules. The polar phosphate heads are hydrophylic and therefore appear on the inner and outer surfaces of

the membrane. The lipid portions of these molecules are hydrophobic and therefore make up the inner layers of the plasma membrane. The inner lipid layers of the membrane are not electrically conducting. The result is that the plasma membrane is electrically insulating.

In general, the plasma membrane acts as a physical and electrical barrier between the intracellular fluid (cytoplasm) and the extracellular fluid. The presence of many large protein molecules, embedded in the phospholipid bilayer, highly modifies the function of the plasma membrane. These proteins can act as selective pores, receptor sites for external messenger molecules and can be involved in active transport processes. Of greatest importance to the electrical activity of the specialized nerve and skeletal muscle cells are the pores that allow only selected ions to pass and the proteins that are involved in actively transporting ions against concentration gradients.

2.1.2 The Resting Membrane

As stated in the previous section, one of the functions of the plasma membrane is to help maintain differences in composition of the intracellular fluid and the interstitial fluid. For example, the ionic concentrations within a skeletal muscle fiber of a frog are typically 139, 20 and 3.8 millimoles per litre for potassium, sodium and chlorine ions respectively. Concentrations in the interstitial fluid are typically 2.5, 120 and 120 mmole/l for potassium, sodium and chloride (Taccardi, 1983). A strong concentration gradient exists across the plasma membrane for each of these three ions.

If a concentration gradient exists across a membrane for an ion for which the membrane is semipermeable, there will be a net diffusion current from the solution of high concentration to the solution of low concentration. Since the ion is a charged particle, a potential difference will develop across the membrane. A net flow of ions will continue until the electric potential difference just balances the concentration gradient. When this state of equilibrium is reached, the relationship between the transmembrane potential and the ionic concentration in the two solutions is given by the Nernst equation:

$$V = \frac{RT}{ZF} \ln \frac{C_2}{C_1} \qquad (2.1)$$

where V is the transmembrane potential, R is the gas constant, T is the absolute temperature, Z is the valence of the ion, F is the charge of one mole of electrons (Faraday's constant) and the C's represent the ionic concentrations in the two solutions.

If more than one ion is involved, as is the case for the frog muscle, the concentration gradient for each of the ions will tend toward a different equilibrium transmembrane potential. For the concentrations listed above, the equilibrium potentials for potassium, sodium and chloride would be -102 mV, +45 mV and -88 mV respectively. These potentials represent the voltage of the inside of the muscle cell with respect to the interstitial fluid. Of course, these electric potentials cannot all exist across the membrane at the same time. The Goldman equation provides an approximate value for the net membrane electric potential:

$$V = \frac{RT}{ZF} \ln \left[\frac{P_K[K]_o + P_{Na}[Na]_o + P_{Cl}[Cl]_i}{P_K[K]_i + P_{Na}[Na]_i + P_{Cl}[Cl]_o} \right] \quad (2.2)$$

where the P's represent the membrane permeability for each ion and the quantities in square brackets represent the ion concentrations. The "o" subscripts indicate concentrations outside the muscle cell (interstitial fluid), and the "i" subscripts indicate concentrations inside the muscle cell (intracellular fluid). Typical permeability coefficients for frog skeletal muscle are 0.2 nm/s, 20 nm/s and 40 nm/s for potassium, sodium and chloride respectively (Taccardi, 1983). The Goldman equation yields a membrane potential of -88 mV (inside relative to outside) for these conditions. This compares well with the value of -90 mV typically measured for frog muscle cells. This potential is often referred to as the "resting" potential of the membrane.

The membrane potential of -90 mV does not provide an equilibrium condition for either potassium or sodium ions. Thus, both ions will tend to leak through the membrane continuously. The "driving" potential for sodium is much stronger than for potassium, but since the membrane permeability is much greater for potassium, the two ions experience a similar rate of leakage (but in opposite directions). Despite the continual leakage of these two ions, their concentrations remain constant on both sides of the membrane. This is accomplished by means of an active pumping mechanism. The ion pump is made up of large protein molecules embedded in the plasma membrane and consumes energy to transport the ions against the electrochemical barrier. The hydrolysis of ATP provides the required energy. The sodium-potassium pump moves three sodium ions out of the cell for every two potassium ions it pumps into

the cell. The result is a zero net flow of sodium and potassium across the plasma membrane and maintenance of the membrane resting potential of -90 mV.

2.1.3 The Action Potential

The resting potential of a cell membrane may be altered by providing an electrical, mechanical or chemical stimulus. All of these stimuli result in changes in membrane permeablility to one or more of the species of ions. This in turn alters the membrane potential. By placing negative charge (with an electrode) just outside the cell, the membrane can be depolarized from its resting -90 mV. The depolarization effect is observed to decrease with increasing distance from the site of intervention.

If the stimulus is strong enough and the membrane depolarization passes a threshold (between -60 and -40 mV) a sudden change occurs in the membrane permeability. The permeablility to sodium increases by a factor of 5000 and these ions rush into the cell. This causes the local membrane potential to rise very rapidly and the inside of the cell becomes positive with respect to the outside. The membrane then becomes very permeable to potassium ions and they quickly flow out of the cell, bringing the membrane potential back down towards its resting state. Quantitative description of these phenomena was pioneered by Hodgkin and Huxley (1952). These sudden changes in transmembrane potential are called an "action potential". The total duration of the depolarization and repolarization processes is approximately one millisecond. The membrane ion pump works continuously to maintain the "resting" concentrations.

Once an action potential has been stimulated, it propagates along the cell membrane. This propagation occurs because the sudden changes of an action potential cause large unbalanced charge distributions. These charge distributions have electric fields associated with them that can be "felt" at a distance. The electric fields are strong enough to depolarize the cell membrane past the action potential threshold in areas adjacent to the initial site of depolarization. Thus, the depolarization region spreads across the membrane in all directions (except where depolarization has already occurred). A repolarization wave follows the depolarization front. Action potentials can propagate along large myelinated nerve fibres at speeds up to 130 m/s. Typical speeds for muscle fibres are 3 to 5 m/s (Guyton, 1981). The trailing edge of the repolarization region therefore trails behind the leading edge of the action potential by approximately 4 mm (for a muscle fibre). For the large myelinated nerve fibres, the spatial extent of the action potential can be as great as 13 cm.

2.1.4 Membrane Ion Currents

The sudden changes in transmembrane potential associated with the propagation of an action potential result in electric potentials that can be detected outside the cell. These external electric voltages have been measured with needle electrodes and disc electrodes applied to the skin surface. The signal measured from skeletal muscle cells is called the electromyogram and has been used for many years in the diagnosis and research of neuromuscular diseases.

For the present research, it is the ion currents rather than the resulting potential differences that are of primary interest. Ions are charged particles. When they move, they constitute an electrical current. Any isolated current element results in the generation of a magnetic field. It is the magnetic field of ion currents due to action potentials propagating in skeletal muscle fibres that is the object of study in this research.

2.2 MAXWELL'S EQUATIONS

2.2.1 The Origins of Maxwell's Equations

Maxwell's equations provide a complete and concise description of classical electromagnetic phenomena. Each of the equations summarizes a set of experimental observations that were made by many scientists. To establish an understanding of these equations, the origin of each will be briefly considered in the following paragraphs.

The first equation makes the connection between the distribution of electric charge and the resulting electric field vector. It can be derived from Gauss' law (or Coulomb's law). From experimental measurements, Gauss developed a law that stated that the integral, over a closed surface, of the component of the electric field vector (normal to the surface) was proportional to the total charge enclosed by the surface:

$$\oint_S \vec{E} \cdot d\vec{A} = \frac{1}{\epsilon_o} \int_V \rho \, dv = \frac{Q}{\epsilon_o} \tag{2.3}$$

In this equation, E is the electric field vector, dA is an element of area on the surface of integration, rho is the charge density function, dV is an element of volume within the volume of integration and Q is the total enclosed charge. The constant of proportionality is the permittivity of free space.

To obtain the differential form of Gauss' law, the divergence theorem is used:

$$\oint_S \vec{f} \cdot d\vec{A} = \int_V \vec{\nabla} \cdot \vec{f} \, dv \tag{2.4}$$

Applying this to Equation (2.3) yields

$$\int_V [\vec{\nabla} \cdot \vec{E} - \rho/\epsilon_o] \, dv = 0 \tag{2.5}$$

This result must hold for any volume, even infinitely small volumes. Therefore, the integrand must be zero for all points in space and

$$\vec{\nabla} \cdot \vec{E} = \rho/\epsilon_o \tag{2.6}$$

In the most general case, Maxwell's first equation is written in terms of the electric displacement:

$$\vec{\nabla} \cdot \vec{D} = \rho \quad \text{where} \quad \vec{D} = \vec{\epsilon}\,\vec{E} \tag{2.7}$$

The electric displacement is related to the electric field vector by the permittivity tensor.

The second of Maxwell's equations is the magnetic equivalent of the first. The major difference is, of course, that there is no magnetic equivalent of electric charge. Gauss' law for magnetic fields states that the total magnetic flux through a closed surface must be zero:

$$\Phi_B = \oint_S \vec{B} \cdot d\vec{A} = 0 \tag{2.8}$$

where B is the magnetic flux density. Once again, the divergence theorem is employed:

$$\oint_S \vec{B} \cdot d\vec{A} = \oint_V \vec{\nabla} \cdot \vec{B}\, dv = 0 \tag{2.9}$$

Since this must be true for a volume of any size, it must be concluded that

$$\vec{\nabla} \cdot \vec{B} = 0 \tag{2.10}$$

This is Maxwell's second equation.

Maxwell's third equation was derived from Faraday's law of induction. Faraday's experiments showed that the electromotive force (voltage) induced in a circuit is proportional to the time derivative of the magnetic flux through it:

$$\mathcal{E} = \frac{-d\Phi_B}{dt} \qquad \text{where} \qquad \Phi_B = \oint_S \vec{B} \cdot d\vec{A} \tag{2.11}$$

The electromotive force around the circuit is just the integral of the electric field around the path of the circuit:

$$\mathcal{E} = \oint_c \vec{E} \cdot d\vec{l} = -\int_s \frac{\partial \vec{B}}{\partial t} \cdot d\vec{A} \qquad (2.12)$$

The "c" and "s" integral subscripts indicate that the integral is over a circuit path or over a surface area. The element of circuit path length in the first integral was represented by dl. This equation is put in its differential form by making use of Stokes' theorem, which relates the integral of a function over a closed path to the integral of the curl of the function over the area enclosed by the path:

$$\oint_c \vec{f} \cdot d\vec{l} = \int_s (\vec{\nabla} \times \vec{f}) \cdot d\vec{A} \qquad (2.13)$$

Applying Stokes' theorem to Faraday's law yields

$$\oint_c \vec{E} \cdot d\vec{l} = \int_s (\vec{\nabla} \times \vec{E}) \cdot d\vec{A} = -\int_s \frac{\partial \vec{B}}{\partial t} \cdot d\vec{A} \qquad (2.14)$$

and therefore,

$$\int_s \left[\vec{\nabla} \times \vec{E} + \frac{\partial \vec{B}}{\partial t} \right] \cdot d\vec{A} = 0 \qquad (2.15)$$

Since the surface S and bounding circuit C are arbitrary, the integrand must be zero at all points in space, and

$$\vec{\nabla} \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \qquad (2.16)$$

This is Maxwell's third equation.

The last of Maxwell's equations is derived from Ampere's law, which states that the integral of the magnetic flux density, B, around a closed path is proportional to the total current flowing through the

area enclosed by the path:

$$\oint_C \vec{B} \cdot d\vec{l} = \mu_o \oint_S \vec{J} \cdot d\vec{A} = \mu_o I \qquad (2.17)$$

In this equation, J is the current density and the constant of proportionality is the permeability of free space. Using Stoke's theorem again,

$$\oint_C \vec{B} \cdot d\vec{l} = \oint_S (\nabla \times \vec{B}) \cdot d\vec{A} = \mu_o \oint_S \vec{J} \cdot d\vec{A} \qquad (2.18)$$

and therfore,

$$\oint_S (\nabla \times \vec{B} - \mu_o \vec{J}) \cdot d\vec{A} = 0 \qquad (2.19)$$

Once again, this integration must hold for any surface S and bounding circuit C and therefore the integrand must be zero at all points in space:

$$\nabla \times \vec{B} = \mu_o \vec{J} \qquad (2.20)$$

In the most general case, this is written in terms of the magnetic intensity:

$$\nabla \times \vec{H} = \vec{J} \quad \text{where} \quad \vec{B} = \bar{\bar{\mu}}\, \vec{H} \qquad (2.21)$$

The magnetic intensity, H, is related to the magnetic flux density by the permeability tensor. Equation (2.21) is an incomplete form of Maxwell's fourth equation.

Equations (2.7), (2.10), (2.16) and (2.21) summarized electromagnetic theory at the time of Maxwell. The brilliant contribution of Maxwell was the observation that the system of equations was incomplete. It was consistent only for steady state conditions. The divergence of Equation (2.21) is

$$\vec{\nabla}\cdot\vec{\nabla}\times\vec{H} = \vec{\nabla}\cdot\vec{J} = 0 \tag{2.22}$$

and is equal to zero because the divergence of the curl of any vector field is zero. Then, by considering the continuity equation for charge,

$$\frac{\partial\rho}{\partial t} + \vec{\nabla}\cdot\vec{J} = 0 \tag{2.23}$$

it becomes evident that there could be no time variation of the charge density. To allow time variation of the charge density, Maxwell made use of Coulomb's law:

$$\frac{\partial}{\partial t}(\vec{\nabla}\cdot\vec{D}) = \frac{\partial\rho}{\partial t} \tag{2.24}$$

The continuity equation can then be written as

$$\vec{\nabla}\cdot\frac{\partial\vec{D}}{\partial t} + \vec{\nabla}\cdot\vec{J} = 0 \quad \text{or} \quad \vec{\nabla}\cdot\left[\frac{\partial\vec{D}}{\partial t} + \vec{J}\right] = 0 \tag{2.25}$$

and Ampere's law could then be written as

$$\vec{\nabla}\cdot\vec{\nabla}\times\vec{H} = \vec{\nabla}\cdot\left[\frac{\partial\vec{D}}{\partial t} + \vec{J}\right] = 0 \tag{2.26}$$

The extended version of Ampere's law and Maxwell's fourth equation can

thus be written as

$$\nabla \times \vec{H} = \vec{J} + \frac{\partial \vec{D}}{\partial t} \tag{2.27}$$

This final equation is now consistent with the continuity equation for time-dependent fields. Maxwell called the time derivative of the electric displacement the "displacement current". Without this modification, derivation of a wave equation was impossible. In fact, it was Maxwell that predicted that light was an electromagnetic wave.

In the above discussion, it has been shown how Maxwell's equations grew out of the experimentally based laws of Gauss, Faraday and Ampere. Maxwell's equations are summarized below for convenience.

$$\begin{array}{ll} (1)\quad \nabla \cdot \vec{D} = \rho & (3)\quad \nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \\ (2)\quad \nabla \cdot \vec{B} = 0 & (4)\quad \nabla \times \vec{H} = \vec{J} + \frac{\partial \vec{D}}{\partial t} \end{array} \quad \left| \quad \begin{array}{l} \vec{D} = \vec{\varepsilon}\,\vec{E} \\ \vec{B} = \vec{\mu}\,\vec{H} \\ \vec{J} = \vec{\sigma}\,\vec{E} \end{array} \right. \tag{2.28}$$

2.2.2 Source Terms in Maxwell's Equations

The right hand sides of Maxwell's equations represent the sources of electric and magnetic fields. The first equation describes how static (or quasi-static) charge distributions result in electric fields. The second equation expresses that there are no isolated magnetic sources (poles). The third equation accounts for the generation of electromotive force by a changing magnetic field (Faraday's law). The final equation deals with the generation of magnetic fields from current distributions and rapidly changing electric fields (displacement

currents).

To evaluate the contribution of displacement currents to biomagnetic fields, first consider the current density source, J. The current density may be considered to be made up of two parts. The first, called the "impressed current density" arises directly from the electrochemical activity of the excitable membrane and is given the "i" superscript. The second component is known as the "volume current density" and is the current flow due to electric fields in a conducting medium. Thus, for an homogeneous and isotropic medium,

$$\vec{J} = \vec{J}^{i} + \vec{\vec{\sigma}}\,\vec{E} \tag{2.29}$$

Maxwell's fourth equation may then be written as

$$\vec{\nabla} \times \vec{B} = \mu\left[\vec{J}^{i} + \sigma\vec{E} + \epsilon\frac{\partial \vec{E}}{\partial t}\right] \tag{2.30}$$

If we consider signals of frequency f, the bracketed expression in Equation (2.30) may be expanded as

$$J_{o}^{i}\sin(2\pi ft + \theta) + \sigma E_{o}\left[\sin(2\pi ft) + \frac{2\pi\epsilon f}{\sigma}\cos(2\pi ft)\right] \tag{2.31}$$

The last term in the square brackets is due to the displacement current. The importance of the displacement current is determined by comparing the amplitude of the cosine with that of the sine. For frequencies of 1 kHz, the ratio determining the cosine amplitude is found to have a value ranging from 0.01 to 0.1 for various organ tissues (Tripp, 1983). This ratio decreases with decreasing frequency. Since there is very

little power in biomedical signals for frequencies above 1 kHz, the displacement current will provide a very small contribution to the magnetic field. It is usual in biomagnetic modelling to neglect the displacement current and assume that the tissues are purely resistive (Tripp, 1983; Plonsey, 1981).

For the frequencies typical of bioelectric phenomena, it can also be shown that Faraday induction also provides a negligible contribution to the electric field (Tripp, 1983). Thus, both time derivatives may be removed from Maxwell's equations:

$$\vec{\nabla} \times \vec{E} = 0 \tag{2.32}$$

$$\vec{\nabla} \times \vec{B} = \mu \left[\vec{J}^{i} + \sigma \vec{E} \right] \tag{2.33}$$

Bioelectric signals vary so slowly that the electric and magnetic fields may be considered quasi-static. Because of this, biological tissues appear transparent to magnetic fields of frequencies below the 1 kHz range. The weak induced eddy currents cause very little attenuation or distortion of biomagnetic signals.

2.3 SOURCE MODELS

2.3.1 Homogeneous Media

To allow comparison of bioelectric and biomagnetic measurements the solution of Equation (2.32) will be sought in terms of the electric potential, V, and the solution to Equation (2.33) will be sought in terms of the magnetic flux density, B. The electric field is the negative gradient of the electric potential. Since the curl of the gradient of a function is zero, Equation (2.32) is satisfied. Since the displacement current was shown to be insignificant in bioelectric phenomena, it can also be shown that the continuity equation (Equation 2.23)) reduces to

$$\vec{\nabla} \cdot \vec{J} = 0 \tag{2.34}$$

If the current density is split into "impressed" and "volume" components (Equation (2.29)) then,

$$\vec{\nabla} \cdot \vec{J} = 0 = \vec{\nabla} \cdot \left[\vec{J}^i + \sigma \vec{E} \right] = \vec{\nabla} \cdot \vec{J}^i - \sigma \nabla^2 V \tag{2.35}$$

and therefore,

$$\nabla^2 V = \frac{\vec{\nabla} \cdot \vec{J}^i}{\sigma} \tag{2.36}$$

An equation of similar form may be obtained by taking the curl of Equation (2.33) and taking into consideration that the permeability of biological tissues is very close to that of free space:

$$\nabla \times \nabla \times \vec{B} = \mu_o \left[\nabla \times \vec{J}^i + \sigma \nabla \times \vec{E} \right] \tag{2.37}$$

The last term is zero because of Equation (2.32). By making use of the vector identity

$$\nabla \times (\nabla \times \vec{A}) = \nabla(\nabla \cdot \vec{A}) - \nabla^2 \vec{A} \tag{2.38}$$

and taking into consideration Maxwell's second equation, Equation (3.37) may be written as

$$\nabla^2 \vec{B} = -\mu_o \nabla \times \vec{J}^i \tag{2.39}$$

Equations (2.36) and (2.39) are in the form of Poisson's equation. The right hand sides, which represent the sources, both involve the impressed current. Thus, it is from the impressed current that both the electric potential and magnetic flux density arise.

In an infinite or semi-infinite homogeneous and isotropic medium, the solutions to Equations (2.36) and (2.39) may be derived by defining a vector potential as described by Plonsey (1981). The solutions are

$$V = \frac{-1}{4\pi\sigma} \int_v \frac{\nabla \cdot \vec{J}^i}{R} \, dv \tag{2.40}$$

and

$$\vec{B} = \frac{\mu_o}{4\pi} \int_v \frac{\nabla \times \vec{J}^i}{R} \, dv \tag{2.41}$$

where R is the distance between the source point and the point of

measurement.

It is important to note that only the impressed current density appears in these solutions. The impressed current is the component of the current density that is due to the electrochemical activity of the membrane and cannot be accounted for by the presence of an electric field in a conducting medium. Neither the electric potential nor the magnetic flux density depend on the volume current. Also, since the divergence and curl of a vector field are independent of each other, the electric potential and magnetic flux density must be independent (Plonsey, 1981).

2.3.2 Inhomogeneous Media

In the previous section, the electric potential and the magnetic flux density resulting from an arbitrary "impressed current density" was studied in an infinite (very large compared to the size of the current source), homogeneous and isotropic medium. Unfortunately animal bodies are not infinite or homogeneous. However, a body can be modelled as a finite number of homogeneous regions, each of finite extent. As stated previously, the permittivities and permeabilities of biological tissues vary little and are very close to the values of free space. It is the conductivity that varies significantly from one tissue to the next. A reasonable biological model consists of a finite collection of regions, each with a uniform conductivity. The volume current will have zero divergence and curl everywhere except at the boundaries and will therefore only contribute to the electric potential and magnetic field at these boundaries (Tripp, 1983).

A boundary between two regions of differing conductivity may be modelled as one homogeneous region with a source in place of the boundary. This has been shown by Tripp (1983) by replacing the impressed current density with the total current density in Equations (2.40) and (2.41). Stoke's theorem was then applied to the volume integral of the "volume current density" to convert it into an integral over the boundaries between regions of differing conductivity. The solutions for the electric potential and magnetic flux density are

$$V = \frac{1}{4\pi\sigma}\left[\oint_V \vec{J}^i \cdot \vec{\nabla}\left(\frac{1}{R}\right) dv + \sum_k (\sigma_k^o - \sigma_k^i) \oint_{S_k} V_k d\vec{A} \cdot \vec{\nabla}\left(\frac{1}{R}\right)\right] \tag{2.42}$$

and

$$\vec{B} = \frac{\mu_o}{4\pi}\left[\oint_V \vec{J}^i \times \vec{\nabla}\left(\frac{1}{R}\right) dv + \sum_k (\sigma_k^o - \sigma_k^i) \oint_{S_k} V_k d\vec{A} \times \vec{\nabla}\left(\frac{1}{R}\right)\right] \tag{2.43}$$

where the surface integrals are over the k boundaries between regions of differing conductivities. The "o" and "i" superscripts of the conductivities indicate that it is the conductivity on the "outside" or "inside" of the boundary. The forms of these equations show that the boundaries act as sources in addition to the direct contribution of the impressed current.

2.3.3 Secondary Sources

It is important to note that the surfaces in the summations of equations (2.42) and (2.43) include membrane surfaces at the cellular level and the macroscopic level (the surfaces of organs and the body). These cellular and macroscopic contributions may be separated. The

cellular surface terms may be considered with the impressed current term to yield a net effective source of volume currents in the surrounding regions that contain no active membranes.

The combination of the impressed current and the cellular surface sources is called the primary source. The macroscopic surface sources are therefore called secondary sources. Both the primary and secondary sources provide important contributions to the electric potential and magnetic flux density measured at a distance from these sources.

Although it is the impressed current that drives both the cellular surface terms and the secondary sources, it has been shown that the direct contribution of the impressed current is small compared to the cellular surface contribution (Tripp, 1983). The impressed current term may therefore be dropped. In the case of an excitable membrane in an infinite uniform medium, it was found that only the impressed current contributed to the external fields. However, when there are boundaries between regions of markedly different conductivities, the impressed current is found to only act as the "motor" behind the cellular and macroscopic surface sources and does not contribute directly to the external fields.

2.3.4 The Cylindrical Cell

As a first approximation, many excitable cells may be considered to be cylinders with circular cross-section. Although this may be a significant simplification for skeletal muscle fibres, it is a good approximation for nerve axons. In Figure 2.1a, the depolarization

region of an action potential (propagating from left to right) has been shown for a cylindrical cell in an infinte conducting medium.

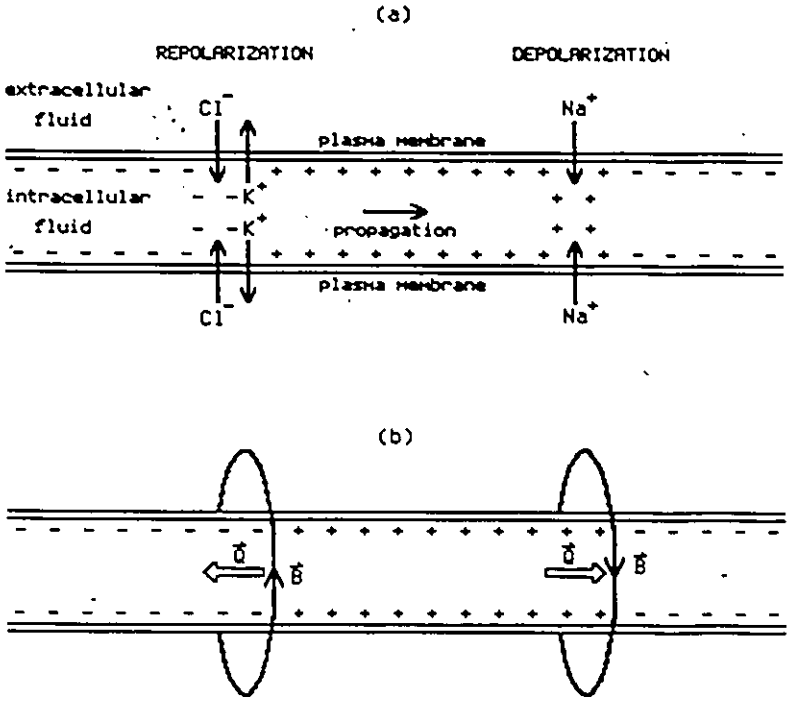


Figure 2.1 : Illustration of the pair of spatially separated current dipoles used to model the electromagnetic source characteristics of an action potential propagating along the membrane of a cylindrical cell in an infinite homogeneous medium.

In the case of a cylindrical cell of infinite length in an infinite homogeneous medium, Tripp (1983) has shown that the cellular surface terms of equations (2.42) and (2.43) can be represented by a pair of current dipoles. The impressed current flows in a small volume and has radial symmetry. The area of this volume is determined by the

size of the depolarization region of the action potential and the thickness is just the thickness of the membrane. For these reasons, the impressed current contributes very little to the external fields.

Calculations by Swinney & Wikswo (1980) have shown that by far the strongest current density associated with the depolarization region of an action potential is to be found on the inner surface of the membrane. It is an axial current density in the direction of action potential propagation. Calculations also showed that this current density could be modelled by a single current dipole, Q, as shown in Figure 2.1b. An azimuthal magnetic field is generated by this current dipole (also shown in Figure 2.1b).

Associated with the repolarization region of the action potential is a second area of high current density just inside the membrane. This current may be modelled by a second current dipole which is of the same amplitude as the first, but points in the opposite direction. Thus, as an action potential propagates past a measurement point, external to the cylindrical cell, a biphasic azimuthal magnetic field should be detected.

All current densities, other than the one modelled by the current dipoles discussed above, provide negligible contributions to the external magnetic field. As discussed in section 2.3.1, the volume currents external to the cell will provide no contribution. And as was discussed in section 2.3.2, any boundary will provide a much stronger contribution than the impressed current. Since there are no other boundaries in the case of an isolated cylindrical cell, the boundary provided by the cell membrane will provide the strongest source.

Plonsey (1981) has shown that for one model of a single fibre in an extensive extracellular medium, the electric and magnetic fields are dependent on the transmembrane potential in a similar way. The electric potential can be determined from complete specification of the magnetic field. In this case, magnetic measurements should not provide any information not already available in electric potential measurements.

2.3.5 Muscle As A Magnetic Source

No modelling specific to the generation of magnetic fields by skeletal muscles could be found in the literature. Although skeletal muscle is much more complex than any of the idealized structures discussed in this chapter, consideration of the electrophysiology of excitable cells, basic electromagnetic theory and recent biomagnetic modelling, has provided insight into biomagnetic processes that may be applicable to muscle. A summary of conclusions from the discussion of this chapter, that may apply to muscle, is presented in the next few paragraphs.

First, muscle fibres are known to be surrounded by a plasma membrane that will support the propagation of action potentials as for a nerve axon. It was found in section 2.1 that it is the membrane ion currents associated with the propagation of action potentials that generate biomagnetic signals.

Because of the small variation in electric permittivity and magnetic permeability from free space values and the high conductivity of biological tissues and because action potentials contain relatively low frequency components, the electric displacement current and Faraday

induction contribute little to the magnetic and electric fields. Bioelectric phenomena are quasi-static and biological tissues are essentially transparent to magnetic fields. Because of large variations in electrical conductivity, biological tissues are not transparent to electric fields.

In a "semi-infinite" medium (large with respect to source dimensions), the volume current density does not yield a net electric or magnetic field. In general, the electric potential arises from the divergence of the impressed current and the magnetic flux density comes from its curl. Thus, without consideration of "real" sources, the electric potential and magnetic flux density should be independent of each other.

The boundaries between regions of different electrical conductivity act as sources that are related to the electric potential at the boundary due to volume current flow. Contributions from such boundaries, to the external electric and magnetic fields, can be much larger than from the impressed current. In fact, the impressed current density contribution is negligible compared to the contribution due to the boundary effect of the cell membrane. The impressed current contributes only in that it drives the volume currents that make the significant contributions at the boundaries.

For an isolated cylindrical excitable cell in a semi-infinite medium, the cell membrane contribution dominates generation of the magnetic field. The axial current densities on the inside of the cell membrane can be modelled as a pair of current dipoles propagating along with the depolarization and repolarization regions of the action potential. The resulting magnetic field is observed to be a biphasic (in

time) azimuthal field.

All of the points summarized in the previous paragraphs have some importance when considering muscle as a biomagnetic source. Muscle fibres have excitable limiting membranes that propagate action potentials like nerve axons. As a first approximation, muscle fibres may be considered cylindrical in shape. It should therefore be expected that the magnetic field from a muscle have a biphasic azimuthal component.

Muscles are usually composed of groups of many parallel fibres. If the fibres are synchronously activated, the magnetic fields of the individual fibers will superpose. Within the muscle the azimuthal components of the magnetic fields from adjacent fibres will tend to cancel out. However, external to the muscle, the fields from individual fibres will reinforce. An azimuthal field of the same orientation as for the individual fibres will exist external to the bundle of fibres. Thus, a whole muscle should yield a magnetic flux density with a biphasic azimuthal component as predicted and observed for an isolated nerve axon.

It might be considered possible for the influx of calcium ions from the sarcoplasmic reticulum to contribute to the net magnetic field. The calcium ion concentration required to activate the contractile machinery of a muscle fibre is very low. The intracellular concentration of calcium ions varies from 0.1 to 10 micromoles per litre during a contraction-relaxation cycle, while sodium concentrations range between 20 and 120 millimoles per litre (Carlson & Wilkie, 1974). It is therefore unlikely that there is an appreciable calcium ion current density (compared with sodium and potassium).

Finally, there is the potential contribution of the transverse tubule system to the external magnetic field. The action potentials that propagate along the outside of a muscle fiber also propagate in to the centre of the fibre along a system of tubules whose walls are continuous with the external plasma membrane. If the tubules are oriented radially and the propagation of an action potential can be modelled as a current dipole pair then there should be no net magnetic field. A radial current distribution does not result in a net magnetic field. However, if the tubule structure is more complicated, a measurable magnetic field may result.

In section 2.3.4 it was stated that the electric potential and magnetic field were not independent for the case of a single fibre in an infinite medium. This may, however, not be the case for skeletal muscle. The greater structural complexity of muscle may yield independent contributions to the external electric and magnetic fields. A detailed study of transverse tubule structure and some careful modelling will be required to determine their contribution to external electric potential and magnetic flux density measurements.

CHAPTER 3

DATA ACQUISITION HARDWARE

To study the magnetic signal from an active skeletal muscle, a suitable magnetometer had to be chosen. To allow connection of this signal with other muscle variables, other physiological signals had to be monitored. Of particular importance were the electric potential near the surface of the muscle and the force generated by the muscle. Controlled study of the muscle required that it be externally stimulated (rather than allowing voluntary contractions). These components of the data acquisition system are shown schematically in Figure 3.1.

To eliminate the labour-intensive analysis of chart recordings, data was collected directly by a computer. A custom interfacing instrument was required to coordinate the data acquisition process. The following sections provide a detailed description of each of the functional units of the data acquisition system required for this research.

3.1 MAGNETOMETER

3.1.1 Choice Of A Magnetometer

The MMG from human muscles under voluntary control has been shown to contain magnetic action flux pulses with peak-to-peak amplitudes of 3 to 100 picotesla (pT) (Cohen & Givler, 1972; Reite et al.,

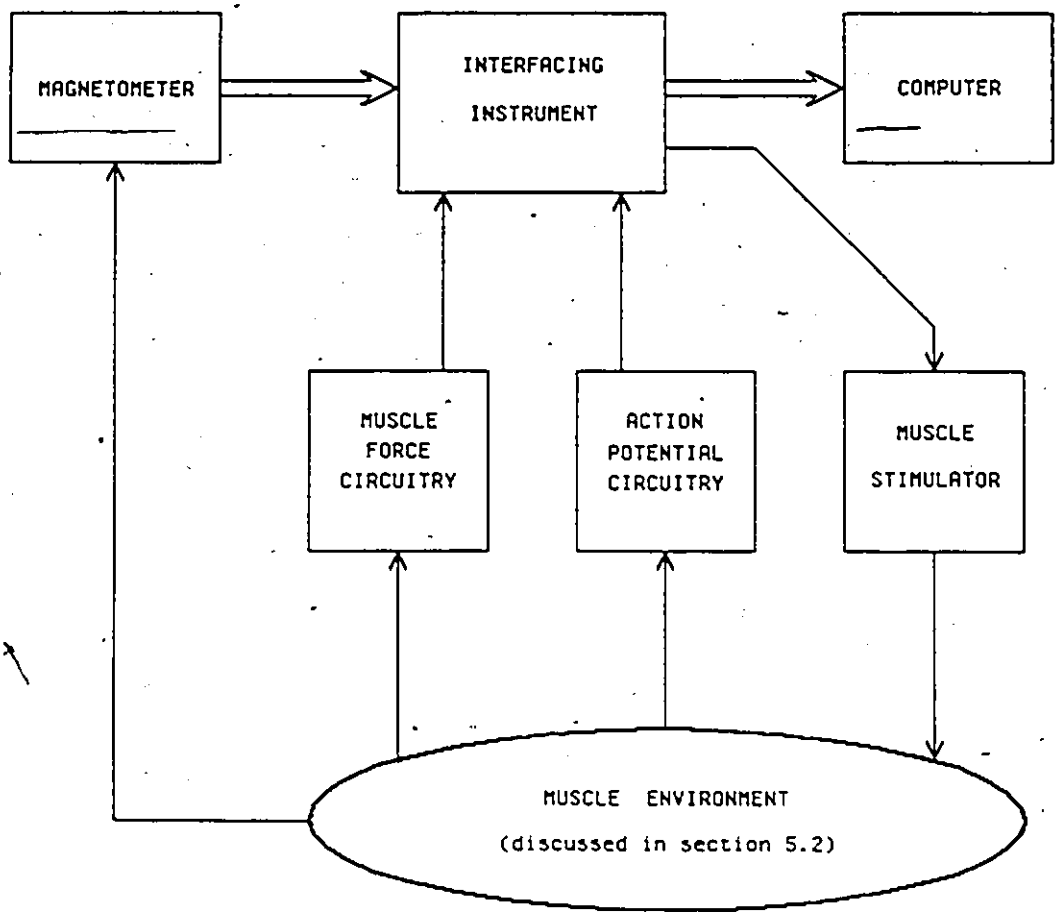


Figure 3.1 : Block diagram showing major functional components of the data acquisition system.

1976; Williamson & Kaufman, 1981; Koga & Nakamura, 1983). To put the size of these signals into perspective, consider that the earth's static field is approximately 70 microtesla (a million times stronger). The alpha rhythm from the brain generates a signal of 1 pT amplitude and auditory evoked responses are on the order of 0.1 pT.

There are presently many types of commercially available magnetic sensors. In choosing a magnetometer, the primary considerations are sensitivity and frequency response. Of secondary importance is the spatial resolution. Once the measurement technology has been chosen, the required spatial resolution can usually be accommodated by design modifications.

Sensitivity may be defined as the noise power (per unit of frequency) generated by the sensor itself. Alternately, the sensitivity may be defined by the root-mean-squared (RMS) magnetic signal power that yields a sensor output equal to the RMS noise from the sensor (measured in "picotesla per root hertz"). This sensitivity is a function of frequency for most magnetometers for frequencies below 10 Hz. For frequencies above 10 Hz, the noise of many magnetic sensors is white, and the sensitivity is constant.

It is useful to compare the expected signal amplitude with the total RMS noise of the magnetometer over the bandwidth of the signal. The MMG amplitude was expected to be at least 20 pT. To be sure of not missing any of the MMG signal, a bandwidth of 2 kHz was chosen for the measurements (0 Hz to 2000 Hz). If the sensitivity of the magnetometer was $0.1\ pT/\sqrt{Hz}$, then multiplying by the square root of the bandwidth gives a total noise amplitude of 4.5 pT. This would be a signal-to-noise ratio of only 4.4 (13 dB). A higher value would be desirable.

Hall-effect, fluxgate and magnetic resonance magnetometers are generally not suitable for biomagnetic measurements. The Hall-effect and fluxgate magnetometers are not sensitive enough. Commercial fluxgate magnetometers can detect fields as small as 100 pT at freqencies

below 1 kHz (Zimmerman, 1983). The magnetic resonance magnetometers are sensitive, but generally offer poor spatial resolution and are not sensitive to the direction of the magnetic flux (Zimmerman, 1982).

Induction-coil magnetometers make use of tightly wound coils of fine wire and may have a ferrite or air core. The induced voltage is proportional to the time rate of change of the magnetic field and is thus preferentially sensitive to higher frequencies. Integration with respect to time is required to yield the original magnetic signal. From data published by Williamson & Kaufman (1981), an air core induction coil with a frequency response from 10 Hz to 2 kHz would have a sensitivity of 17 pT. For this frequency band, the air core induction coil magnetometer would not be sensitive enough for measurement of the MMG.

Commercial sensors based on superconducting quantum interference devices (SQUIDs) offer the greatest sensitivity presently available. These devices involve a superconducting toroid with a Josephson junction, a weak link, that displays a very sensitive voltage-current characteristic. The junction may be biased by several methods. The rf-biased SQUID provides a wide frequency response (starting at DC) and is therefore useful for almost any biomagnetic measurement. There are many reviews of SQUID magnetometer principles, operation and applications (Williamson & Kaufman, 1981; Zimmerman, 1983). With a SQUID magnetometer in a shielded chamber and a bandwidth of 10 Hz to 2 kHz, the sensitivity should be approximately 0.4 pT. In an urban setting, without shielding, the smallest detectable field should be approximately 10 times larger.

The wide bandwidth and the extreme sensitivity of the SQUID magnetometer make it the preferred sensor for this research (it is presently used for most biomagnetic research). The major drawback of the SQUID magnetometer is its reletively large purchase and running costs. To keep the SQUID components superconducting, it must be bathed in liquid helium in a cryogenic dewar. The low temperature is maintained by allowing the helium to boil off at atmospheric pressure. Thus, there is the constant expense of replacing, or collecting and liquifying the helium.

Two new technologies are likely to have an impact on biomagnetic measurements in the near future. The first involves a magnetostrictive material applied to an optical fiber. When placed in a magnetic field, the length of the fiber is caused to change very slightly by the magnetostrictive coating. Minute changes in length can be detected by comparing the phase of laser light that has passed through this fiber with that from a standard reference fiber (Giallorenzi, Bucaro, Dandridge & Cole, 1986). The second involves the development of ceramic materials that display superconducting properties at temperatures as high as 80 degrees Kelvin (Lemonick, 1987). Use of such materials in SQUID magnetometers would allow use of liquid nitrogen for cooling and would simplify dewar designs. The use of liquid nitrogen would significantly reduce the running cost of a SQUID magnetometer.

3.1.2 CTF Systems SQUID Magnetometer

The SQUID magnetometer was chosen as the sensor best suited for measuring the MMG because of its high sensitivity and wide bandwidth (it is the standard measurement tool in the field of biomagnetism). Funds were not available for the purchase of a SQUID magnetometer. It was therefore necessary to locate a research team equipped with a SQUID magnetometer and negotiate access to it. The Department of Earth and Planetary Studies at the Erindale College campus of the University of Toronto was found to have a SQUID magnetometer. Use of the magnetometer was arranged and guided by Dr. N. Sugiara and Dr. W. Pearce.

The SQUID magnetometer was manufactured by CTF Systems Inc. of Port Coquitlam, British Columbia, Canada. This magnetometer featured a stabilized point contact toroidal SQUID, a low noise RF amplifier and a digital control module with a non-standard parallel digital interface. The dewar reservoir volume (for liquid helium) was 10 liters.

The main body of the dewar was cylindrical with a diameter of 35 cm and length of 110 cm. The dewar was supported with its cylindrical axis in the vertical direction. A "tail" projected approximately 19 cm from the bottom surface of the dewar. It had a diameter of 12 cm. As shown in Figure 3.2, the pickup coil assembly was located in the dewar tail.

The coil system of this magnetometer was a coaxial asymmetric second-order gradiometer. Three coils were mounted one above the other with their axes coincident with the axis of the cylindrical dewar. The "sensing" coil consisted of 5 turns of 2.54 cm diameter and was mounted 1.3 cm from the outside surface of the bottom of the dewar. The other

two "gradient" coils were of 5.68 cm diameter, of fewer turns and mounted 4.00 cm and 8.00 cm above the primary coil. The middle coil had a turns-area product twice that of the other two coils and was wound in the opposite direction to yield a second order spatial derivative of any magnetic field from a distant source.

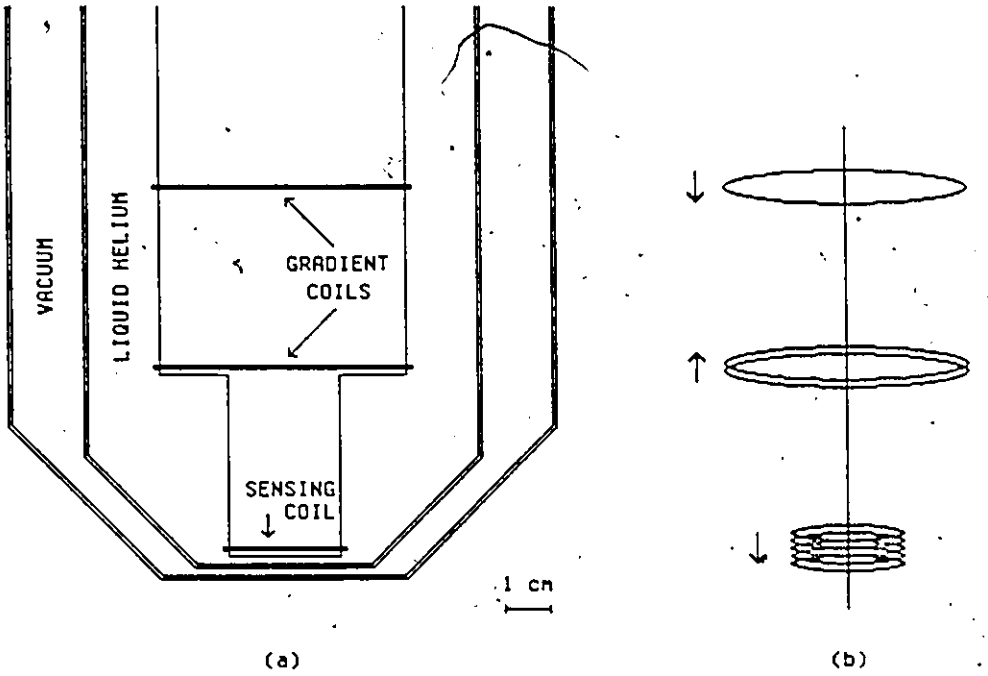


Figure 3.2 : Detector coils of the SQUID magnetometer. Part (a) shows the coils wound on a substrate suspended in liquid helium in the tail of the magnetometer dewar. Part (b) shows the number of turns, the diameters and orientations of the three coils. This is an assymetric second order gradiomter coil configuration.

For small magnetic sources close to the end of the dewar tail, the magnetic flux density measured by the magnetometer may be considered

as the average flux density over the cross-sectional area of the sensing coil. The other two coils are far enough away that only a small fraction of the flux lines from the source will pass through them. The upper two coils serve only to cancel out fields from distant sources such as the earth's field and fields from ventilation motors and fluorescent light ballasts. Thus, for this research, the magnetometer signal was considered to represent the average flux density over the cross-sectional area of the sensing coil due to the activity of the nearby muscle preparation.

A SQUID control and signal processing unit, also designed and manufactured by CTF Systems Inc. was used to run the magnetometer. The SQUID was biased with an RF signal of 19 MHz. The demodulated SQUID ouput was sampled at a rate of 100 kHz. The DC offset of the SQUID output could be adjusted with a control in the feedback path of the RF loop. A 13-BIT analogue-to-digital converter (ADC) and flux quantum counter made up the 32 BITs of data made available (for each sample) to a digital filter. Resample circuitry allowed for digital data output rates up to 32768 samples per second. The digital filter frequency had to be selected to prevent aliasing of the resampled digital output. For the measurements discussed in this thesis, a resample rate of 8192 samples per second and a digital filter (3-pole Butterworth) cutoff frequency of 2048 Hz were used.

The 32-BIT digital representation of each sample of the magnetic field was made available to a serial and a parallel digital interface. The serial interface did not allow a high enough BIT transfer rate, so the parallel interface was used. The parallel digital interface

provided 8 data lines, 8 control lines and 8 grounded lines on a 24 pin connector. The interface did not conform to the IEEE-488 standard. The four data bytes for each sample were made available on the parallel interface in the order: most significant byte to least significant byte.

The least significant BIT of the magnetometer digital output corresponded to an average flux density of 0.006134 pT at the sensing coil. A biomagnetic signal of 20 pT would therefore occupy only 12 BITs of the magnetometer output. It was therefore decided that 16 BITs would provide adequate range for the MMG measurements and the two more significant bytes were discarded.

The magnetometer was supported with its tail pointing down, one half meter above the floor of a shielded room. The shielding consisted of two thin layers of high permeability Mumetal. A plywood support for the dewar enclosed a space around the dewar tail that measured approximately 31 cm wide, 30 cm deep and 36 cm high. The dewar tail was accessible from inside the shielded room through a 31 cm (wide) by 25 cm (high) window in the front wall of the plywood support. It could also be accessed from outside the shielded room by way of a 4 cm by 3.5 cm port in the side wall of the room.

3.1.3 Magnetic Noise Management

As for most university and hospital laboratories, the laboratory in which this research was performed (Erindale College, Mississauga) was contaminated with magnetic noise from many sources. For frequencies above 10 Hz, the noise appeared to be dominated by "white" processes (flat power spectrum) except for a large peak at 60 Hz. Below 10 Hz, a

1/f characteristic was observed. The greatest problem was the elimination of the large 60 Hz peak due to the power distribution system, ventilation motors and fluorescent light ballasts within the building.

The SQUID magnetometer was maintained in a shielded room. The walls of this room consisted of two thin layers of high permeability Mumetal. This shielding was optimized to minimize the earth's static field within the room. It unfortunately had little effect on alternating fields.

The 60 Hz noise problem was significantly reduced by performing experiments with only incandescent lighting (no fluorescent lights). It was also found that there was less 60 Hz noise late at night. This is presumably because very few researchers perform experiments at night and there are fewer machines operating in the building. The hours between 11:00 pm and 7:00 am were found to be the quietest.

The most effective tool against the 60 Hz noise signal was found to be a large copper ring. It was constructed of gauge #2 multistranded cable and was 61 cm in diameter (with one soldered joint). This ring was suspended so that the magnetometer sensing coil was at its center. The orientation of the ring was then adjusted to minimize the 60 Hz noise in the magnetometer output. Lenz's law states that the net magnetic flux through a conducting loop will tend to be cancelled by an electrical current set up in the loop. Thus, the copper loop was able to cancel out the magnetic flux from distant sources. The magnetic signal from a small source near the center of the loop will not be affected because its flux lines will not extend far enough to link with the copper loop.

The copper ring was so successful at cancelling 60 Hz magnetic noise that experiments could be performed while making use of fluorescent lighting. Experiments continued to be performed at night because the copper ring was found to be more effective at night. Typical noise spectra under ideal experimental conditions are shown in Figure 3.3.

There was a piece of equipment in the laboratory building that switched on for 2 minutes at a time, every 15 minutes. It generated such a strong 60 Hz signal that no measurements could be made while it was on. This piece of equipment was never located and identified, but was probably part of the building's ventilation system.

3.2 ANALOGUE HARDWARE

3.2.1 Force Transducer Circuitry

The force transduction system (described in section 5.2.3) involved a pressure transducer with a bridge type strain gauge. It had a sensitivity of 50 microvolts per millimeter of mercury and per volt of excitation. Either a direct or alternating excitation could be used with this transducer. Unfortunately the alternating excitation was picked up by the magnetometer. A direct voltage of 10.00 volts was therefore used to bias the bridge. As shown in Figure 3.4a, the 10.00 v reference was derived from the output of a dual FET stabilizer buffered by an LF351 operational amplifier (National Semiconductor). When the gate is biased by a resistor in the source lead the drain current is quite insensitive to the drain voltage. When two or more FETs are

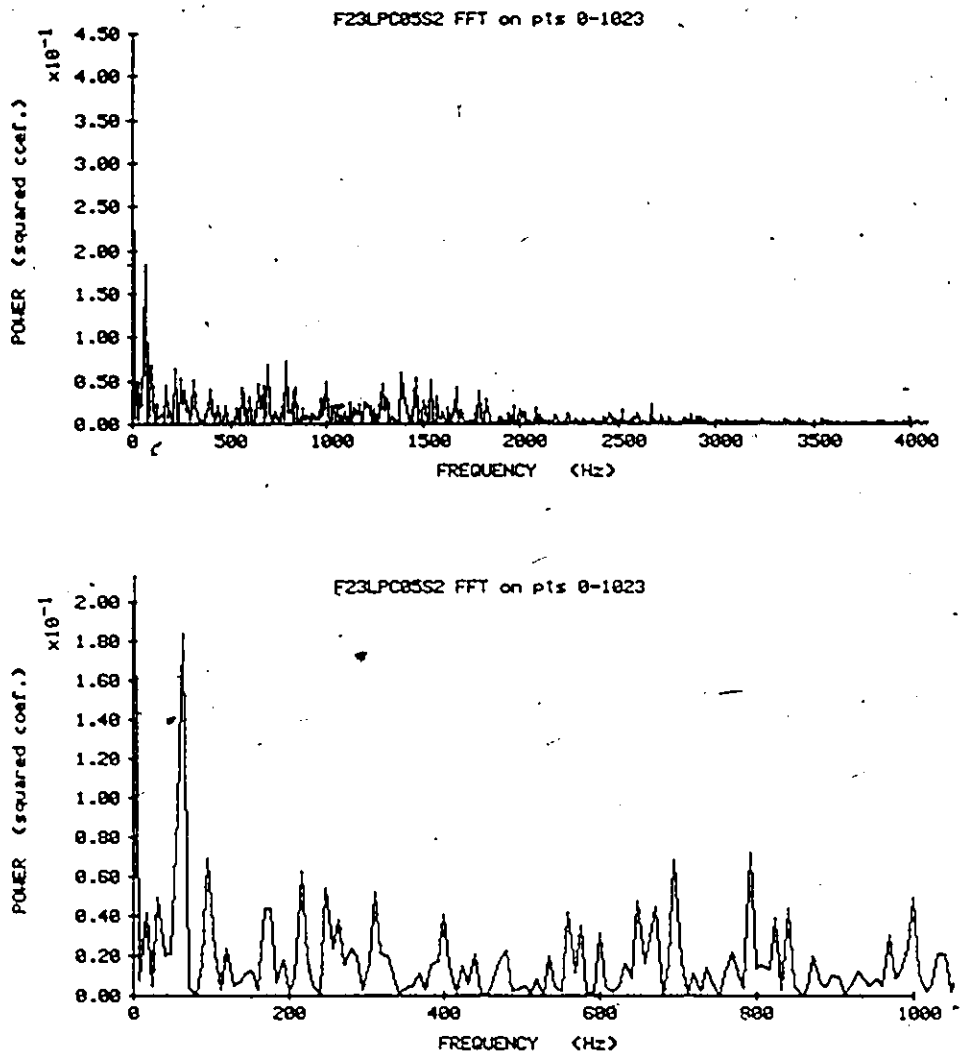


Figure 3.3 : Shown here, on two frequency scales, is a typical magnetic noise spectrum. This power spectrum was calculated from a noise signal that was measured with the apparatus adjusted to minimize the RMS noise.

chained as in Figure 3.4a the voltage drop across the source resistor becomes essentially independent of the supply voltage (MacHattie, 1972). A zero temperature coefficient can be obtained for moderate temperature

excursions with the drain current in the neighbourhood of 0.35 mA. Finally, the bridge was balanced with a 100 kohm trimpot as shown in Figure 3.4a.

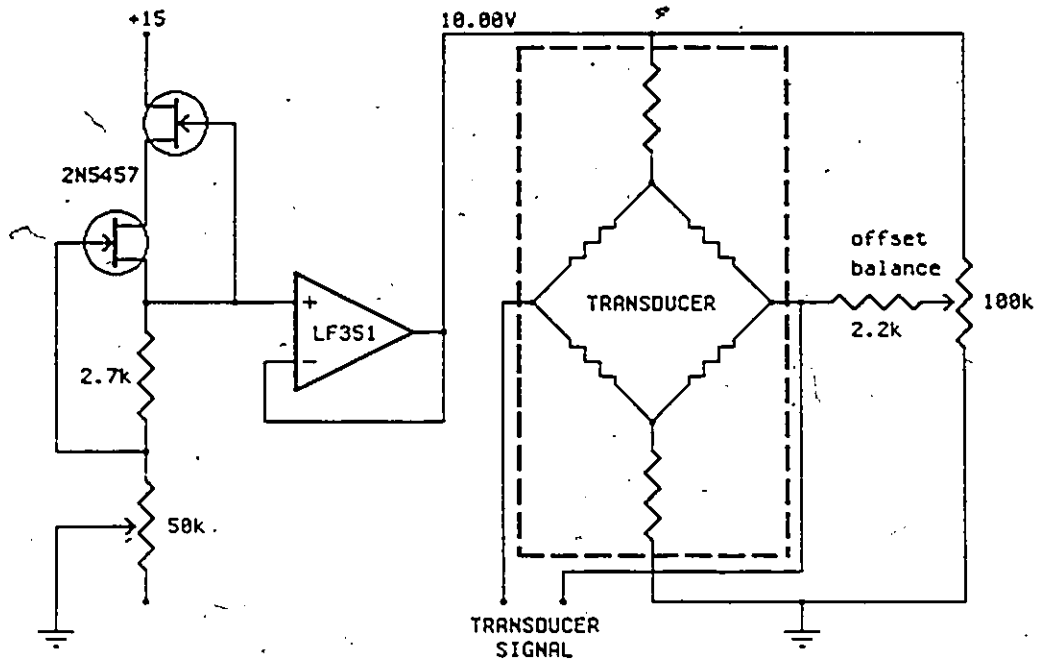
The strain gauge output was amplified by a factor of 100 with an Analog Devices AD524 instrumentation amplifier. A following LF351 operational amplifier provided an additional gain adjustable from 1 to 6. As shown in Figure 3.4b, the final stage provided DC offset adjustment and a gain of 2. No low-pass filtering was required to condition the output of this transducer.

3.2.2 Action Potential Circuitry

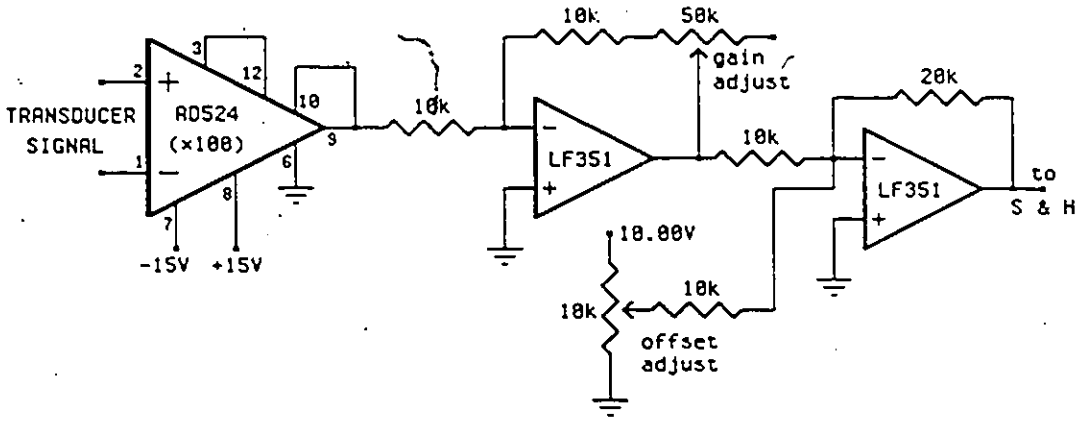
The AD524 instrumentation amplifier was also used as the preamplifier for the action potential measurements. As described in section 5.2.4, the action potential signal was derived from three fine silver wire electrodes. These wires were connected to the inputs of the instrumentation amplifier which provided a gain of 100. An additional adjustable gain of 5 to 50 was provided by a five-pole Butterworth low-pass filter with 3 dB cutoff frequency of 1.975 kHz. This circuitry is shown in Figure 3.5.

Carbon film resistors of 1 % tolerance and polystyrene capacitors of 2 % tolerance were used to determine the time constant of each stage. The 3 dB frequency was measured to be 1960 Hz. A test signal was found to drop by -29.8 dB in going from 2.5 kHz to 5.0 kHz.

A single pole high-pass filter with 3 dB cutoff frequency of 8 Hz was inserted between the instrumentation amplifier and the 5-pole low-pass filter. This was done to remove the low frequency fluctuations



(a)



(b)

Figure 3.4 : Force transducer circuitry. Part (a) shows the transducer biasing circuitry and part (b) shows the preamplifier and amplifier circuits.

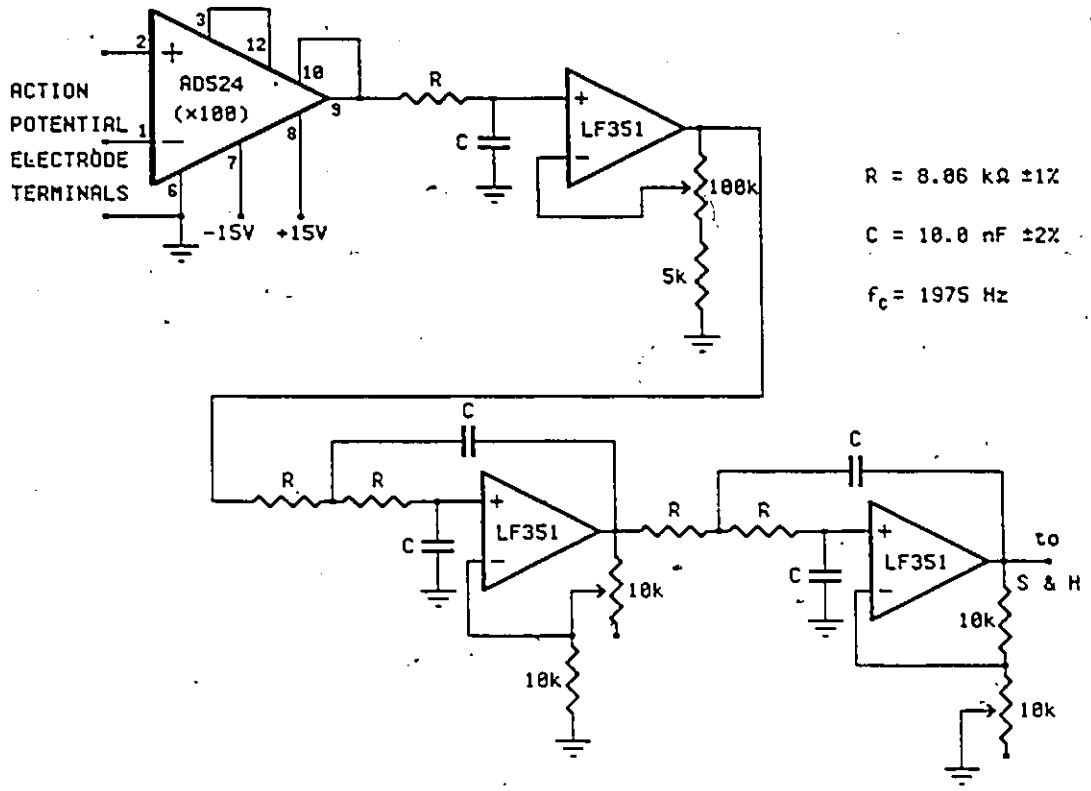


Figure 3.5 : Whole muscle action potential circuitry. Shown here are the instrumentation amplifier and the 5-pole low-pass Butterworth filter.

in the electrode half-cell potentials produced by adsorption of biological materials, growth of oxide films and slow electrochemical reactions. An operational amplifier was used in a unity gain follower configuration to buffer the high pass filter output.

3.2.3 Calibration Circuits

A simple circuit for generating low amplitude square waves was developed for calibration of the action potential circuitry. As shown in Figure 3.6, the output of a CMOS Schmitt-trigger NAND gate was fed back to its input with an RC delay. This yielded a 9 V square wave of 200 Hz. This square wave was used to turn a bipolar transistor on and off. When on, the transistor shorted a 2.000 V reference to ground. When off, the 2.000 volts was allowed to drop across a potential divider. The output square wave had a peak-to-peak amplitude of 1.00 mV or 10.0 mv depending on which resistor was selected to make up the bottom of the potential divider. Once again, a dual FET stabilizer was used to provide the reference voltage (MacHattie, 1972). The circuit was powered by a 9 volt NICAD battery and was packaged in a small plastic case. During calibration procedures at the beginning of each experiment, the output of this circuit was applied to the input terminals of the action potential instrumentation amplifier. The total gain of the action potential signal conditioning circuitry could then be determined.

3.3 THE COMPUTER

3.3.1 Computer Requirements

The experiments were to be performed in the laboratory at Erindale College (50 km from McMaster University) where the SQUID magnetometer was located. Several other research projects were already making use of the magnetometer. It would therefore be necessary to set up and

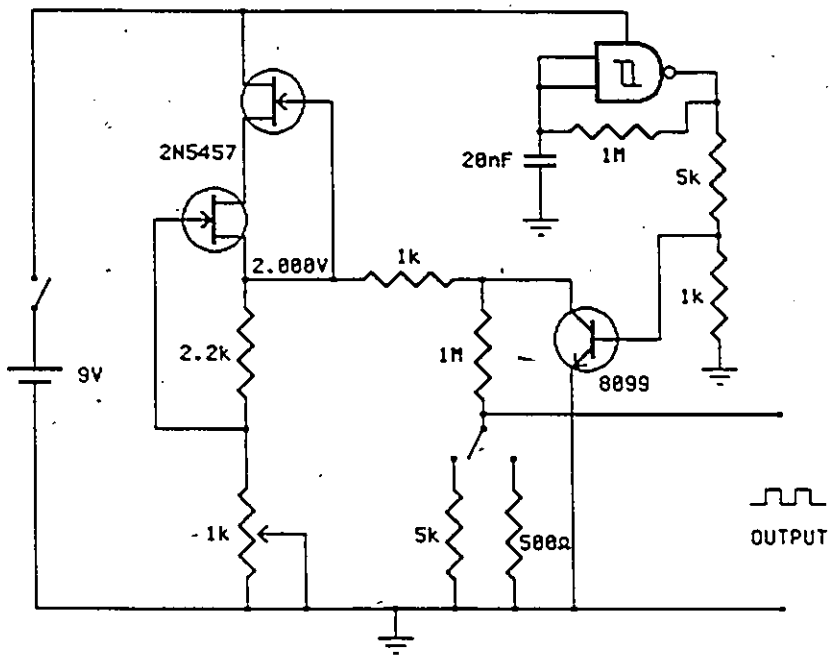


Figure 3.6 : Whole muscle action potential calibration circuit. This circuit was used to generate a 200 Hz square wave for calibration of the action potential amplifiers. The peak-to-peak amplitude of the output could be set at 1.00 mV or 10.0mV.

completely disassemble the equipment (specific to this research) for each run of an experiment. This process would be greatly simplified if the data acquisition computer were portable.

There was also the choice of whether to use two computers or one computer for both data acquisition and analysis. If used for only data acquisition, the computer would need to be little more than a sophisticated recording device. If it was required to perform the data analysis, the computer would have to provide a reasonable engineering

software environment and adequate computational power (full 16-BIT microprocessor). It was decided that a computer would be sought to perform both tasks. Requiring that the computer also be portable considerably reduced the number of products to choose from (spring of 1985).

Since the primary purpose of the computer was to perform data acquisition, it was important that the computer be equipped with a versatile digital interface such as the IEEE-488 interface. It was expected that up to 4 signals would need to be collected, each at a rate of 8192 samples per second. If each sample were represented by 8 BITs, the digital interface would have to be able to support a sustained transfer rate of 32768 bytes per second.

A mass storage device, such as a disc drive, was required to store the data for later processing. A built-in printer was also desired so that data could be checked and intermediate results printed during an experiment. This was especially important for testing the data acquisition hardware and experimental procedure. As already mentioned, a 16-BIT microprocessor and an engineering software environment were required for data anlaysis.

3.3.2 The Hewlett-Packard Integral Personal Computer (9807)

The Hewlett-Packard series 9000 Integral Personal Computer (model 207) was found to offer the best combination of features for this research. At the time of purchase (spring, 1985), the Integral Personal Computer (IPC) had been selling in the US for 6 months and had just appeared on the Canadian market. It offered a portable package the size and weight of a typical home sewing machine.

The IPC was built around an 8 MHz Motorola 68000 microprocessor. This microprocessor offers a 16-BIT data bus (32-BITs internal) and a separate 24-BIT address bus. The IPC also featured a custom 16-BIT graphics processing unit, a real-time clock, an IEEE-488 digital interface and a built-in dot matrix printer. The built-in disc drive featured the new 3.5 inch double-sided double-density Sony discs. The display was BIT-mapped and featured the new electroluminescent technology. The 512 kb RAM (expandable to 8 Mb) was expanded to 1 Mb.

One of the most appealing features of this computer was that it offered a UNIX System V operating system called HP-UX. UNIX is a large operating system that normally occupies a vast amount of disc space. In this implementation, the UNIX kernel was packaged in a 256 kb ROM and is immediately available when booting up the system. Technical BASIC and C were the languages initially available. The combination of the C programming language and the UNIX operating system provide an ideal engineering software environment.

3.3.3 Limitations of the IPC

The major limitations of this machine turned out to be inadequate documentation and poor support from Hewlett-Packard Canada. These two factors severely compounded every small problem that was encountered. Most of the HP-UX documentation was written for the whole line of HP computers that can run HP-UX and did not deal with any of the special features or limitations of the IPC. Most of this documentation was in dictionary form and contained no examples. There also appeared to be little interest in providing technical support from HP Canada. Learning

C and UNIX in this environment was very difficult.

The major hardware limitation was the absence of a direct memory access (DMA) controller. The lack of a DMA meant that the data from the IEEE-488 interface had to pass through the MC68000 on its way to a buffer in main memory. Despite the multitasking operating system, nothing else could run while the digital interface was being used for moderate speed data acquisition.

The lack of a DMA controller and a software deficiency combined to create a significant data acquisition problem. It was found that data could not be collected continuously without losing an unknown number of bytes several times each second if the rate was more than 100 bytes per second. It was suggested to Hewlett-Packard (Corvallis) several times that the problem might be caused by scheduling interuptions from the multitasking operating system. We were assured that this could not be the source of our problem. Many months were devoted to reevaluation of our custom data acquisition instrumentation. Use of an HP-1615A logic analyser finally proved that five times per second, the computer prevented interface "handshaking" for a period of 15 ms. During these 15 ms periods, hundreds of data bytes were being lost.

Equipped with proof that the computer was responsible for the lost data, a telephone call to HP (Corvallis) was greeted with the casual response that there was a software patch for this problem (a problem that HP had never admitted existed). The software patch consisted of writing the hexadecimal number 80 to RAM address E40023, performing the data acquisition and then writing the number 2 to the same address (see C source-code listing of the "acquire.c" subroutine in

Appendix A). It was explained that this would disconnect the operating system from the real-time clock during the data acquisition. The operating system would then not know when to interrupt the current process to time-share with any waiting processes (if there were any). Though it is crude, this patch was found to work.

This problem of the multitasking operating system interrupting the operation of the IEEE-488 interface would not have occurred if a DMA controller had been included in the computer and should have been handled by the software of a special Device I/O Library purchased from Hewlett-Packard to perform the data acquisition. Another piece of software, a C language system call called rtprio, should also have been able to solve the problem. This function allows a process to change its priority so that it will not be subjected to the time slicing of normal round-robin time sharing. A description of rtprio exists in the documentation, but it does not exist in either the 1.0.0 or the 5.0 release of the C language package.

3.4 INTERFACING INSTRUMENT

3.4.1 Purpose and General Description

The purpose of the interfacing instrument was to coordinate the acquisition of the magnetic, electric and force signals from an active skeletal muscle. This involved processing the analogue force and electric potential signals, converting them to digital forms, and coordinating the delivery of these signals and the magnetometer signal to the

computer. A schematic diagram of the many functions required to perform this task is shown in Figure 3.7. Arrows representing the flow of data and control signals from one functional block to another have been included in this diagram.

Sample-and-hold circuits, an analogue multiplexer, an analogue-to-digital converter (ADC) and digital control logic were used to convert the analogue voltage signals to 8-bit representations. The digital data path involved a non-standard parallel interface for the magnetometer, a digital multiplexer to select data from the ADC and the magnetometer and an interface for the computer. The computer interface offered a subset of IEEE-488 digital interface functions.

Also included in the interfacing instrument was a counter that was used to count the number of samples transferred to the computer. The muscle stimulator was triggered when exactly half of the data buffer was filled. As a result, the first half of the data buffer contained a "noise" record and the second half included the muscle response.

The digital integrated circuits, including the ADC, were B-series CMOS. A combination of two-sided printed circuit boards and wire-wrap were used to connect the integrated circuits. The circuit boards were mounted in an aluminum instrument enclosure. The analogue circuits were shielded from the digital circuits. Analogue circuits were powered with plus and minus 15 V and the digital integrated circuits were powered with +5 V. The ADC was powered with a 10.24 V reference.

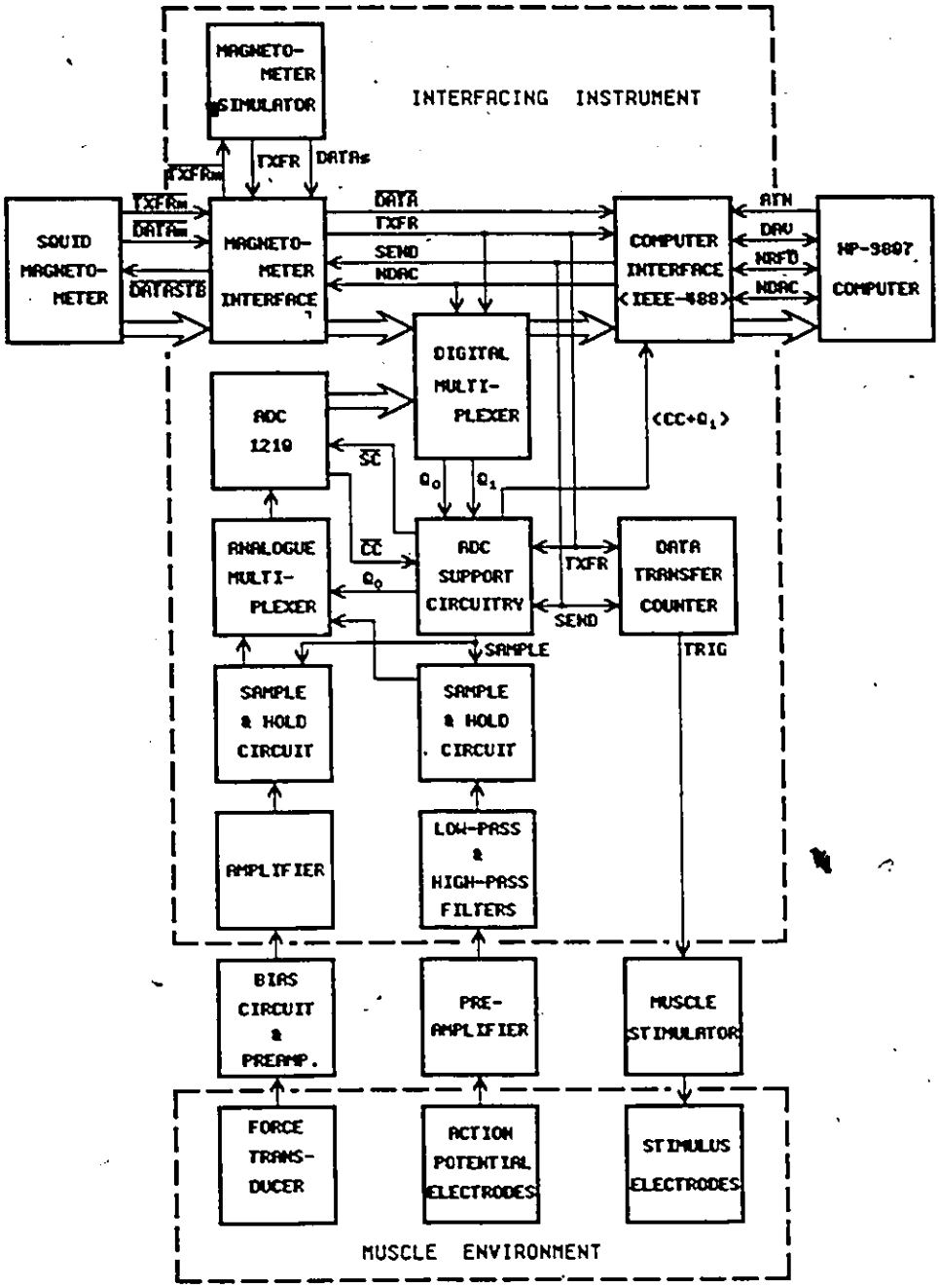


Figure 3.7 : Schematic diagram of the interfacing instrument. Arrows have been included to indicate the flow of data and control signals.

3.4.2 Computer Interface

The HP-9807 computer was equipped with a built-in IEEE-488 digital interface. A large portion of the interfacing instrument was devoted to providing a subset of the IEEE-488 standard functions (ANSI/IEEE-488, 1978). Four major circuits made up the computer interface of the interfacing instrument: talker address identification circuit, listener handshake circuit, talker handshake circuit and the transceiver circuit. It should be pointed out that the IEEE-488 interface operates on negative logic. That is, the "true" state is represented by a low voltage (0 V) and the "false" state is represented by a high voltage (>2 V). All IEEE-488 signal lines are held at 3.3 V by a resistor network, but any device connected to the interface can pull a line low (true). The interfacing instrument circuits were based on regular logic (true = +5 V and false = 0 V).

The "talker address identification circuit" constantly monitors the data lines of the IEEE-488 interface. As shown in Figure 3.8, it consists of a 4-BIT DIP-switch for setting the device address, a 4-BIT comparator, an RS flip-flop and a few inverters and AND gates. If the data lines corresponded to the address set by the DIP-switch while the ATN (attention) and DAV (data valid) lines were low and the talk command appeared on data lines 6 and 7, then this circuit recognized that the computer had addressed the interfacing circuit to start transmitting data and the TALKER flip-flop was set high. Transmission did not actually start until the SEND signal of the transceiver circuit went high. The (ATN*DAV) signal was generated in the listener handshake circuit, where the UNTALK signal was used. Any device address from 16 to 31

could be set with the DIP-switch. An address of 25 is set in Figure 3.8.

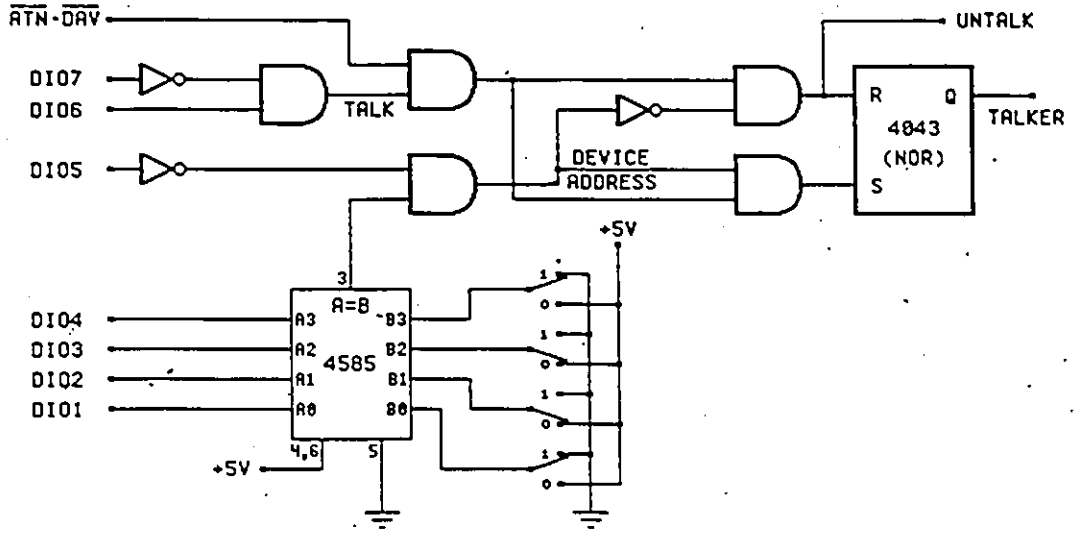


Figure 3.8 : Talker address identification circuit. The TALKER signal was set high whenever the data lines matched the address set by the DIP switch (and ATN and DAV were low).

When listening to the IEEE-488 data lines, the interfacing instrument must acknowledge receipt of every byte. It does this by pulling the NRFD (not ready for data) and NDAC (no data accepted) lines low in an appropriate sequence. The "listener handshake circuit" did this by constantly monitoring the IEEE-488 control lines. This circuit, shown in Figure 3.9., was an active listener only when the computer had pulled ATN low, indicating that it was going to issue a command. The 1 microsecond delay was required to prevent NRFD from being pulled low too

soon after the computer declared that the data lines were valid by pulling DAV low. The UNTALK signal was generated by the talker address identification circuit.

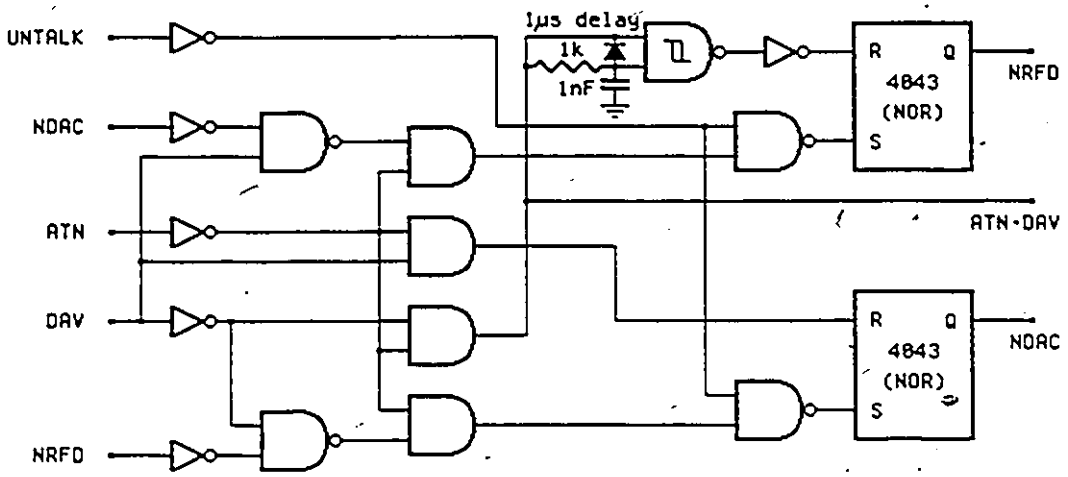


Figure 3.9 : Listener handshake circuit. This circuit was responsible for acknowledging receipt of each byte from the IEEE-488 interface when commands were being issued by the computer.

When the interfacing instrument was talking on the IEEE-488 interface, the "talker handshake circuit" (Figure 3.10) pulled the DAV signal low when either a byte of magnetometer data or a byte from the ADC was currently on the data lines. The DATA signal told this circuit when data from the magnetometer was valid and the CC (conversion complete) signal indicated when data from the ADC was valid. While not an active talker, SEND was low and the DAV flip-flop was kept in its high state.

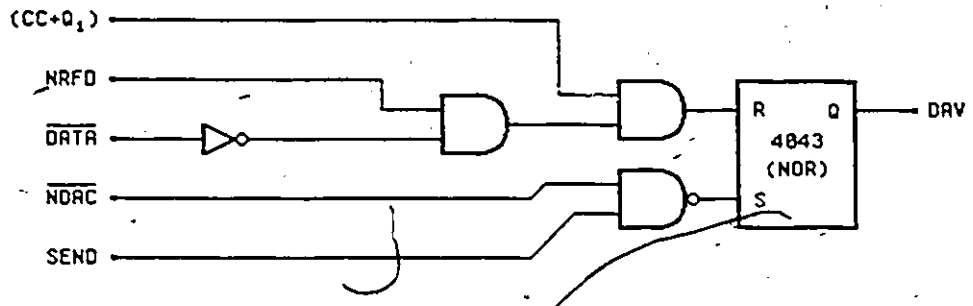


Figure 3.10 : Talker handshake circuit. When the interfacing circuit was sending data to the computer, this circuit generated the DAV (DAta Valid) signal to indicate when data was on the data lines.

The "transceiver circuit" primarily involved MC3446AP quad interface transceivers as shown in Figure 3.11. These transceivers were designed specifically for interfacing CMOS instruments with the IEEE-488 digital interface. They include the termination resistors to maintain the passive high voltage of the interface signal lines. The upper two transceivers directed the IEEE-488 data lines to the address identification circuit and sent data from the digital multiplexer to the IEEE-488 data lines. The lower transceiver handled the IEEE-488 control lines. The purpose of the few logic gates was to generate the SEND signal, which enabled the transceivers to pull the IEEE-488 signal lines low when sending data. The SEND signal did not go high until TALKER had been set high by receipt of a talk command from the computer, the ATN line had been released (high) by the computer and TXFR went high to indicate that the first sample of magnetometer data was ready. SEND stayed high (and data was transmitted) until ATN was pulled low again by

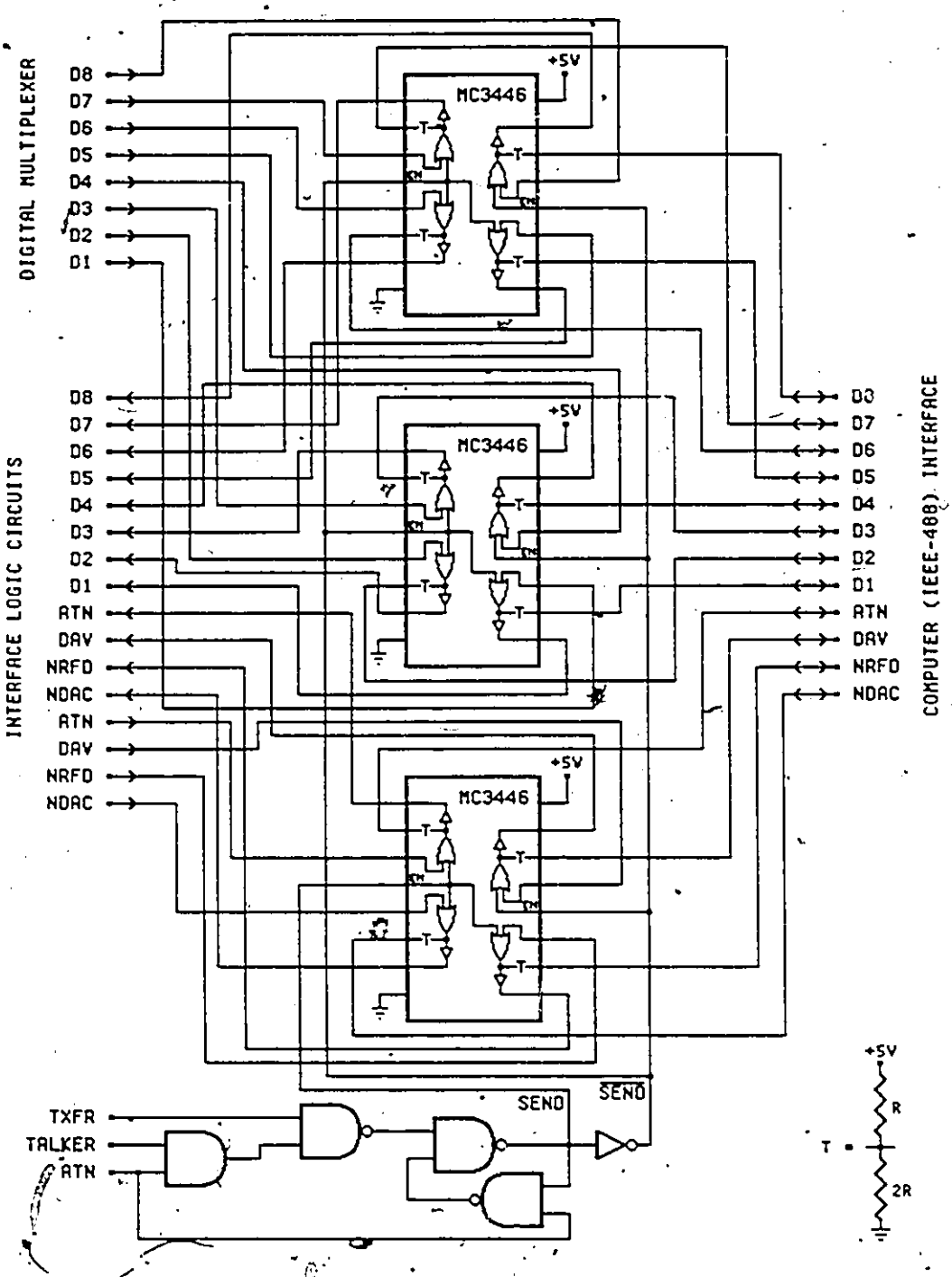


Figure 3.11 : Computer transceiver circuit.

the computer. This was done when the computer's data buffer was filled.

The digital multiplexer circuit is shown in Figure 3.12. It consisted of two 4519 quad 2-channel data selectors and two JK flip-flops. The two flip-flops counted pulses on the NDAC line from the computer. This indicated how many bytes had been read by the computer since the last TXFR signal from the magnetometer (the TXFR signal reset the counter). The Q1 signal was used to tell the data selectors which source to take data from. The first two bytes (of each group of four) were taken from the analogue-to-digital converter (ADC) and the last two bytes were taken from the magnetometer. The Q0 and Q1 signals were also used in controlling the ADC, the analogue multiplexer and the sample-and-hold circuits.

3.4.3 Magnetometer Interface

The magnetometer interface was much simpler than the IEEE-488 interface. The CTF Systems magnetometer parallel interface offered 8 regular-logic (true > 2 V) data lines and several negative-logic (true = 0 V) control lines. A 0.5 microsecond pulse (from 3.3 V to 0 V) on the TXFR control line from the magnetometer signaled the availability of each 4-byte data sample. The TXFR pulses came at regular intervals determined by the resample rate set on the front panel of the magnetometer (every 0.122 ms for 8192 samples per second). The magnetometer pulled the DATA line low when each data byte was placed on the data lines. This signal was used in the computer interface talker handshake circuit to generate the DAV signal. The DATASTB (data strobe) control line was pulled low by the interfacing instrument magnetometer

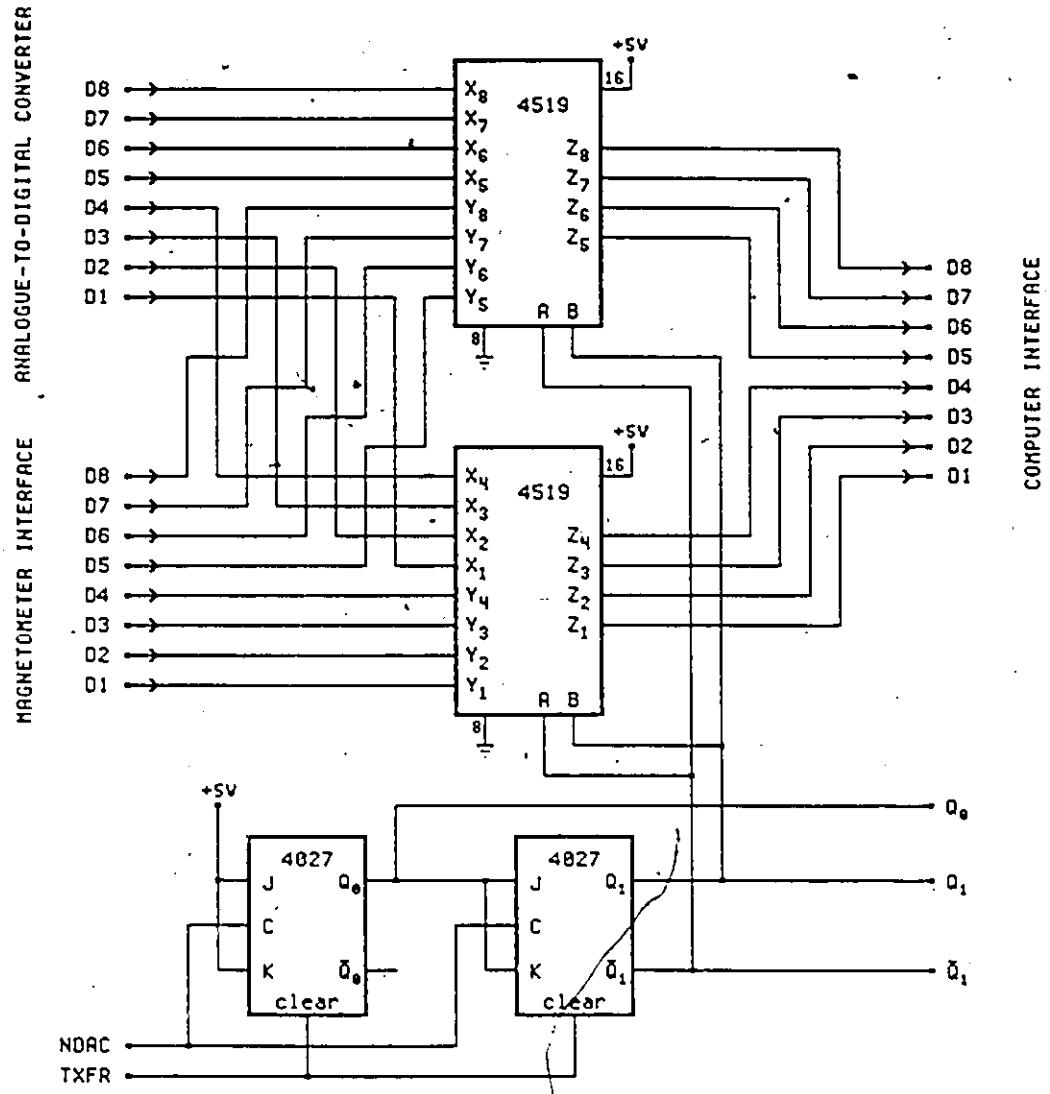


Figure 3.12 : Digital Multiplexer. The purpose of this circuit was to select data from the analogue-to-digital converter (ADC) or from the magnetometer. The first 2 bytes of every group of four were selected from the ADC and the last two from the magnetometer.

interfacing handshake circuit as shown in Figure 3.13. When the computer was collecting data the DATASTB signal was derived from the

computer letting NDAC go high to acknowledge receipt of a byte. When the computer was not collecting data, the magnetometer's own DATA signal was sent back on the DATASTB line with a 2 microsecond delay. This kept the magnetometer running continuously and avoided startup delays in the digital filter.

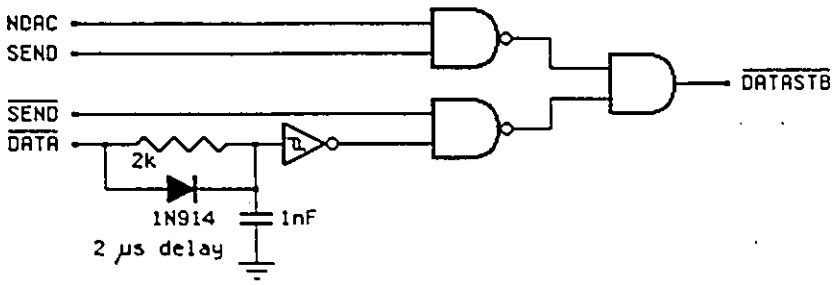


Figure 3.13 : Magnetometer handshake circuit. This circuit generated the DATASTB signal which acknowledged the receipt of each data byte from the magnetometer.

As for the IEEE-488 interface, 3 MC3446AP transceiver chips were required for the transceiver circuit of the magnetometer interface (Figure 3.14). The first two transceivers were wired for incoming data only and transferred the data directly to the digital multiplexer. The third transceiver handled the control signals.

A magnetometer simulator circuit, shown in Figure 3.15, was developed to allow data acquisition tests to be performed at McMaster University when the magnetometer was not available. The purpose of this circuit was to generate a TXFR pulse (every 200 microseconds) and

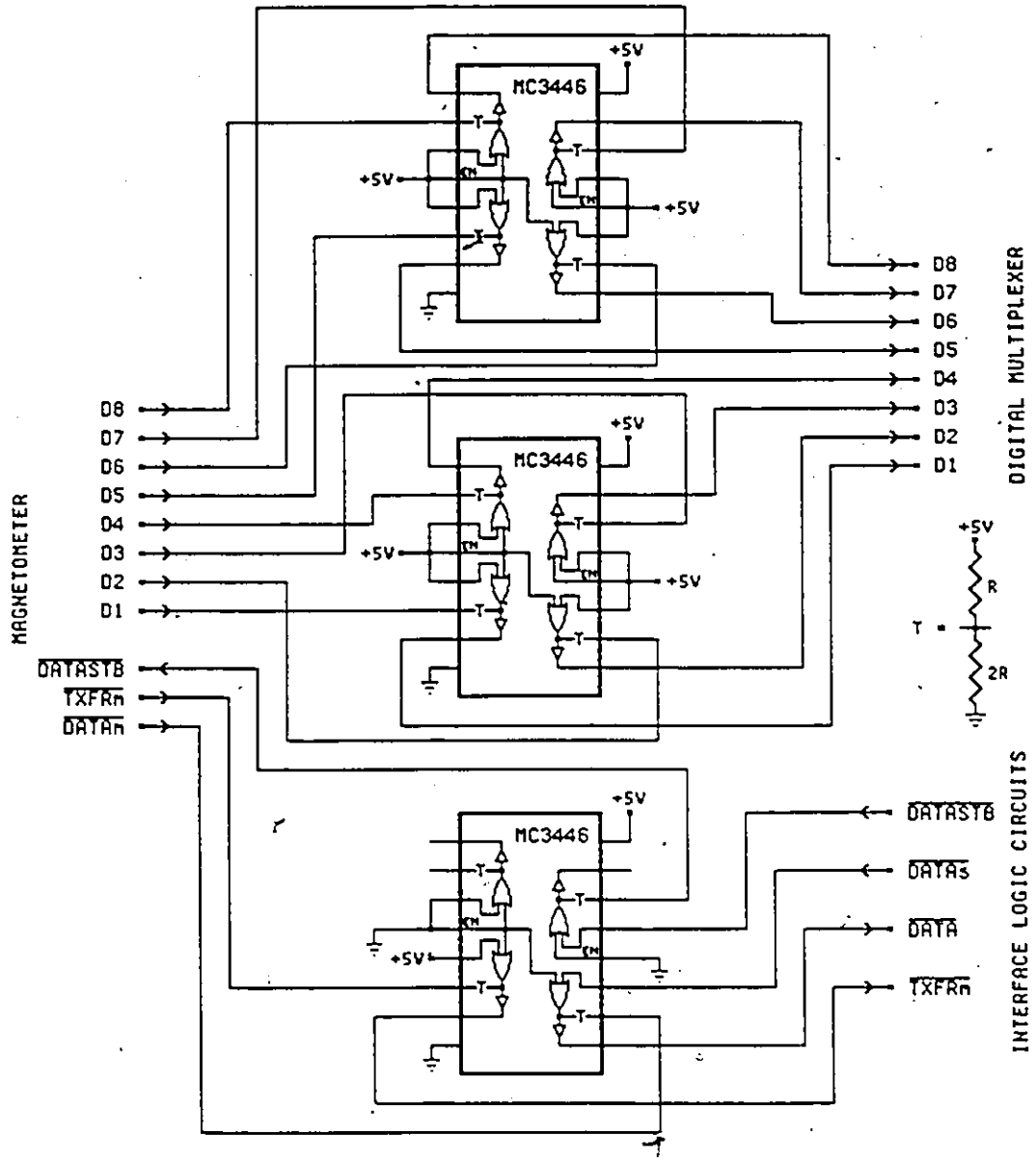


Figure 3.14 : Magnetometer transceiver circuit. As for the computer interface, the MC3446 IC was used to buffer signals received from and sent to the magnetometer.

generate a DATA signal. Both of these signals were derived from the 320 kHz clock signal used to drive the ADC (see the next section). A DPDT switch was used to disconnect this circuit from the magnetometer interface when the magnetometer was present.

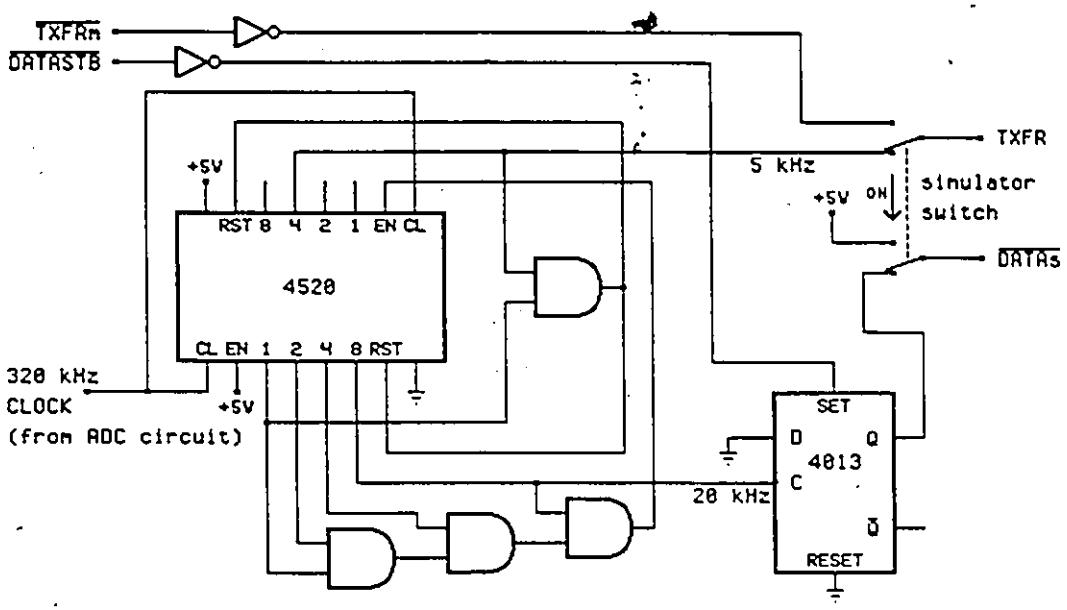


Figure 3.15 : Magnetometer simulator circuit. This circuit was designed to generate the control signals normally received from the magnetometer. This circuit was used extensively during the development of the data acquisition system when the magnetometer was not available. During data collection, this circuit was disconnected.

3.4.4 Analogue-to-Digital Conversion

The interfacing instrument housed some of the analogue signal conditioning circuitry already described in section 3.2. The force signal amplifier (gain = 2) and the electric potential low-pass Butterworth filter were both included on the analogue circuit board. Also included on the analogue circuit board were the sample-and-hold (S&H) circuits and the analogue multiplexer.

The S&H circuits are shown in Figure 3.16. The LF398A S&H circuit was used with a 11 nF polystryrene capacitor. The trimpot was used for zeroing the input offset voltage. The S&H circuits were normally kept in the sample mode (SAMPLE signal high). Only when TXFR was pulled low by the magnetometer (to signal that a magnetometer sample was ready) were the S&H circuits put in the hold state. An AD7501JN analogue multiplexer (controlled by the Q0 signal from the digital multiplexer circuit) first selected the electric potential signal for analogue-to-digital conversion and then the force signal. Because of the S&H circuits, the digital representations corresponded to the two analogue signal values at the time the TXFR signal went low.

Analogue-to-digital conversion was performed by the National Semiconductor ADC1210 which is a CMOS 12-BIT, medium speed, successive approximation converter. It required an external voltage reference and could be used in several different configurations. The ADC and its supporting circuitry are shown in Figure 3.17. It was configured to receive bipolar inputs in the range of -5.12 V to +5.12 V.

To obtain 12-BIT accuracy, the conversion rate had to be kept below 10,000 samples per second to allow the internal comparator to

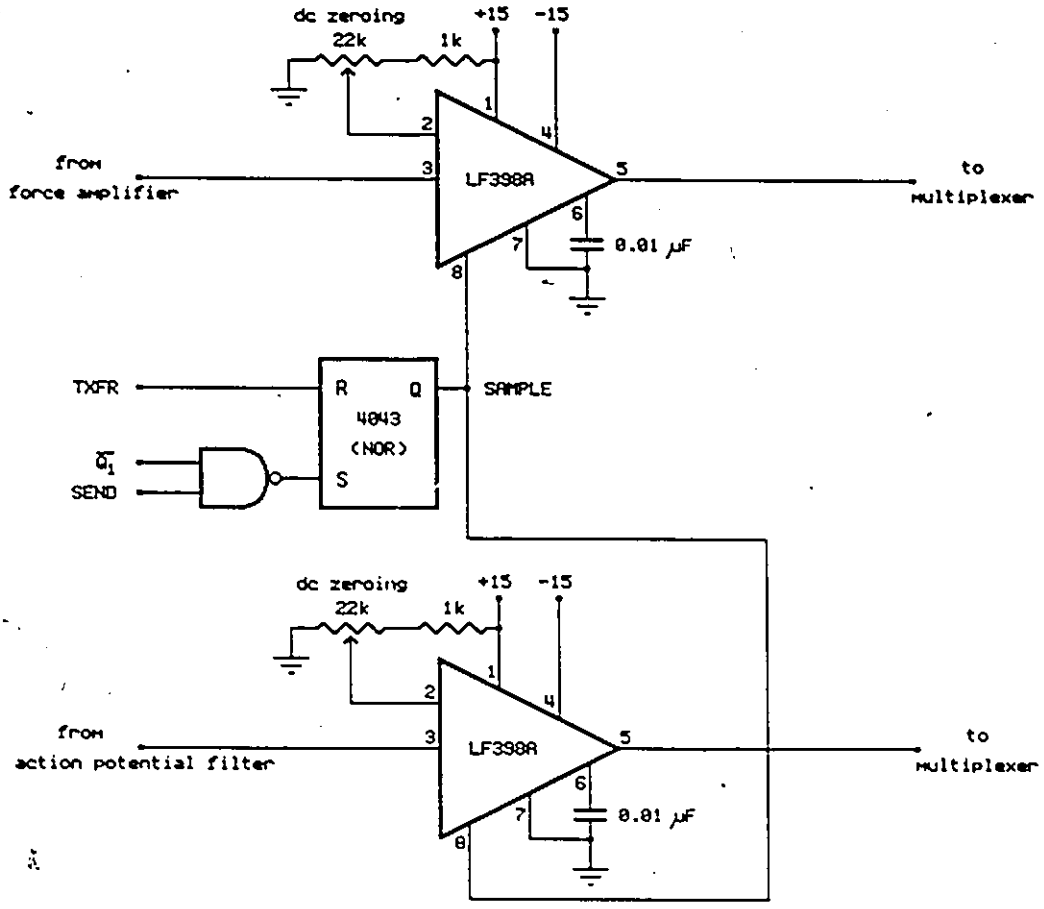


Figure 3.16 : Sample-and-hold (S&H) circuits. The S&H circuits were used to sample the twitch force and action potential signals at the beginning of each sample cycle and hold these values until the ADC had calculated digital representations for them.

settle. Conversion accuracy was reduced to 10-BITs so that the rate could be increased to 24,600 samples per second (clock frequency = 320 kHz). Unfortunately, during each sample period, only two bytes of data could be sent from the ADC. The ADC bytes were sent in place of

the upper two bytes from the magnetometer because the magnetic signal only occupied the lower two bytes of each 4-byte sample. As a result, the muscle force and the electric potential signals could each be represented by only one byte. Therefore, only the 8 most significant BITs were connected to the digital multiplexer for transmission to the computer (eventhough the converter used 13 clock cycles to generate 12-BIT conversions).

The ADC1210 was designed so that the reference voltage was also used to power the chip. The National Semiconductor LH0071 voltage reference was used to supply the 10.24 V required for the bipolar input range specified earlier. A 4.7 microfarad tantalum capacitor and a 100 nF ceramic disc capacitor were required to decouple this reference from switching transients. The digital high logic level range of the ADC was from 5.12 V to 10.24 V. Consequently, the ADC had to be buffered from the 5 V logic of the rest of the interfacing instrument with 4050 translators and bipolar transistors.

The clock signal to the ADC did not need to be very stable because its only function was to step the ADC through its successive approximation algorithm (and drive the magnetometer simulator circuit). The critical timing of analogue signal sampling was derived from the magnetometer TXFR signal. A 4584 Schmitt trigger inverter with an RC time constant in the feedback loop was found to be adequate in providing the required 320 kHz clock signal.

Two small circuits were required to provide logic functions related to ADC operation. They are shown in Figure 3.18. The first generated the SC (start conversion) signal for the ADC. It was derived

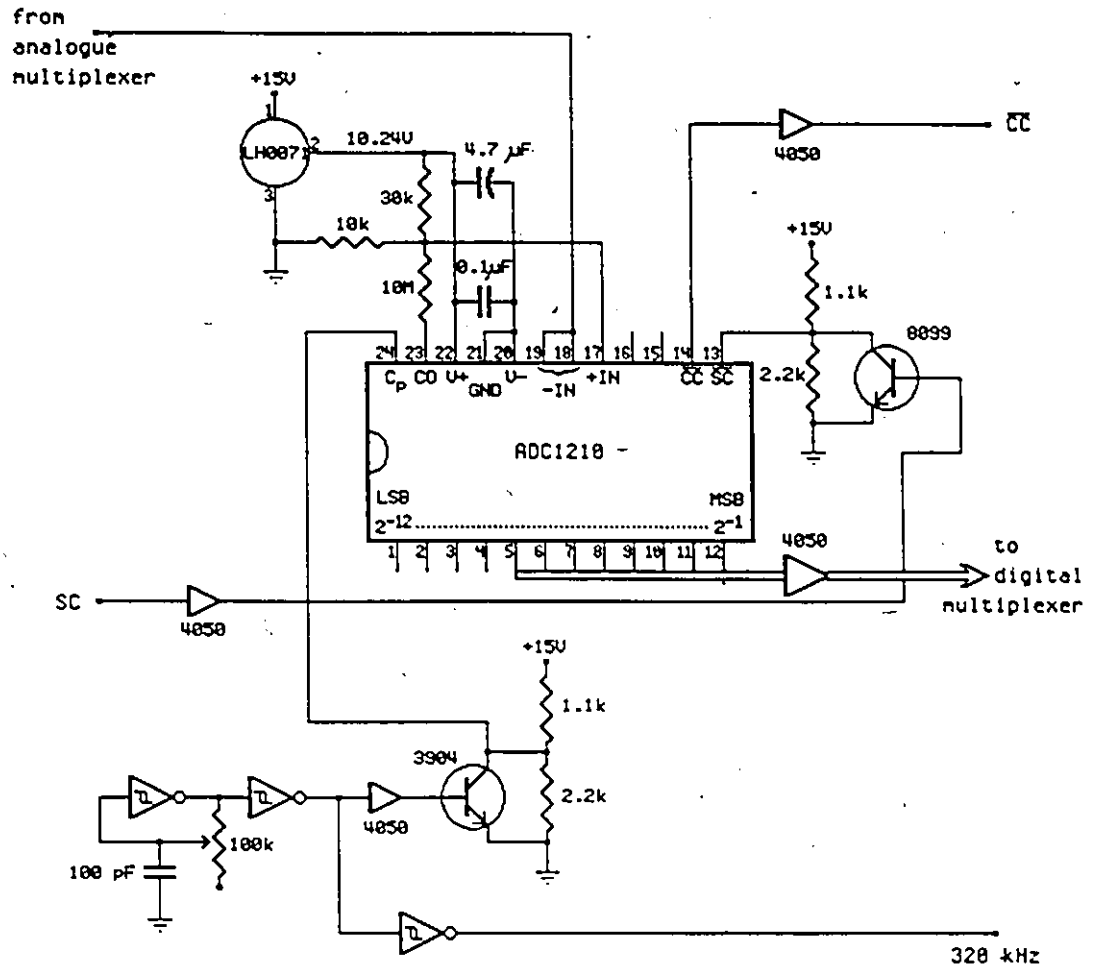


Figure 3.17 : Analogue-to-digital converter (ADC) and support circuitry. The ADC1210 was used to generate 8-BIT representations of the twitch force and whole muscle action potential signals.

from the magnetometer TXFR signal and the digital multiplexer signals Q0 and Q1. The second circuit took the CC (conversion complete) signal from the ADC and combined it with the Q1 signal to yield (CC+Q1) for use

in the talker handshake circuit of the computer interface. The capacitive coupling of one input of a NAND gate to its other input allowed the second input to rise momentarily with a low-to-high transition of the first input. The resistor quickly drained one side of the capacitor, leaving the first input in the high state and the second input in the low state. This was a simple way of generating a pulse (of length determined by the RC time constant) from a low-to-high transition.

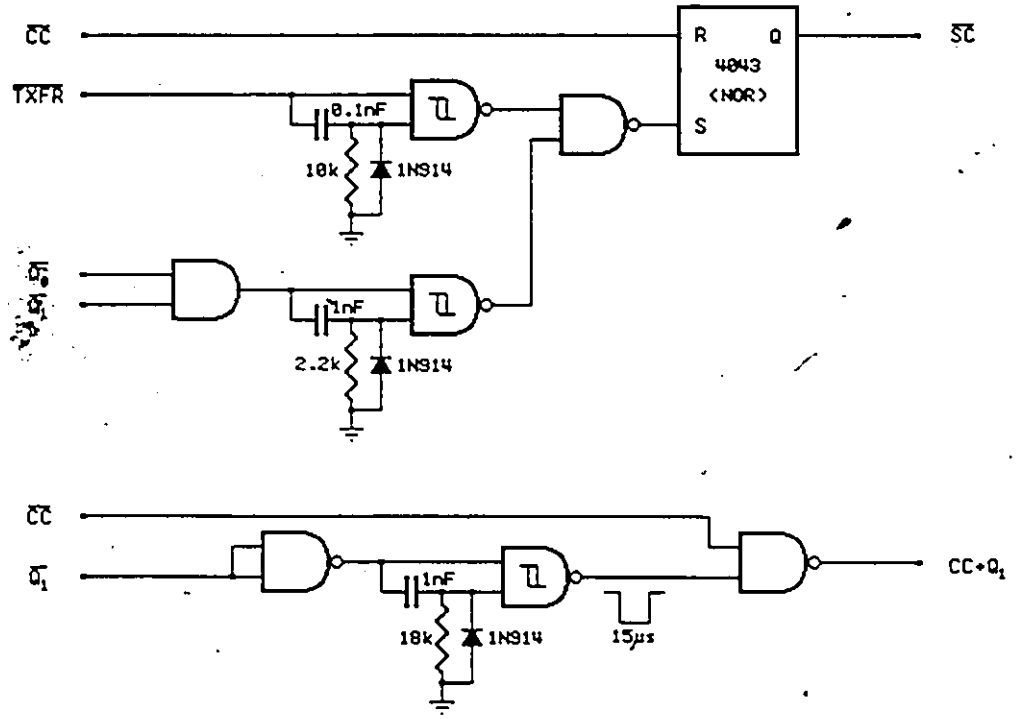


Figure 3.18 : ADC logic support circuits. These two circuits looked after generating the SC (Start Conversion) signal and passing the CC (Conversion Complete) signal to the talker handshake circuit. The RC-diode combinations allowed generation of pulses from low-to-high logic transitions.

3.4.5 Data Acquisition Control and Timing

Timing of the interfacing instrument control signals during one sample cycle (acquisition of 4 bytes) is summarized in Figure 3.19. The sample cycle was initiated by the magnetometer pulling the TXFR signal low. This caused the digital interface byte counter to reset and caused the SAMPLE signal to go low. The low state of SAMPLE caused the two S&H circuits to enter the hold mode. The rising edge of the TXFR pulse caused the SC signal to go low which started the successive approximation procedure of the ADC. Since Q0 was low, the analogue multiplexer directed the electric potential signal from the muscle to the ADC input. As soon as the ADC started its conversion process, it set the CC signal high.

The magnetometer pulled the DATA signal low when the first byte of its 4-byte sample was ready. But, since the ADC byte (representing the muscle electric potential) was to be substituted for the magnetometer byte, the DAV signal was not changed until the ADC signaled completion of the conversion by pulling CC low. The DAV signal was pulled low and the computer responded by pulling NRFD low. The computer read the data byte from the ADC since Q1 was low (which caused the digital multiplexer to select the ADC byte).

The computer let NDAC go high when it had read the data byte. This directly caused the DAV signal to return high and the DATASTB signal to go low. The magnetometer responded to a low DATASTB by letting DATA go high and prepared its next byte of data. The rise of NDAC also caused the digital interface byte count to increment and Q0 to go high. The computer responded to DAV going high by pulling NDAC low again.

Some time later, the computer showed that it was ready for another piece of data by letting NRFD go high again.

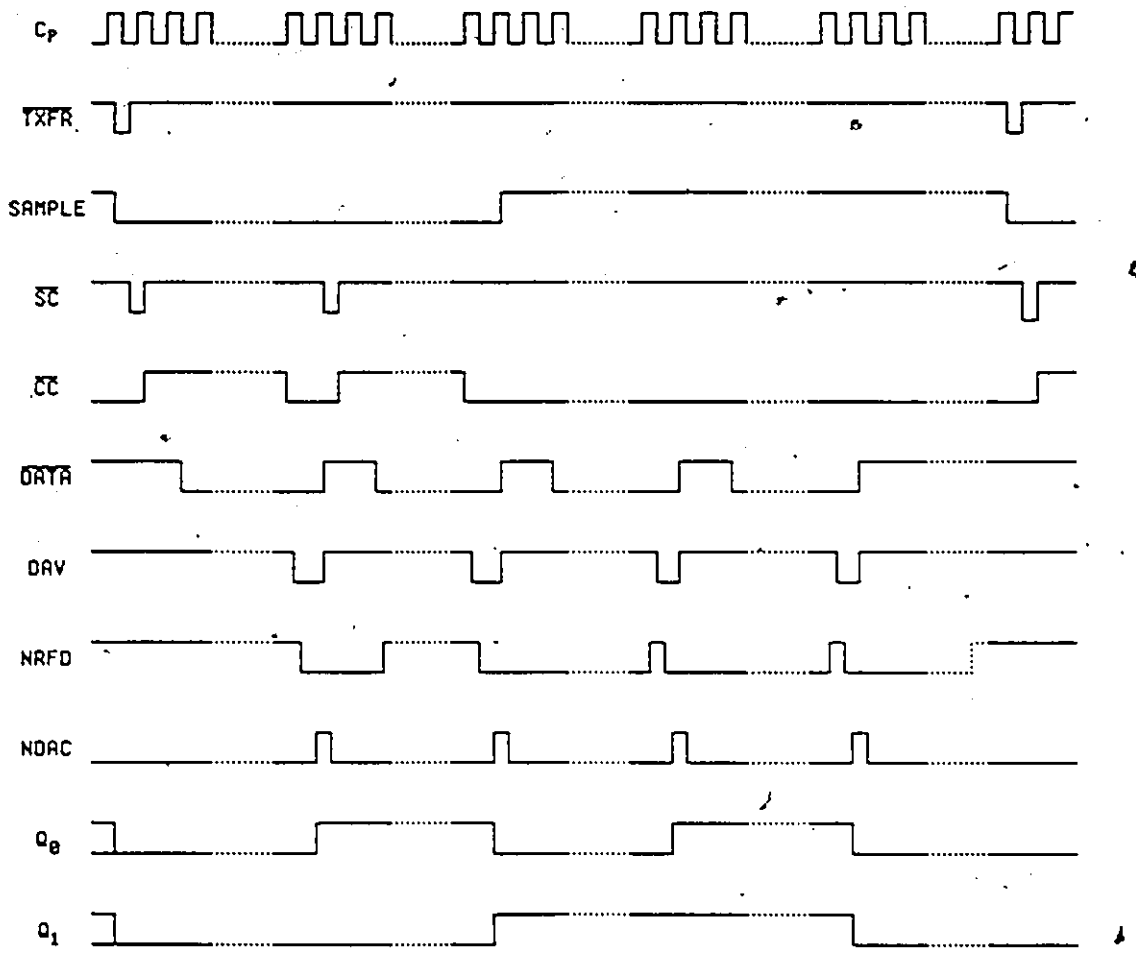


Figure 3.19 : Timing of data acquisition control signals. The clock signal required to drive the ADC successive approximation logic is shown at the top. The acquisition of each group of four data bytes was triggered by the TXFR signal from the magnetometer.

It was the rise in Q0 (in response to the NDAC pulse from the computer) that started the conversion of the second byte. This time, since Q0 was high, the analogue multiplexer selected the muscle force signal for the ADC input. When conversion was complete, this byte was made available to the computer in place of the magnetometer's second byte. As stated earlier, the magnetic signal from a muscle was expected to be so small that it would only occupy the least significant two bytes of the magnetometer's 4-byte representation.

During transfer of the last two bytes (of each group of four) the Q1 signal was high so that the digital multiplexer selected the magnetometer data instead of the ADC output. The ADC was not triggered to start a conversion (SC stayed high) and the S&H circuits were put in sample mode. The DATA signal from the magnetometer directly caused the DAV signal to go low when data was ready and once the computer indicated that it was ready for data by letting NRFD go high. The computer sent the NDAC pulse once the data had been read.

When the last cycle was completed, Q0 and Q1 were both low, the S&H circuits were in the sample mode and the ADC was waiting to start a conversion. As a result of Q0 and Q1 being low, the analogue multiplexer once again had the muscle electric potential connected to the ADC input and the digital multiplexer had the ADC output selected for the computer interface. The cycle did not start over again until the magnetometer pulled TXFR low again (122 microseconds after the previous TXFR pulse).

The magnetometer was fooled into thinking it was delivering data even when the computer was not actually listening. This was done by

returning the magnetometer's DATA handshaking signal on the DATASTB line with a short delay. This was done so that when the computer started collecting data, there would not be a startup delay as the magnetometer digital filter filled with data values. The computer initiated data collection by issuing a talk command to the interfacing instrument. The data acquisition actually started with the first TXFR pulse from the magnetometer after the computer had released the ATN line (let it go high). As stated earlier, the time of sampling of signals was controlled by the magnetometer through its regular TXFR pulses. The data acquisition continued until the computer data buffer was full. The computer then sent an "untalk" command on the IEEE-488 digital interface. The interfacing instrument responded by returning to listen mode on the IEEE-488 interface and started returning the magnetometer's DATA signals on the DATASTB line.

3.4.6 Sample Counter

It was expected that the signal-to-noise ratio of the magnetic signal from an active skeletal muscle would not be high. It was therefore decided that a noise record should be stored with every signal record. This would aid in signal characterization by allowing investigation of the noise power spectrum separate from the signal spectrum.

The computer data buffer was made twice as long as the desired length of the record containing the muscle signal. The computer initiated the data acquisition process, but the muscle was not stimulated to contract until the data buffer was half full. The first half of the data buffer thus contained the noise record and the second half of the

buffer contained the muscle response.

Stimulation of the muscle (when the buffer was exactly half full) was accomplished by means of the sample transfer counter shown in Figure 3.20. One set of data samples was collected each time the magnetometer generated a TXFR pulse. These pulses were counted with 4520 dual synchronous divide-by-16 counters. When the count matched the switch settings, a pulse was sent to the muscle stimulator. Counting, and thus muscle stimulation only occured while SEND was in the high state.

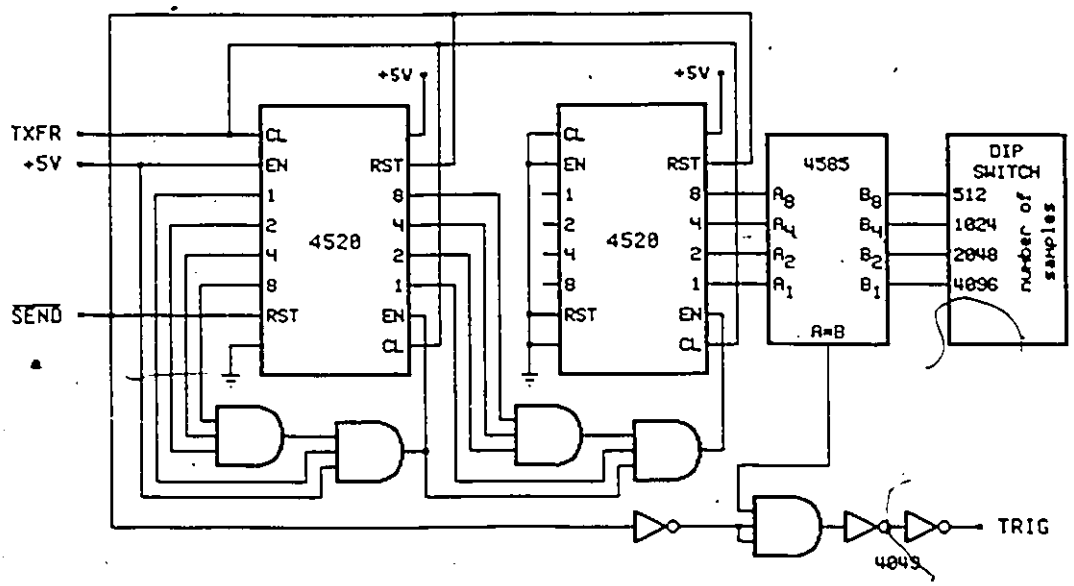


Figure 3.20 : Sample counter circuit. This circuit was designed to count the number samples collected by the computer and send a trigger signal to the muscle stimulator when the number of samples set by the DIP switch had been transferred.

For the experimental results discussed in this thesis, the computer buffer was set to hold 2048 samples of the muscle electric potential, force and magnetic field (a total of 8192 bytes). The sample counter was set at 1024 so that when the 1024th TXFR pulse was generated by the magnetometer, a trigger pulse was delivered to the muscle stimulator. The stimulus artifact in the magnetometer signal due to the muscle stimulator did not appear until the 1026th sample. The delay from when the magnetometer produced the 1024th TXFR pulse and when the stimulus artifact appeared in the magnetometer output was less than 0.24 ms (the samples were taken every 0.12 ms).

3.4.7 Ground Loops

In an analogue data acquisition system, ground loops can introduce noise into the signals. As for any conducting loop, a circuit composed of ground wires for several instruments and transducers can have voltages induced in them by changing magnetic fields from external sources such as electric motors and fluorescent light ballasts. Some of these wires will not remain at true ground potential and some measurements may have errors introduced to them. It is therefore important to make sure that the necessary ground wires are not inadvertently connected to form loops.

Several precautions were taken in the design and construction of instrumentation for this research. The ground wires were of as heavy a gauge as was practical. The shields of cables were grounded only at one end (usually at the interfacing instrument end). The muscle was a common point for two sets of electrodes. The electrodes for measuring the

electric potential of the muscle and the electrodes for stimulating the muscle to contract. The muscle provided a relatively low resistance bridge between the two circuits. To prevent creation of a loop, a muscle stimulator with an isolated output was used (discussed in section 5.2.5). The magnetometer and the force transducer circuitry did not pose a problem because these systems were not electrically connected to the muscle.

Ground loops within digital circuits were not considered to be a problem. Noise introduced by ground loops would be small compared with the digital voltage levels and would not cause digital errors.

CHAPTER 4

SOFTWARE DESIGN

The Hewlett-Packard Integral Personal Computer (9807) was used to perform experiment control, data acquisition and data analysis functions. This computer made use of the Motorola 68000 microprocessor with a clock frequency of 8 MHz. The operating system was a scaled-down version of System V UNIX, called HP-UX. The UNIX kernel was ROM-based and available upon boot-up. HP-UX differs from UNIX primarily in that it lacks multi-user support. Technical BASIC and C programming languages were available when the computer was purchased. The C language was chosen for software development because of the power and flexibility it offers. The UNIX operating system and the C programming language provide an excellent engineering software environment. The data acquisition and experiment control functions were performed by one collection of subroutines controlled by a main programme called SMDA (SQUID Magnetometer Data Acquisition). Data analysis was performed by a set of individual programmes that were run from shell procedures (UNIX command programmes).

4.1 EXPERIMENT CONTROL

4.1.1 The Problem

Preliminary investigations with excised muscle preparations showed that the muscle can deteriorate significantly over a period of 30 minutes. To minimize this deterioration, it was desired to perform each experiment as quickly as possible. However, as procedures are speeded up, it becomes more and more likely that something will be forgotten. It was decided that a good way of improving speed of execution while ensuring adherence to protocol is to use the computer to direct the experiment as well as collect the data.

One large programme, called SMDA (SQUID Magnetometer Data Acquisition), was developed to perform the experiment control and data acquisition functions. The experimental control functions were divided into three major areas: calibration; dissection and anatomical data entry; determination of independent variables for three experiments. The calibration routine issued instructions and provided auditory feedback for zeroing transducer DC offsets and determining system gains. The dissection and anatomical data entry routine also issued instructions as to the sequence of tasks to be performed and requested entry of data describing the specimen. Finally, instructions (including calculated settings for the apparatus) were issued for any one of three experiments. This computer control optimized speed while minimizing decision making and confusion during each experiment.

Presented in Figure 4.1 is a block diagram of the SMDA programme, showing its major components. A complete listing of the C

source code for this programme and all the subroutines it calls is to be found in Appendix A. The main programme is listed first, after which all the subroutines are listed alphabetically by name. Any of the 7 major function blocks could be entered from the SMDA main programme by selecting the corresponding special function key (labels for the special function keys were displayed along the bottom of the IPC display). Control returned to SMDA once a function was completed. This allowed the flexibility of executing functions in any order. In Figure 4.1, the special function key labels displayed by the SMDA main programme have been used to label (in upper-case letters) each of the major function boxes. The actual subroutine call names have been listed in lower-case letters. Besides controlling subroutine branching, the main programme displayed a start-up message describing programme execution and displayed a list of settings for the magnetometer controls.

Generally, an experiment was initiated by pressing the SET-UP function key. The "set_up" function handled the calibration procedures and then called the "muscle_dat" routine. The "muscle_dat" function requested anatomical information about the muscle and then aided in mounting the muscle and adjusting the apparatus. The "muscle_dat" routine could be called from the SMDA main programme at any time if a new muscle was to be used. One of the three experiments was selected once all of the set-up procedures were completed.

Many of the major subroutines called lesser subroutines. Some of these ("beep", "pause_continue", "fkey_lbl", "fkey_scan", "repeat_complete", "sequence") are simple enough that the comments included at the beginning of their listings will provide sufficient

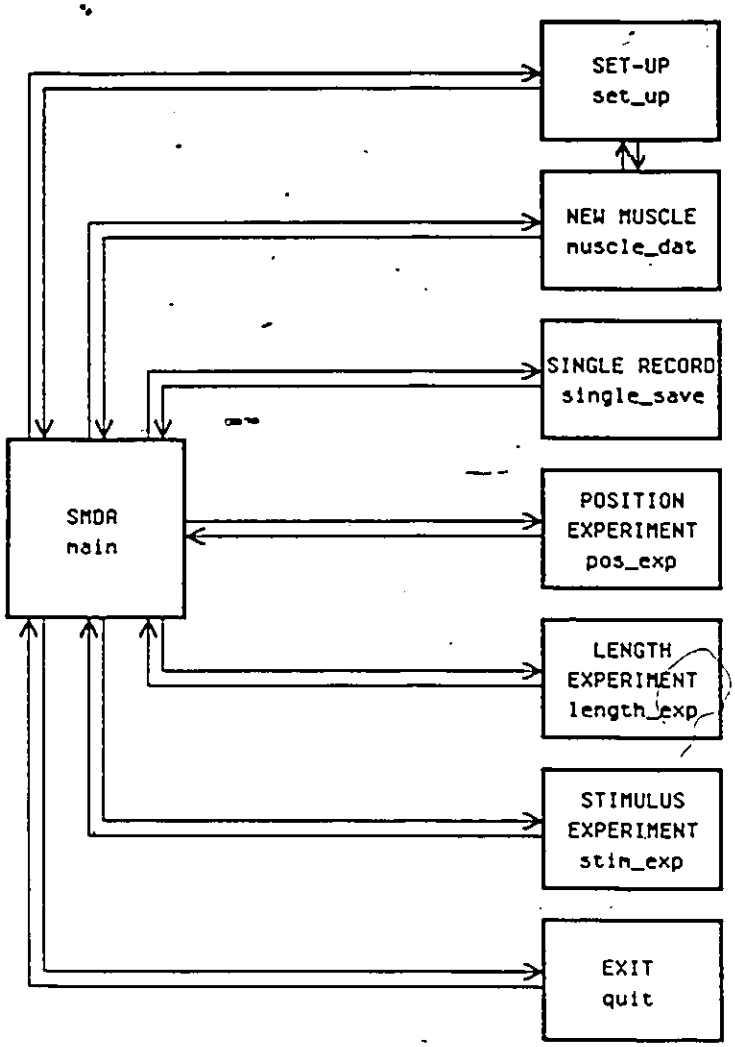


Figure 4.1 : Block diagram showing functional units of the SMDA programme. This was the main experiment control and data acquisition programme. The subroutine call-names are shown in lower-case letters while the function key labels are presented in upper-case letters.

explanation. Subroutines that will be discussed in this section and section 4.2 are: "acquire", "acquire1", "acquire3s2a", "save", "length_exp", "maxforce", "muscle_dat", "pos_exp", "set_up", "single_save", "quit", "stim_exp".

The "single_save" routine provided the facility for collecting and storing a set of data records without having to follow the fixed procedure of one of the three experiments. This function was particularly useful while testing improvements in the apparatus and experimental procedure. The data file names from this routine were partially derived from the time of day so that every name would be unique (ensuring that no data could be overwritten).

The "quit" function performed several tasks. First, it asked whether the operator was sure that the procedure should be terminated. It was important to have this opportunity to change one's mind because the set-up procedure required considerable time and would be a nuisance to repeat. Also, the "EXIT" function key may have been pressed by accident. If the operator chose not to terminate the process, control was returned to the SMDA main programme. If the desire to quit was confirmed, the data buffer was deallocated, the function key labels were blanked out and all open files were closed by making the exit(0) system call.

4.1.2 Calibration Procedures

As shown in Figure 4.2, the first task of the "set_up" subroutine was zeroing the force transducer DC offset. Because the force transducer bias circuitry and preamplifier were mounted close to the

transducer in the shielded room with the magnetometer, the computer could not be seen while adjusting the zero offset. This problem was solved by using the computer to provide auditory feedback related to the transducer output. This feedback consisted of pairs of beeps from the computer speaker. The first beep had a frequency of 200 Hz and acted as a reference. The second beep had a frequency that went up 10 Hz per integer of transducer output. When the force transducer yielded a zero output, the two beeps were of the same pitch. The operator would start this process by pressing the CONTINUE function key on the computer and enter the shielded room to adjust the offset trimpot while the pairs of beeps were generated. If needed, the process was repeated by pressing the REPEAT function key.

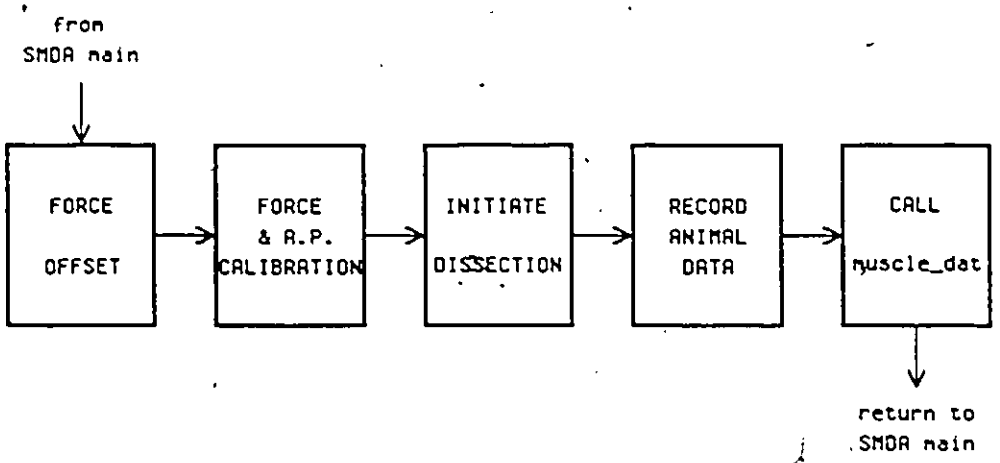


Figure 4.2 : Block diagram showing functions of the "set-up" subroutine.

Calibration of the force transducer system and muscle electric potential amplifiers was the next task. A square wave calibration signal was applied to the electric potential preamplifier inputs and a standard mass was applied to the force transducer. Twenty samples of these two analogue signals were taken by the computer. The maximum value of each set of samples was taken as a calibration integer. The procedure was repeated several times to check the repeatability of the calibration integers. The mass and square wave amplitude were then entered into the computer and calibration factors were calculated in units of micronewtons per integer and microvolts per integer for the muscle force transducer and the muscle electric potential amplifiers respectively.

For both the force transducer zeroing and the calibration procedure, the "acquire1" subroutine was used to acquire single samples of the three signals (force, electric & magnetic). This subroutine will be discussed in section 4.2. Also, the "pause_continue" and "repeat_complete" subroutines were used to control programme operation. The "repeat_complete" routine caused the value of the external variable "fkey" to be modified so that it could affect the test at the end of the do-while loop from which it was called.

4.1.3 Anatomical Data Entry and Mounting of the Muscle

The last part of the "set_up" routine dealt with the recording of information about the animal whose muscle was to be studied (species, sex, size). An instruction to start the dissection was then issued by the computer and the time of death of the animal was automatically recorded. This time was recorded so that each data record could have

stored with it the time that had elapsed since the muscle lost its circulation. The last action of this routine was to call the "muscle_dat" subroutine.

As shown in Figure 4.3, the first task of the "muscle_dat" subroutine was to collect information about the excised muscle (length, mass and from which leg of the animal). This information, along with the information about the animal (entered in the "set_up" routine), was stored with each data record. With this method, no measurement could become "anonymous" by incomplete recording of experimental conditions or misfiling of this information. The information for completely specifying the experimental conditions for each measurement were guaranteed to be stored with it.

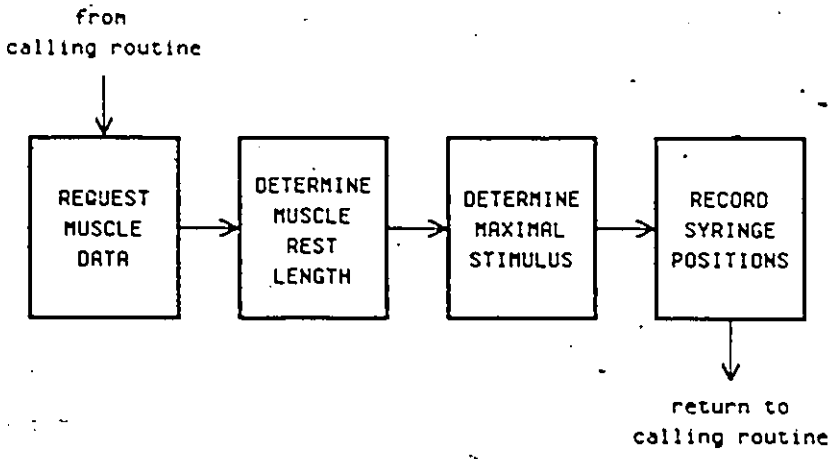


Figure 4.3: Block diagram showing functions of the "muscle_dat" subroutine.

Once the excised muscle was mounted in its supporting apparatus (discussed in section 5.2), its rest length had to be determined. This was done by stretching the muscle to its maximum length for which the force transducer still registered no tension. The auditory feedback method (used to zero the force transducer DC offset) was once again used since the procedure required the operator to be in the shielded room and the computer was not visible. The first beep of each pair provided a reference pitch and the pitch of the second beep was higher by 10 Hz times the force integer. The greatest muscle length for which the two beeps had the same pitch was taken as the rest length.

The maximal stimulus for the muscle preparation was next to be determined. As the nerve of a muscle is stimulated with voltage pulses of larger amplitude, the number of activated motor units increases. This causes a rise in the amplitude of the twitch force. The maximal stimulus is the amplitude of the stimulus, above which, no increase in the twitch force is observed. This was determined by stimulating the muscle to twitch (discussed in section 5.2) and collecting complete data records. The "maxforce" subroutine was used to determine the maximum force in the recorded twitch response. The stimulus amplitude was increased with each stimulus until there was no longer an increase. The twitch force amplitude corresponding to the maximal stimulus was stored in the external variable, "force100". This force level was used in the stimulus experiment for determining desired stimulus levels. Also stored were the maximal stimulus pulse amplitude and duration.

The final task of "muscle_dat" was to record the settings of the positioning device when the muscle was centered under the magnetometer.

During each of the experiments, the computer calculated position settings based on these initial values. Programme control returned to "set-up" or the SMDA main programme depending on which had made the call to "muscl_dat".

4.1.4 Three Experiments

Three experiments were of major importance in this research. A subroutine was written to control each of these experiments. The procedures and experimental results are discussed in chapters 5 and 6 respectively. The subroutines were named after the independent variable that was caused to change in that experiment: "pos_exp", "length_exp" and "stim_exp". The muscle position experiment, "pos_exp", investigated the spatial dependence of the muscle's magnetic signal by measuring the magnetic signal with the muscle in different positions with respect to the magnetometer sensing coil. The muscle length experiment, "length_exp", studied changes in the magnetic signal with muscle length. Changes in the magnetic signal due to changes in the stimulus amplitude to the muscle's nerve were investigated in the stimulus experiment, "stim_exp".

Each of these experiment control subroutines performed the same set of tasks. First, settings for the independent variables that did not vary throughout the experiment were displayed for the operator to set. During each experiment, a sequence of values for the independent variables that were to be changed were displayed. Data was acquired and a set of data records was saved for each of the independent variable values. Responsibility of the operator was reduced to careful and

time-efficient setting of the variables as directed by the computer. Through control subroutine, "sequence", the operator also had the opportunity to repeat any set of measurements if a complication (such as increased magnetic noise) occurred.

The "stim_exp" subroutine was slightly more complicated than the other two experiment control routines. In this experiment it was desired to adjust the stimulus pulse amplitude so that the resulting twitch force amplitude was a chosen fraction of the maximal twitch force amplitude. The relationship between the stimulus pulse amplitude and the resulting twitch force amplitude was highly nonlinear. A trial-and-error method was therefore used to find the desired twitch force amplitude. The "acquire" subroutine was used to collect single data records and the "maxforce" subroutine was used to find the twitch force amplitude. These routines were run under the control of "repeat_complete" until the desired twitch force amplitude was obtained. Measurements for a complete set of data records were then made by "acquire3s2a".

4.2 DATA ACQUISITION SOFTWARE

4.2.1 Requirements

The three muscle signals to be measured by the computer were electric potential, twitch force and magnetic flux. As discussed in section 3.4, each 4-byte sample from the interfacing circuit consisted of one byte for each of the electric and force signals and two bytes of

magnetic information. To be sure of not band-limiting the magnetic signal, it was decided that the signal would be low-pass filtered with a 3 dB cutoff freqency of 2048 Hz. To be sure that aliasing effects would be negligibly small, the sampling frequency was chosen to be four times the filter cutoff frequency (8192 samples per second). To be sure that the twitch force was not temporally truncated, the duration of data collection would have to be at least 100 ms. Since a noise record was to be collected just prior to stimulation of the muscle, the total duration of the measurement was chosen to be 250 ms. Thus, a total of 8192 bytes were to be collected at a rate of 32768 bytes per second (8192 samples per second for each signal). Each of the three signals was represented by a record of 2048 data points.

From the above discussion it is clear that requirements for the data acquisition software includes the facility to collect bursts of 8192 bytes of data at a rate of 32768 bytes per second. The IEEE-488 digital interface standard was designed to handle transfer rates up to 1 Mb/s. Any rate limitations would be due to software design. The software also had to be able to manipulate data vectors of up to 8192 bytes. Finally, the data acquisition software had to provide a means of storing the data in a compact, but accessible form. Besides the data vectors, many scalar variables describing the experimental conditions were to be stored.

4.2.2 Implementation

The data buffer was declared an external character vector at the beginning of the SMDA main programme. The 8192 bytes required for this buffer plus 64 bytes for storage of experimental-condition variables were allocated in main memory with the C subroutine, "calloc". The subroutine called "acquire" was used to read the data from the IEEE-488 digital interface into this buffer. The "acquirel" subroutine was similar to "acquire" except that only 4 bytes of data were collected and the muscle was not stimulated. The single sample of each signal was useful in calibration procedures.

The first task of the "acquire" subroutine was to open the device driver file for the IEEE-488 digital interface. The digital interface was also called the HP-IB (Hewlett-Packard Interface Bus). The following list of interface commands was sent through the interface to prepare for the transfer of data from the interfacing instrument to the computer:

UNTALK -> any device talking on the interface is deactivated

UNLISTEN -> listening devices release control of handshake lines

TALK 25 -> the interfacing instrument is addressed to talk

IPC LISTEN -> the computer addresses itself to be the listener.

Just before reading the data, the digital interface was put in "burst" mode so that a high speed handshake algorithm was used.

When the computer started to execute the "read" subroutine, it released the ATN (Attention) control line of the interface. The interfacing circuit immediately started sending data at a pace determined by the magnetometer TXFR signal. When enough samples had been transferred

to half fill the data buffer, the interfacing instrument sent a trigger signal to the muscle stimulator. Thus, the first half of the data buffer was a noise record and the second half contained the muscle response.

When the required number of bytes had been read, the "read" subroutine terminated (and returned with the number of bytes read), the "burst" mode was disabled and the computer sent the UNTALK command to deactivate the interfacing instrument. Before returning to the calling routine, the "acquire" subroutine checked that the required number of bytes had been read and closed the interface device driver file.

The "hpib_bus_status", "hpib_send_cmnd" and "io_burst" subroutines were from a Device Input/output Library (DIL) purchased from Hewlett-Packard. Although these routines were generally effective, they did not handle (disable) interrupts from the multitasking operating system. After months of searching for the cause of missed data in the interfacing instrument hardware, it was discovered (by use of a logic analyser on the interface control lines) that the computer was shutting down the data transfer for periods of 15 ms, 5 times per second. A C language system call, "rtprio", was discovered in the Hewlett-Packard documentation that should have corrected this problem. Its function was to change the priority of a process and could be used to prevent the interruption of a critical process by the time-sharing operating system. Despite its description in the documentation, "rtprio" was not available in either of the first two releases of the C compiler.

A software patch was obtained from HP-Corvallis to disable the time-sharing interrupts from the multitasking operating system. The two

lines of code just before and just after the "read" call in "acquire" write hexadecimal numbers to memory location E40023. This prevented the operating system from receiving signals from the real-time clock and consequently it did not know when to interrupt the current process. Though crude, this software patch was effective.

The "save" subroutine performed the task of creating a floppy disc file and writing the collected data to it. The data buffer was written directly onto the disc without modification. The data in the buffer was already in a conveniently compact form. As already described, the buffer contained a total of 8192 bytes, in groups of four. The first byte of each group represented the electric potential of the muscle, the second was the twitch force and the last two bytes were from the magnetometer. The signals stored in the first half of the buffer were of the background noise and the second half contained the muscle response signals.

Before writing the data buffer to the disc file, 22 integer variables describing the experimental conditions were appended to it. Depending on the expected range of each variable, it was given one or two bytes of space in the disc file. The most significant byte of a two byte representation was written to the disc first. By storing a complete description of the experimental conditions with each data record, it was not possible to confuse the origin of a data record in later processing. The memory expense was minimal considering the benefit (64 bytes of the 8256 byte disc file was allocated for these variables).

Several data files were saved for each set of experimental conditions. Originally, one file was to contain an average of 9 data

records, a second file was to contain an average of 25 data records and three files were to contain single data records. The averaging of records was performed to improve the signal to noise ratio. The single records were saved for study of record-to-record variation. Control of the data collection and storage of these data files was provided by the "acquire3s2a" subroutine. This routine called "acquire" to perform the data acquisition and "save" to store the data on a floppy disc. The signal averaging was performed within the "acquire3s2a" routine.

Experience showed that data acquisition for the 5 files for each set of experimental conditions caused the muscle to deteriorate before a whole experiment could be completed. Each set of conditions required 28 stimulations of the muscle to fill the data files. As shown in the listing of Appendix A, "acquire3s2a" was modified to collect data for only three files per set of conditions: two files, each containing a single data record; one file containing an average of 9 records. This reduced the load on the muscle preparation to only 11 stimulations per set of experimental conditions.

Each of the data records stored on disc had to be given a unique name. The names consisted of 6 parts. An example is "F24RSC05S2". The first three characters specified the frog number: the letter F followed by a two digit number. The frogs were numbered sequentially as they were used. The frog number was followed by a letter specifying which leg of the animal had been used: R for the right and L for the left. Next came a letter that determined which experiment had been performed on the leg: L for the "length" experiment, P for the "position" experiment and S for the "stimulus" experiment. The "condition number"

consisted of the letter C followed by a two digit number. Every set of experimental conditions within an experiment was assigned a unique "condition" number. Assignment of the first 8 characters of the data file name occurred within each of the three experiment subroutines of SMDA: "length_exp", "pos_exp" and "stim_exp". The last two characters of the data file name were added by the "acquire3s2a" subroutine. The first of these two characters specified whether the file contained a single data record (S) or the average of 9 successive data records (A). The final character specified whether it was the first or second file of the type specified by the first 9 characters of the file name.

4.2.3 The DATA_CHECK Programme

As described in the previous sections of this chapter, the SMDA programme performed the required experiment control and data acquisition tasks. It did not, however, provide any facility for checking the data as it was collected. It was found that during an experiment there was occasionally reason to doubt that everything was operating as expected. In these cases it was necessary to see the data to dispel or confirm the suspicion. The DATA_CHECK programme was developed to fulfill this need.

The DATA_CHECK programme provided facilities for printing and plotting data already stored on a floppy disc. It also allowed the collection of single data records so that changes in the experimental apparatus could be tested without using up the predefined data files of SMDA. The multitasking operating system was used to run DATA_CHECK simultaneously with SMDA. A complete listing of DATA_CHECK is presented in Appendix B.

A block diagram describing the functional units of DATA_CHECK is shown in Figure 4.4. The first block provided the ability to collect single data records to check adjustments in the muscle environment without affecting operation of the SMDA programme. The "acquire" subroutine, presented in the previous section, was slightly modified for use in DATA_CHECK. The character data buffer was allocated within "acquire" rather than in the "main" calling routine. After reading data into the buffer, it was split into three data vectors. It is these three data vectors that were used for display or plotting by the other DATA_CHECK functions.

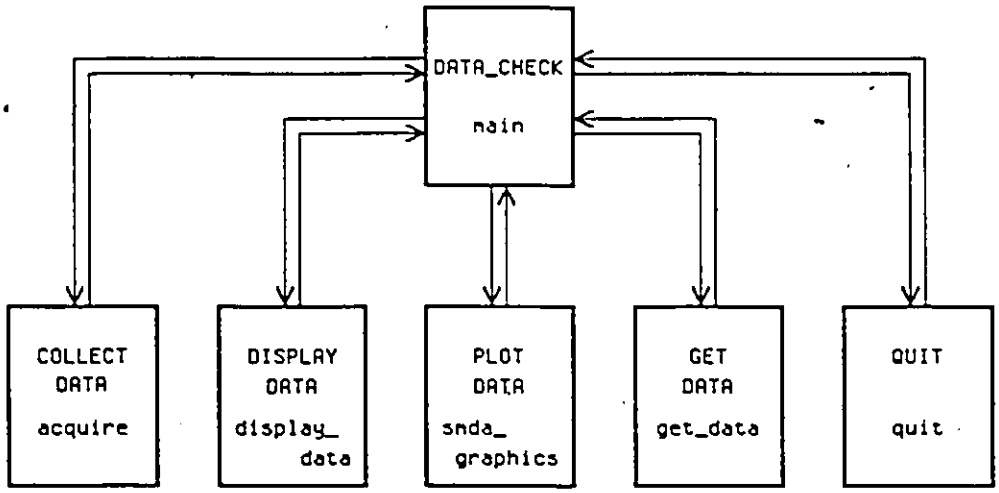


Figure 4.4 : Functional units of programme DATA_CHECK. This programme was run concurrently with SMDA and was used to check the equipment and muscle preparation by collecting and displaying data. The subroutine call-names are shown in lower-case letters while the function key labels are presented in upper-case letters.

To access data already saved on disc by the SMDA programme, the "get_data" function was used. Once the disc data file was opened and read into the character data buffer, it was split into the three data vectors and the collection of variables describing the experimental conditions. Pointers to the three data vectors were declared as external variables in the DATA_CHECK main programme.

Once the data vectors were defined, they could be printed on the computer display with the "display_data" subroutine. The first part of the display was devoted to printing the set of variables describing the experimental conditions. Four columns were then printed. The left most column numbered the rows from 0 to 2047. Each of the other three columns contained integer representations of the three signals: EMG, the muscle electric potential; FORCE, the muscle twitch force; MMG the magnetic flux signal. Printing to the display was paused each time a new screen was filled. Printing was resumed when the CONTINUE function key was pressed.

The plotting function was the most complicated part of the DATA_CHECK programme. The "smda_graphics" subroutine looked after the selection of which data vector was to be plotted. The FFT (Fast Fourier Transform) of any one of the data vectors could also be plotted. Once the data vector to be plotted had been selected, the "plot_prep" subroutine was called, which in turn called the "plot" subroutine. It is the "plot" subroutine that actually wrote graphics commands and data to the plotter emulator window. The graphics window was created by the "open_graph" subroutine, called early in the DATA_CHECK main programme.

The "plot" subroutine was designed to plot a y-direction (vertical) data vector against an x-direction (horizontal) data vector. It automatically chose the x and y scales so that the plot would fill the screen. It also automatically placed the graph axes located tick marks and numbered them. Straight lines were drawn between the data point locations. No character was drawn at the data point locations.

The "plot_prep" subroutine provided an interactive shell for the "plot" routine. Once a graph had been generated, "plot_prep" allowed the user to examine any part of the graph more closely. This was done by changing the limits of the plotting area and redrawing the plot. The function key menu provided the opportunity to replot the top half, middle, bottom half, left half, horizontal middle or right half of the graph. Repeated use of these functions allowed the user to zoom in on a feature and magnify it until it was clearly visible.

The "cfft" subroutine provided the facility to investigate the frequency content of any of the data signals. If selected, the "smda_graphic" subroutine would call "cfft" before calling "prep_plot". The "cfft" routine calculated the complex Fourier coefficients of a complex data vector. The number of data values in the data vectors was required to be an integral power of 2. This FFT routine was based on a published FORTRAN routine (William, Flannery, Teukosky & Vetterling, 1986). The algorithm involved the usual BIT reversal and Danielson-Lanczos lemma. It was the magnitude of the complex Fourier coefficients that was plotted.

4.3 DATA RETRIEVAL AND ANALYSIS SOFTWARE

Many small programmes were written in C to perform individual data analysis functions. They were executed interactively or in predetermined sequences by shell programmes (UNIX command files). These functions included unpacking the data from the ASCII vectors on disc, plotting the MMG, EMG and twitch force data, performing fast Fourier transforms and performing linear least-squares fits of variables. Source code listings for some of these programmes are presented in Appendix C.

4.3.1 Data Retrieval

Initial data analysis involved visual inspection of data vector listings and plots. To retrieve the data vectors from their packed disc files and convert them into an accessible form for the data analysis programmes, the "efdtr" (Erindale Frog Data To RAM) programme was developed. This programme read the packed character data of a specified disc data file into a character buffer. It then unpacked the data into three data vectors; "emg", "force" and "mmg". The set of variables that described the experimental conditions (stored at the end of each data file) was written into a buffer called "parameters". The unpacked data were in the form of 2-byte integers and were written to RAM files with the same names as the data vectors. For the "emg" and "force" data files, these were the integers generated by the ADC in the interfacing instrument. For the "mmg" data file, the stored integers were as received from the magnetometer interface. These integers could be

converted to scaled numbers with units by multiplying them with calibration factors stored in "parameters".

A listing of any of the integer data files could be generated with the "btascii" (Binary To ASCII) programme. The specified RAM data file was read by "btascii" into an integer buffer. Printing of the integers started and ended at positions in this buffer specified in the "btascii" command. If these positions were not specified, the whole 2048-point data vector was printed. The data values were written across the page, with 8 values per row. The left most column indicated the position in the data vector of the first data value in that row. These data vector listings were used to calculate signal amplitudes, delays and durations.

4.3.2 Plotting Software

The IPC supported graphics windows that emulated Hewlett-Packard plotters. The Hewlett-Packard graphics language (HP-GL) was used to generate data plots in these windows and with an HP-7470A plotter. The subroutine "cgw" (Create Graphics Window) was written to create a graphics window with the desired characteristics. The graphics window was of the plotter emulator type, filled the entire computer display and did not destroy itself when its device driver file was closed. All of the plotting programmes were originally written to draw plots in the graphics window. The plotting programmes were later modified to drive the HP-7470A plotter so that high quality hard copies could be obtained. The HP-UX "print_screen" command provided a facility for dumping the BIT-mapped graphics window to the internal printer for draft quality

hard copies. The versions of plotting programmes listed in Appendix C are all for driving the HP-7470A plotter.

The general purpose plotting subroutine was called "plot". It was called by several programmes that generated different types of graphs. The "plot" subroutine was designed to plot a data vector, automatically choosing and labelling scales so that the plotted data would fill the display. The data were written into an x-direction (horizontal) and a y-direction (vertical) data vector by the calling programme. The number of data points to be plotted and pointers to the two data vectors were passed to the "plot" subroutine. Also passed to the subroutine were pointers to 32-character strings containing the title, x-axis label and y-axis label. Finally, an array of four values for specifying the plotting limits could be specified. If zero values were specified, automatic scaling would occur. Placement of the title and axis headings, drawing of the axes, placement and numbering of axis tick marks were all built in to the "plot" subroutine. A scale factor (integral power of 10) was printed for either axis if required.

Of the many programmes that called the "plot" subroutine, "efdhplt", "asciihpltreg" and "ffthplt" have been listed in Appendix C. The "efdhplt" (Erindale Frog Data Hard Plot) programme was used to plot any of the three data vectors stored in RAM by "efdtr". The "efdhplt" command allowed specification of the portion of the data vector to be plotted, the plot axis scales, and where on the plotter paper the graph was to appear. An appropriate title and axis labels were automatically generated for each type of data vector. Also, calibration factors were read from the "parameters" RAM file so that the plots could be scaled

with appropriate units.

The "asciihpltreg" (ASCII Hard Plot with Regression) programme was written to plot an ASCII file of data point coordinate pairs and fit a straight line to them. The ASCII data files consisted of the graph title, x-axis label, y-axis label and number of data points listed on the first four lines. The data point x-y coodinate pairs were then listed, each on a separate line. The form of "asciihpltreg" was very similar to "efdhplt". Differences exist in the way the data files were read because of their different formats. Also, "asciihpltreg" did not create graph titles and axis labels, but read them from the data file. To provide the linear regression least squares fit, the "plot" subroutine had to be modified. Early in the "plot" subroutine, the slope, y-intercept and coefficient of determination were calculated for the best fit straight line. The modified "plot" subroutine has been listed as part of the "asciihpltreg" programme in Appendix C. Programmes similar to "asciihpltreg", but without the linear regression calculations, were used to plot ASCII data files with different formats (they were not included in Appendix C).

4.3.3 Muscle Fatigue Corrections

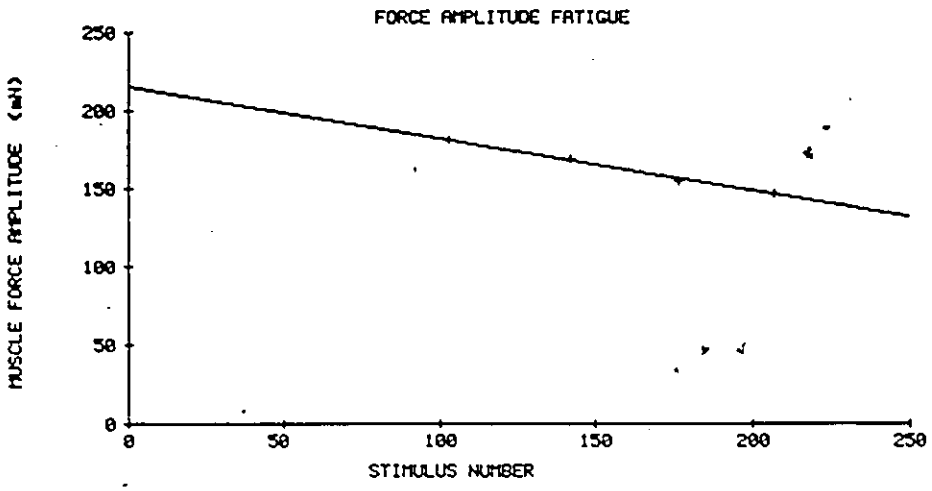
Several of the dependent variables derived from the muscle signals were found to change predictably with each stimulation of the muscle. This change was attributed to deterioration of the muscle preparation. Without correcting for these muscle fatigue effects, data analysis would lead to erroneous results. Muscle fatigue corrections depended on repeating a standard measurement several times throughout

each experiment. For variables that displayed a fatigue effect, the values from the repeated measurement were plotted as a function of the number of stimulations the muscle had received. The "asciihpltreg" programme was used to determine the slope of a best fit straight line. This slope was called the "fatigue slope". The fatigue slope was used to calculate a correction for each value of the fatigue dependent variable. The corrected values were calculated to represent the value that would have been obtained from the first stimulation of the muscle.

An example of a dependent variable that displayed a fatigue effect is the amplitude of the muscle twitch force. Figure 4.5 shows the "asciihpltreg" plot of four muscle twitch amplitudes for exactly the same experimental conditions except that the muscle had been stimulated 30 to 38 times between each of the data points. The fatigue slope was calculated to be 0.33 mN per stimulation and the y-intercept was 216 mN (milliNewton). The equation in Figure 4.5 was used to calculate the fatigue-corrected values. The table shows the actual values that were plotted and the corrected values that were calculated. Notice how much the mean and standard deviation changed as a result of the fatigue correction procedure.

4.3.4 Power Spectra

The frequency content of the data vectors was investigated with the aid of the "cfft" subroutine. This FFT routine was based on the usual BIT-reversal implementation of the Danielson-Lanczos lemma and provides the complex Fourier transform of a complex data vector. It was translated from a published FORTRAN subroutine (Press, Flannery,



$$\text{CORRECTED FORCE} = \frac{(\text{Y-INTERCEPT} \times \text{FORCE AMPLITUDE})}{(\text{Y-INTERCEPT}) + (\text{FATIGUE SLOPE} \times \text{STIMULUS NUMBER})}$$

WHERE Y-INTERCEPT = 216 mN
FATIGUE SLOPE = 0.33 mN/STIMULUS

STIMULUS NUMBER	103	142	177	207	MEAN	SD
FORCE AMPLITUDE (mN)	181	169	155	147	163	15
CORRECTED FORCE (mN)	215	217	214	217	216	1.5

Figure 4.5 : Muscle fatigue correction calculations. This example shows how the amplitude of the twitch force was corrected for the muscle fatigue effect. The corrected data represent what would have been obtained from the very first twitch of the muscle preparation.

Teukolsky & Vetterling, 1986) into the C language listing presented in Appendix C. The only restriction is that the number of data points be an integral power of 2. With no modifications to optimize performance

on the IPC (such as the use of register type variables), 512-point FFT's took approximately 13 seconds and 1024-point FFT's required approximately 27 seconds.

The "cfft" subroutine was called by the "ffthplt" programme. The purpose of this programme was to plot the power spectrum of a specified data vector. The "ffthplt" command allowed specification of the plotting scales, size and position of the plot on the paper and what portion of the data vector was to be transformed. The specified RAM data file was read into a 2-byte integer data buffer. The specified portion of the buffer was then copied to the FFT data buffer. During this copy procedure, the appropriate scale factor, obtained from the "parameters" RAM file, was used to scale the integer data. The "cfft" subroutine was called and the complex Fourier coefficients were returned in the FFT data buffer. The power spectrum was calculated from the squared magnitude of the complex coefficients. A factor of 4 was included to account for the contribution from negative frequency coefficients.

An example of a power spectrum from "ffthplt" is shown in Figure 4.6. This is the power spectrum of the magnetic signal from a muscle twitch. The many small peaks were found to change greatly from one data record to the next. The wide base upon which they sit was found to be repeatable. The small peaks were due to the constantly changing noise characteristics and the wide base was due to the repeatable muscle signal.

It was desired to reduce the amplitude of the many small peaks so that the muscle signal characteristics could be more easily

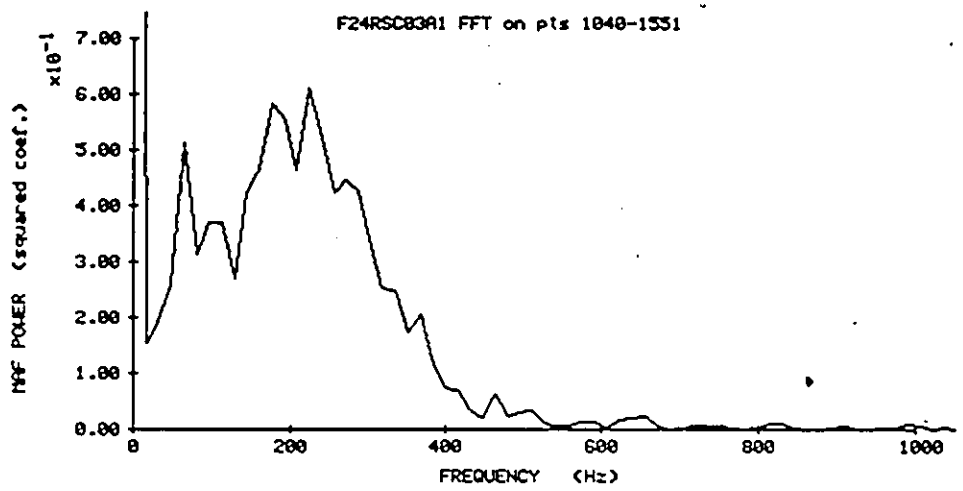


Figure 4.6 : A typical magnetic power spectrum derived from the Fourier transform of 512 data points from the "mmg" data vector (including the MAF from the muscle).

determined. To remove these peaks, a windowing procedure was used. The muscle response was short in comparison with the length of the transformed data vector. The muscle's magnetic response spanned only 35 of the 512 data points in the FFT of Figure 4.6. Thus, a lot of "extra" noise was included in the transform. This length of transform was necessary to obtain the desired resolution in the power spectrum. By multiplying the magnetic data vector by a square window, wider (in time) than the muscle response, none of the signal from the muscle was lost while a great deal of noise was excluded. The resulting spectrum is shown in Figure 4.7. The noise peaks have disappeared and only the base (due to the muscle signal) remains.

The "mmgmdfy" programme was used to perform the windowing operation on selected data vectors. To aid in the selection of the starting

and ending points of the window, the "mmgmdfy" programme displayed the data values from a specified section of the data vector. All the data values from the beginning of the data vector to the start of the window and all the values from the end of the window to the end of the vector were set to zero. The only non-zero data values were within the duration of the selected window. Note that the timing of the muscle response (position and duration in the data vector) was not altered in any way. Finally, the mean value of the first half of the record was subtracted from the windowed data to remove the arbitrary DC offset of the magnetometer. This modified data vector was then stored on disc with an "m" character appended to its name to indicate that it was in a modified form.

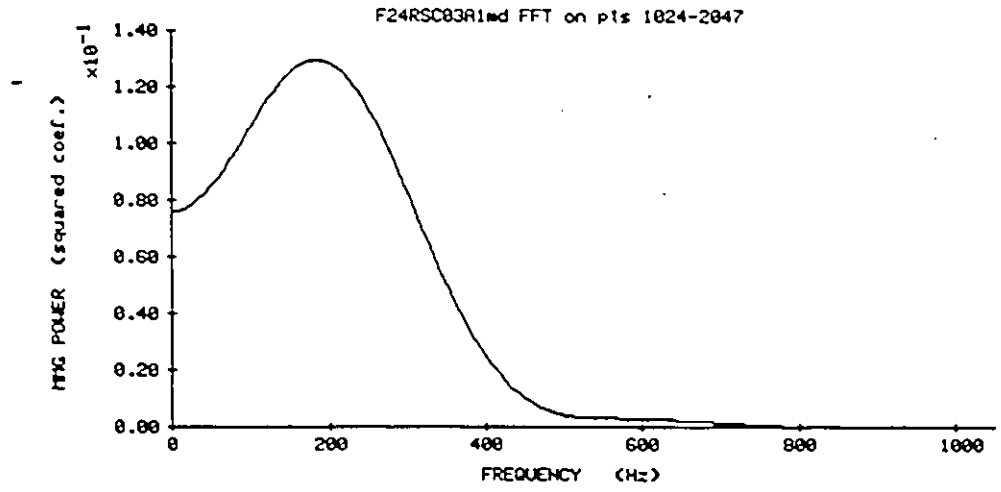


Figure 4.7 : Power spectrum derived from the Fourier transform of 1024 data points from the modified magnetic data vector. A square window was used to temporally isolate the MAF from noise in the rest of the data vector.

4.3.5 Experiment Analysis

Each experiment (described in section 5.3) generated a collection of 21 to 42 data files. To aid in analysis of these groups of files, shell command programmes were written. These programmes made use of the many data analysis programmes described above (and listed in Appendix C) to print and plot the data vectors.

An example of a shell command programme is "posexan". It is listed at the end of Appendix C. The first task was to copy into RAM all the data analysis programmes that were to be required. This allowed them to run faster since they did not have to be loaded from disc every time they were called. The "cgw" programme was run to create a graphics window for plotting. The directory containing the data files for the "position" experiment was selected and a loop was set up to process every "A1" type file in the group. The processing involved plotting and printing a section of the magnetic data vector, plotting the twitch force data vector and plotting and printing the power spectra of two sections of the magnetic data vector.

A second example of a shell command programme is "reptest". It is very similar to "posexan". The major difference is in the loop control mechanism that selected which files were to be processed.

CHAPTER 5

IN VITRO MEASUREMENTS OF THE EVOKED MAGNETIC ACTION FLUX

5.1 EXPERIMENTAL DESIGN

5.1.1 Objectives

The purpose of this research was to investigate the potential of the MMG as a research and diagnostic tool for neuromuscular diseases. The investigation could have taken many forms. A single human muscle could have been studied under voluntary control in healthy and diseased subjects. A survey of MMGs from a variety of muscles could have been taken. The MMG of a muscle group could have been studied while the subject was required to perform a diverse set of tasks.

There are two problems with these forms of investigation. First, the physiological variables on which the MMG depends have not yet been identified. Second, since the muscle is under voluntary control, many of these varibles cannot be determined and many of them may change simultaneously. When more than one variable is changing, the effect of individual variables becomes very difficult to determine.

In the early stages of investigating a phenomenon it is important to control as many experimental variables as possible. Ideally, all of the independent variables are under the control of the researcher. The effect of each variable can be studied by controlling its change while holding all other variables constant.

Investigation of the MMG from a muscle under controlled physiological conditions has not yet been reported in the literature. It was decided that this is where the present research should start. The physiological variables of the muscle under study were to be controlled as completely as possible. Complete control of a muscle in vivo is very difficult and is usually unpleasant for the subject. The measurements were therefore to be made from excised muscles. The muscle was to be indirectly stimulated through its nerve by single isolated pulses. The avoidance of the asynchronous activity of muscle fibers under voluntary control was a significant and necessary simplification.

The independent physiological variables were: stimulus voltage amplitude, stimulus pulse duration, muscle position and orientation (with respect to the magnetometer), muscle length, muscle temperature, animal size, muscle size, time since the muscle lost its circulation, number of stimulations the muscle has received. These variables were measured and controlled as carefully as possible during each experiment. The dependent variables were: electric potential, magnetic flux density and muscle twitch force. These variables were measured as functions of time during maximally stimulated isometric contractions of the muscle.

5.1.2 The Specimens

An excised skeletal muscle was to be the biomagnetic source of study in this research. The choices of animal and specific muscle were directed by convenience and conforming to standards in the study of other biological signals. It was important to be able to compare the mechanical response of the muscle with previously published results and

compare the magnetic signal with electric potential measurements.

The frog was chosen because it is easily obtained and maintained (in cold storage). They are inexpensive and easy to work with. In these experiments, fresh bullfrogs (Rana catesbeiana) and leopard frogs (Rana pipiens pipiens) were used. For preliminary investigations, frogs were purchased from several scientific supply companies and were stored at 42 degrees Fahrenheit. For the data presented in this thesis, the frogs were caught in local ponds (southern Ontario, Canada) in early August, were identified (Wright & Wright, 1949) and were used within two days.

The gastrocnemius muscle was chosen as the muscle of study because it is a standard in the electrophysiological literature. It is composed of three types of fast-twitch fibres (Smith & Ovalle, 1973). Less than 1% of the fibres can be classified as slow twitch (Engel & Irwin, 1967). The sartorius was also considered. Its fibers are more uniformly parallel than the gastrocnemius which would simplify modelling of the muscle as a biomagnetic source. The gastrocnemius muscle has a significantly greater cross-sectional area than the sartorius and was expected to produce a larger magnetic signal. The prospect of a larger signal was a powerful argument since all biomagnetic signals are very small and difficult to measure.

In conclusion, it was decided that an excised frog gastrocnemius muscle be used for study of the MMG. This muscle preparation provided the opportunity to strictly control the independent physiological variables and study their individual effects on the dependent variables (EMG, MMG and twitch force). Since the muscle was stimulated indirectly

with single square pulses to the sciatic nerve, the EMG and MMG signals are better described as the "whole muscle action potential" (WMAP) and the "magnetic action flux" (MAF) respectively. The MAF is the magnetic equivalent of the electric action potential. It should be noted that since indirect stimulation was used, the WMAP and MAF are "compound" signals: the summation of signals from all of the active fibres.

5.2 MUSCLE ENVIRONMENT

5.2.1 Muscle Bath

The muscle bath consisted of a shallow (7 mm deep) lidless Plexiglass box, measuring 3.4 cm by 6.8 cm. It is shown in Figure 5.1. Care was taken to use non-metalic materials so that the magnetic field of the muscle would not be distorted. A nylon screw in a threaded hole in the end wall of the box provided an anchor for the proximal end of the muscle. The bath was filled with a medium so that it just overflowed. A small slab of Plexiglass was placed in the bath, under the muscle, so that the center of the muscle was kept in line with the two silk threads tied to its ends. A paper tissue, soaked in the medium, was placed over the muscle to keep its top surface from becoming dry.

As discussed in section 6.2, two bath media were used in these experiments. Mineral oil (liquid paraffin) provided an excellent medium for the WMAP measurements because it is a good electrical insulator. MacKenzie's toad saline (McDonald, Boutilier & Towes, 1980; de la Lande, Tyler & Pridmore, 1962) was also used because it closely simulates the

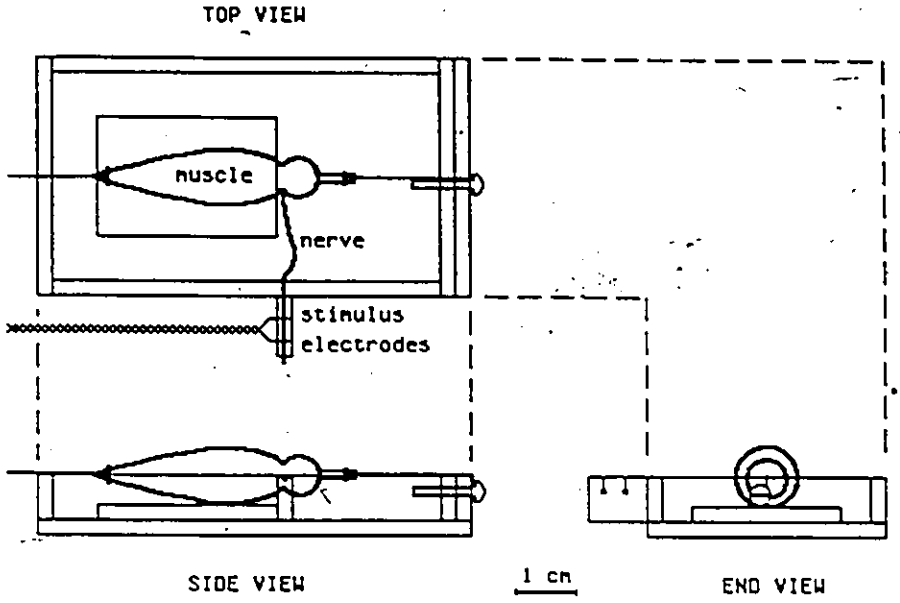


Figure 5.1 : Top, side and end views of the muscle bath. The bath consisted of a shallow lidless box constructed of Plexiglass. To minimize distortion of the muscle's magnetic field, no metallic materials were used.

ionic composition of the interstitial fluid. It is a good electrical conductor and tended to short circuit the WMAP electrodes. The saline solution provides a more natural environment than mineral oil and the MAF signal was found to have a much greater amplitude when the saline medium was used (see discussion of section 6.2).

The muscle's nerve was held by a small trough mounted on the side wall of the bath. As shown in Figure 5.1, it ran perpendicular to the wall and was flush with the top of the wall. The trough was 1 mm deep and approximately 1 cm long. Mounted across the trough, and separated by 4 mm were the stimulation electrodes. Each consisted of a

loop of chloridized silver wire, soldered to a copper lead. The leads (from the stimulator) were twisted about each other so as to reduce the magnetic field generated by the stimulus pulse. The trough was filled with the bath medium and a strip of paper tissue, soaked in the bath medium, was laid on top to prevent the nerve from drying out.

5.2.2 Muscle Manipulator

The magnetometer dewar was a cylinder of 35 cm diameter and 110 cm long. It had a capacity for 10 liters of liquid helium and a total mass of 30 kg. It was supported by a stand constructed of 19 mm plywood. The magnetometer was too large and too heavy to move about the muscle preparation to perform a spatial mapping of the muscle's magnetic field. Consequently, a device was developed to move the muscle preparation with respect to the magnetometer sensing coil.

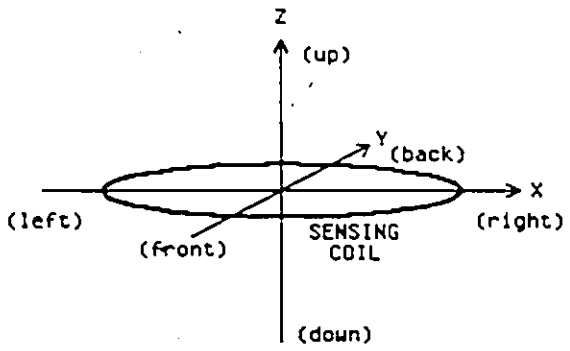


Figure 5.2 : Cartesian coordinate system used to describe the position of the centre of the muscle with respect to the centre of the magnetometer sensing coil. Directional cues such as "up" and "front" have been included to simplify discussion of the muscle positioning device.

The muscle manipulator provided remote control of the muscle position (from outside the magnetometer shielded room) in three orthogonal directions. The three directions were labelled as the x, y and z axes. As shown in Figure 5.2, the z-axis was coincident with the axis of the magnetometer gradiometer coils. The positive direction was up. When viewing the magnetometer from the "front", the x-axis was directed from the left side to the right and the y-axis was from the front to the back. The origin of the coordinate system was the center of the magnetometer sensing coil.

Again, it was important to not use any metallic materials that would distort the muscle's magnetic field. As for the muscle bath, the muscle manipulator was constructed of Plexiglass. Chloroform was used as a solvent cement to fasten pieces together. Conventional worm drives were replaced with an hydraulic system of glass syringes filled with mineral oil. Elastic bands were used to hold the glass syringes and their plungers firmly against plexiglass supports.

Pairs of syringes were connected by stiff hoses. One of the syringes was mounted on the muscle manipulator and the other was passed through a port in the side wall of the shielded room in which the magnetometer was kept. When the plunger of the syringe outside the shielded room was pushed into the syringe, the plunger would be forced out of the syringe mounted on the muscle manipulator. The fluid in each syringe pair and connecting hose was mineral oil. The combination of the close fitting glass syringes and the lubricating fluid yielded a smooth and precise mechanism for actuating changes in muscle position.

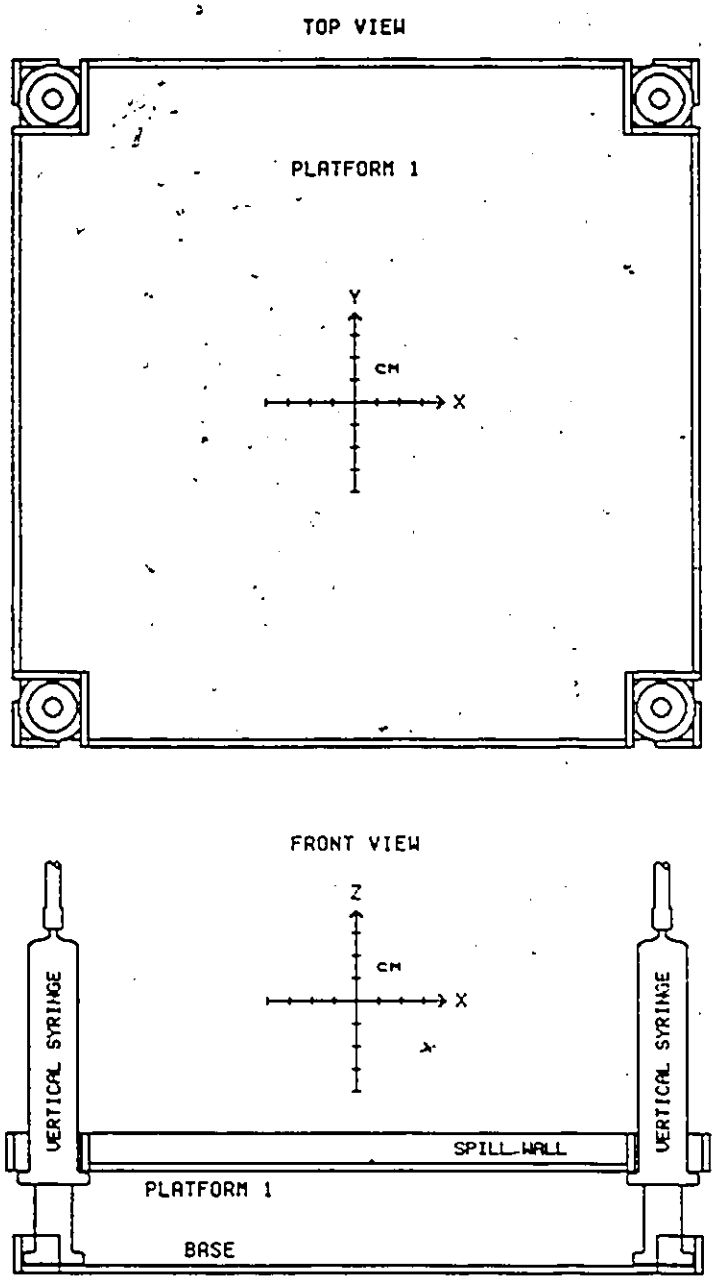


Figure 5.3 : Base and platform 1 of the muscle positioning device. Platform 1 was moved vertically by means of a syringe (filled with mineral oil) at each of its four corners.

The base of the muscle manipulator consisted of a 30 cm square sheet of 4.7 mm thick Plexiglass. A platform of the same dimensions was supported above the base by four glass syringes, one in each corner. It is labelled as "platform 1" in Figure 5.3. These syringes provided the facility for vertical positioning of the muscle. Each of these syringes had a matching syringe connected to the other end of a 3 m hose. When the plunger of one of these remote syringes was pushed into its barrel, fluid was forced into the corresponding syringe of the manipulator. Since the plunger of this syringe was fixed to the base, the barrel of the syringe was forced to rise, carrying the platform with it. When raising or lowering the platform, the four corner syringes were sequentially adjusted by small increments until the desired position was obtained. The platform was always kept as close to horizontal as possible. A valve at the remote end of each syringe pair was closed when the desired vertical position was achieved so that the weight of the platform did not force the fluid back out to the remote syringe.

A second platform was mounted on top of platform 1 as shown in Figure 5.4. Platform 2 was caused to slide in the y-axis direction by a syringe mounted in the horizontal plane. The barrel of the syringe was fastened to supports mounted on platform 1 while the plunger was fastened to supports mounted on platform 2. Guide rails mounted on the under side of platform 2 fit snugly between the support rails mounted above platform 1. These rails kept platform 2 properly aligned with respect to platform 1.

A final platform was mounted on top of platform 2 as shown in Figure 5.5. Platform 3 was caused to slide in the x-axis direction by a

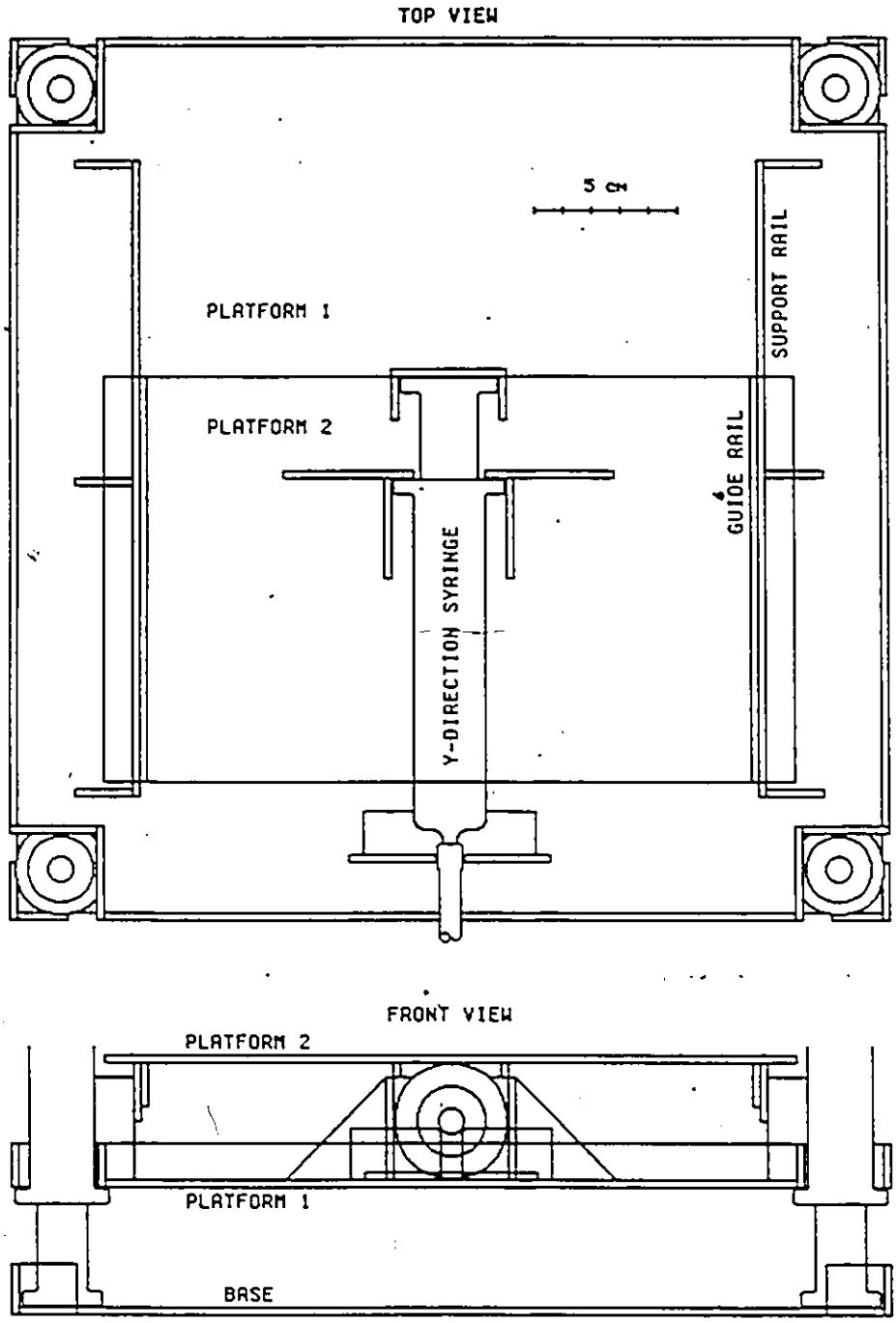


Figure 5.4 : Top and front views of platform 2 (and 1).

syringe mounted in the horizontal plane, but perpendicular to the one below it. As before, the barrel of the syringe was fastened to the platform below and the plunger was attached to the platform above. A snug fitting set of rails was also used as before. Platform 3 had a spill wall around its perimeter to catch any of the bath medium that might spill during an experiment.

The muscle bath was mounted on top of platform 3. As shown in Figure 5.6, the muscle length syringe and the force transduction system (discussed in the next section) were also mounted on this top platform. By means of the vertically oriented syringes, the y-axis syringe and the x-axis syringe, the center of the muscle bath could be moved to any position under the magnetometer sensing coil in a cube measuring 8 cm in the z (vertical) direction, 9 cm in the y direction and 9 cm in the x-direction.

5.2.3 Force Transducer

The force transduction system consisted primarily of a mineral oil filled syringe and an arterial pressure transducer (manufactured by Gould). This design was used because most commercial force transducers involve a metallic strain element and are often housed in a metal case. As previously stated, metal in the vicinity of the muscle may cause distortions in its magnetic field. The Gould pressure transducer was encased in plastic and the syringe was made of glass.

A loop of silk surgical suture was tied around the head of the syringe plunger. A second piece of the silk thread was used to tie the distal end of the muscle to this loop. The proximal end of the muscle

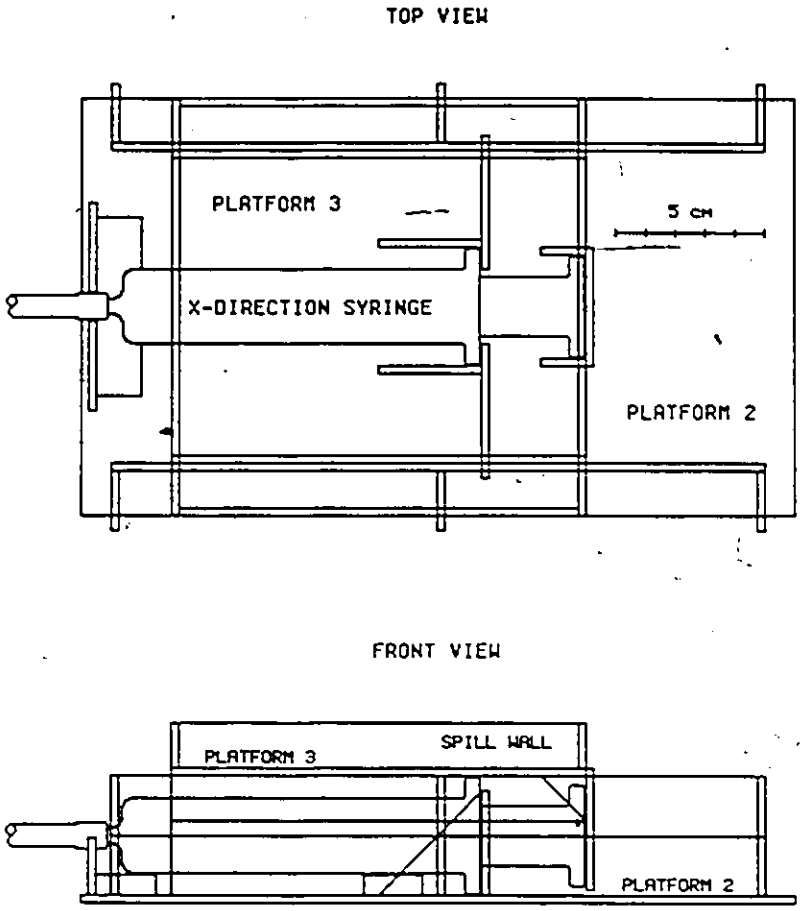


Figure 5.5 : Top and front view of platform 3 mounted on platform 2. Platform 3 was caused to move from left to right by forcing mineral oil into the "x-direction" syringe.

was tied to a screw in the side wall of the muscle bath. The syringe was connected to the pressure transducer through a three-way valve. When the muscle twitched, it pulled on the thread tied to the syringe plunger, caused a drop in pressure in the mineral oil in the syringe. This pressure fluctuation was then detected by the pressure transducer.

From "impulse" and "step" load measurements, the frequency response of this force transduction system was determined to be DC to 50 Hz. The rate of "roll-off" was not determined. The sensitivity was limited to approximately 2 mN (corresponding to 0.2 g) by the analogue-to-digital converter resolution.

A second syringe was involved in the force transduction system. It was connected to the first syringe through the same three-way valve that connected the pressure transducer. This second syringe was used to change the length of the muscle by introducing or removing fluid from the syringe the muscle was fastened to. The three-way valve was used to disconnect the "muscle length" syringe from the force transduction system during measurements.

5.2.4 Action Potential Electrodes

Since the magnetometer sensing coil (2.54 cm diameter) was approximately the same size as the muscle (approximately 3 cm in length), the magnetometer was sensitive to magnetic signals produced anywhere in the muscle. The magnetometer yielded a "whole muscle" measurement. To allow comparison with simultaneous electric potential measurements, it was necessary to use an electrode type and configuration that would allow measurement of the whole muscle action potential.

It was decided that tying fine silver wires around the muscle circumference at several locations along its length would provide a "whole muscle" signal and would cause the muscle minimal damage. It was felt that needle electrodes would not yield a "whole muscle" signal and would cause significant damage to such a small muscle. Disc surface

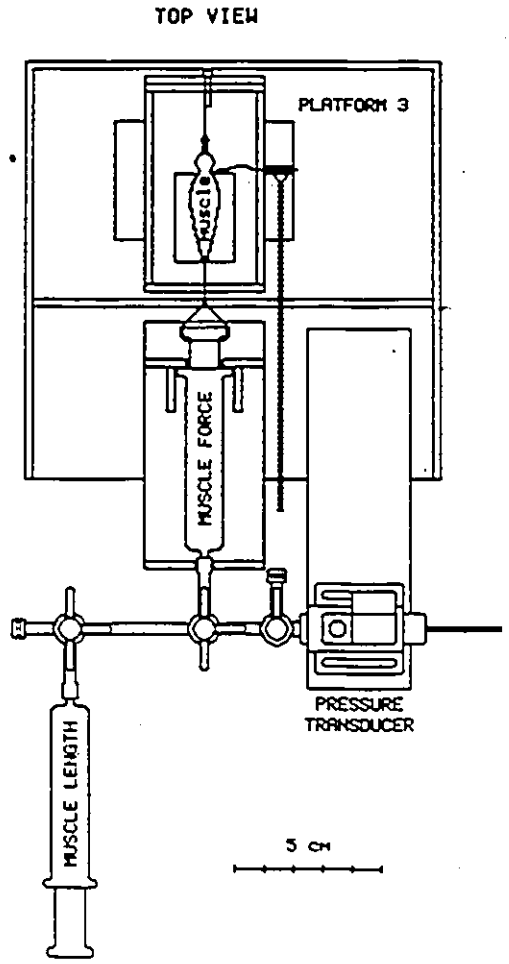


Figure 5.6 : Top view of the muscle bath and force transduction system mounted on platform 3 of the positioning device. The "length syringe" allowed adjustment of the muscle length by changing the amount of fluid in the "muscle force" syringe.

electrodes involve too much metal and would cause distortion of the muscle's magnetic field.

Preliminary experimentation showed that a good signal was obtained when the electrode configuration of Figure 5.7 was used with a

mineral oil bath medium. A ground electrode was tied around the little remaining tissue proximal to the knee. The electrode tied around the middle of the muscle was connected to the inverting input of the preamplifier. The electrode connected to the positive input of the preamplifier was tied around the distal end of the muscle where it was just starting to become tendonous. With this electrode arrangement, the measured whole muscle action potential (WMAP) had a large initial peak in the positive direction, a smaller peak in the negative direction and finally a very small positive deflection.

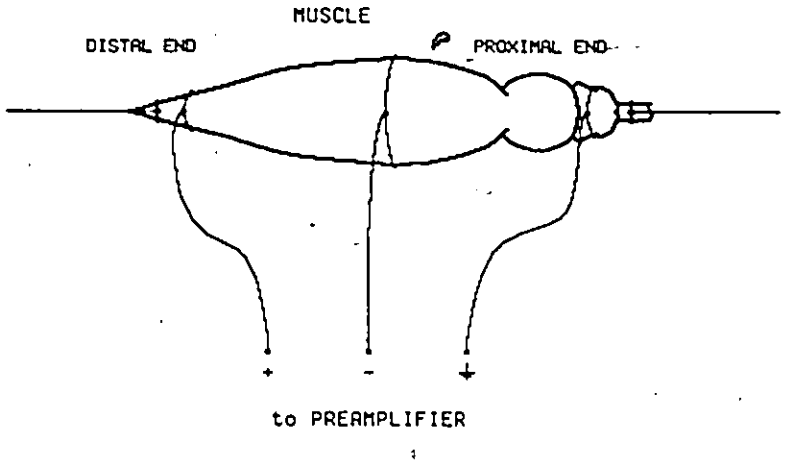


Figure 5.7 : Chloridized silver wire electrodes used for measuring the whole muscle action potential (WMAP). The knee joint was represented by the spherical structure at the proximal end of the muscle.

5.2.5 Muscle Stimulation

The muscle was indirectly stimulated through application of an electric pulse to the sciatic nerve. As discussed earlier in this chapter, the sciatic nerve was laid along a trough that was mounted on the side of the muscle bath. Mounted across the trough, and separated by 4 mm, were two loops of chloridized silver wire. The sciatic nerve was laid over the two loops of wire. The trough was filled with the bath medium and a strip of paper tissue soaked in the bath medium was laid on top.

A voltage pulse was applied to the two loops of wire to stimulate the nerve and activate the muscle. A "Pulsar 6i" muscle stimulator (Frederick Haer & Co. of Brunswick, Maine) was used to provide single square voltage pulses of 100 microsecond duration and less than a half volt amplitude. Submaximal contractions of any amplitude could be generated by adjusting the amplitude of the voltage pulse. The smaller amplitude pulses depolarize only a fraction of the fibers in the nerve bundle enough to initiate action potentials.

The muscle stimulator was triggered by a signal from the interfacing instrument. Generally, it was desired to half fill the computer data buffer with noise before recording the muscle response. A counter, set by a 4-BIT DIP switch on the front panel of the interfacing instrument, was used to count the number of samples collected by the computer. When the number of samples set by the switch was counted, the trigger signal was sent to the muscle stimulator.

5.3 EXPERIMENTAL PROCEDURE

5.3.1 Set Up

The experiments were performed in a laboratory at the Erindale College campus of the University of Toronto in Mississauga, Ontario (50 km from McMaster University). The SQUID magnetometer was being used by other researchers during the tenure of these experiments. Consequently, the apparatus specific to these measurements had to be set up before each experiment and completely removed afterward. This added considerably to the time and effort required to perform a set of measurements.

Once unpacked, power was applied to all equipment for warm-up. Approximately one hour was allowed for thermal settling. During this time preparations were made for the dissection and calibration procedures.

As discussed in section 3.1.3, noise in the magnetometer signal was a major problem. Of greatest concern was a very large 60 Hz artifact from the power distribution system of the laboratory and equipment in adjacent laboratories. The most effective tool in reducing the 60 Hz noise was a 61 cm diameter loop of gauge #2 copper cable. As part of the set-up procedure, this copper ring was positioned around the magnetometer sensing coil. An analogue output signal from the magnetometer was displayed on an oscilloscope. The copper ring was suspended so that the magnetometer sensing coil was at its center. The orientation of the ring was then adjusted to minimize the 60 Hz signal that was initially quite prominent in the oscilloscope trace. Enough of the 60 Hz noise

could usually be removed that the oscilloscope trace looked like white noise. Typical noise spectra under these balanced conditions are shown in Figure 3.3.

5.3.2 Calibration

After the one hour period allowed for thermal stabilization of the electronic equipment, the analogue circuits were calibrated. This involved providing known inputs to the analogue systems and using these values with the resulting integer data collected by the computer to calculate calibration factors. This was done for the WMAP (electric potential) amplifiers and the force transduction system.

As described in section 3.2.3, a battery powered circuit was developed to provide a 200 kHz square wave to calibrate the amplifiers for the WMAP signal. The peak-to-peak amplitude of the square wave could be set to 1.00 mV or 10.0 mV. During the SET-UP section of the SMDA programme, the computer collected a set of data values for calibration purposes. Some of these data values were digital representations of the signal from the WMAP circuits. The computer calculated a calibration factor by dividing the voltage amplitude, entered via the computer keyboard, by the corresponding integer data. The calibration factor then represented the voltage corresponding to the least significant BIT of the WMAP integer data. This WMAP signal calibration factor was stored with every data file during each experiment.

Either a 20 g mass or a 50 g mass was used to load the force transduction system during calibration procedures. A silk thread was tied to the loop on the end of the force syringe plunger. The thread

was draped over a pulley so that the calibration mass could be hung on it. A small Plexiglass frame (used only during calibration procedures) supported the pulley. As for the WMAP signal, the computer collected data from the interfacing instrument ADC while the force transduction system was loaded with the calibration mass. The calibration factor was calculated by multiplying the mass by the acceleration due to gravity and dividing by the resulting ADC integer.

The calibration of the magnetometer was specified by the manufacturer in terms of the average flux density through the cross-sectional area of the sensing coil required to cause the SQUID to jump one flux quantum. A value of 201 pT had been determined by a CTF Systems Inc. technician for the magnetometer used in this research. That is, if an average magnetic field of 201 pT is applied to the sensing coil of the magnetometer and the source is local enough to cause negligible contribution in the other two gradiometer coils, then a field corresponding to one flux quantum will be set up in the SQUID. One flux quantum corresponded to the sixteenth BIT of the 32-BIT magnetometer output. The calibration factor for this magnetometer was thus 201/32768 = 0.006134 pT/integer.

5.3.3 Dissection

The dissection and preparation of the excised muscle followed calibration of the instrumentation. The dissection started with decapitation of the frog. The mass and nose-to-crotch length of the frog were recorded.

The skin was removed from abdomen and leg. The leg muscles were kept wet with McKenzie's saline. The coccyx and flesh of the lower back were removed to expose the sciatic nerves. A silk surgical suture was tied around the sciatic nerve as close to its spinal root as possible. The nerve was cut on the spinal side of the knot. It was then freed from connective tissue and branches as far as the popliteal fossa. A glass probe and fine scissors were used in this procedure. The freed nerve was approximately 4 cm long and was draped over the gastrocnemius muscle and kept wet with MacKenzie's saline for the remainder of the dissection.

The gastrocnemius muscle was excised by first tying a surgical suture to the achilles tendon and then cutting the tendon distal to the knot. The gastrocnemius was then freed from the tibia and the tibia was cut just distal to the knee. All of the thigh muscles were removed from the femur. A second surgical suture was tied to the femur just proximal to the knee and the femur was cut proximal to this knot. Thus, the excised muscle was held by silk surgical threads, tied to the tendon on the distal end and tied to the femur on the proximal end. The origins of the gastrocnemius muscle, superior to the knee, remained undisturbed.

Throughout the dissection the muscle and nerve were kept wet with MacKenzie's saline. Upon completion of the dissection, the mass and length of the excised muscle were measured and entered into the computer. The excised muscle was used immediately or stored in a bath of McKenzie's saline.

5.3.4 Mounting the Muscle

The excised gastrocnemius muscle of the frog was mounted in a small muscle bath that was described in section 5.2.1. The silk thread tied to the femur (proximal end of the muscle) was clamped under the head of a nylon screw mounted in the side of the muscle bath. The silk thread tied to the Achilles tendon (distal end of the muscle) was tied to the loop attached to the plunger of the force transducer syringe. The slack in this suspension system was removed by withdrawing fluid from the force syringe with the muscle length syringe.

The bath was usually filled with MacKenzie's saline. A mineral oil medium was used for some measurements (see section 5.3.8). The sciatic nerve was laid down the trough mounted on the side of the muscle bath and over the stimulation electrodes. The trough was filled with bath medium. To prevent drying of the top surface of the muscle and the nerve, pieces of paper tissue soaked in the bath medium were draped over the muscle and the nerve.

As discussed in section 4.1.3, a small section of the SMDA programme was devoted to determination of the "rest length" of the muscle. The muscle length syringe was used to withdraw fluid from the force transducer syringe so that the muscle would start to become stretched. The output of the pressure transducer was monitored during this procedure. The muscle was stretched slightly past its rest length so that a very small signal was output by the pressure transducer. Then, a little fluid was put back in the force syringe so that the muscle was not stretched and the pressure transducer yielded a zero output. The muscle was then considered to be at its rest length and the three-way valve was

used to disconnect the muscle length syringe from the force transduction system.

The last procedure, before measurements could commence, was to position the muscle under the magnetometer sensing coil. The muscle was raised under the tail of the magnetometer dewar until its top surface was 1 mm below the surface of the dewar. This placed the center of the muscle 17 mm below the plane of the magnetometer sensing coil. The muscle was visually aligned with marks on the side of the magnetometer dewar tail so that the center of the muscle would be directly below the center of the sensing coil. This was the starting position of the muscle for all experiments.

The starting position of the muscle corresponded to coordinates (x,y,z) = (0,0,0) in the SMDA programme. For the presentation of results in this document, the origin of the coordinate system was chosen to be at the center of the sensing coil. The center of the muscle in this initial position would therefore be (x,y,z) = (0,0,-17mm). The positions of the plungers of the syringes used to remotely control the muscle positioning device were entered into the computer so that the computer could calculate the new plunger positions for any desired position of the muscle.

5.3.5 Position Experiment

The purpose of the "position experiment" was to investigate the spatial variation of the MAF. As discussed earlier, in section 5.2.2, the magnetometer was too large to move around the muscle. The muscle was therefore moved around the magnetometer sensing coil.

Since the magnetometer dewar tail was so large, the muscle could not be in close enough proximity to obtain a good signal for some orientations of the muscle. For example, the muscle could not be moved up into the same plane as the sensing coil without being 5 or 6 cm from the coil axis. Also, when the long axis of the muscle was coincident with the axis of the sensing coil, the center of the muscle was at least 4 cm from the sensing coil and no signal could be detected.

The maximum MAF amplitude was obtained with the muscle in the position and orientation shown in Figure 5.8. The long axis of the muscle was oriented parallel to the y-axis. The muscle was positioned 17 mm below the edge of the magnetometer sensing coil with its center in the xz-plane. That is, the muscle center was located at position (x,y,z) = (-12mm,0,-17mm).

Only three spatial mappings of the MAF yielded a signal-to-noise ratio greater than 0 dB. For each of these, the MAF and the twitch force were measured at several positions with respect to the sensing coil. At each of these positions, measurements for three data files were made. One of these files contained the average of 9 successive data records. The other two files contained single unaveraged data records. The MAF and twitch force data vectors were each of 2048 data points, with the first half being a noise record and the second half having the muscle response. As for most of the measurements reported in this thesis, the muscle was maximally stimulated for isometric contractions at its rest length.

The first of the three spatial mappings had the muscle oriented parallel to the y-axis as in Figure 5.8. The muscle was moved parallel

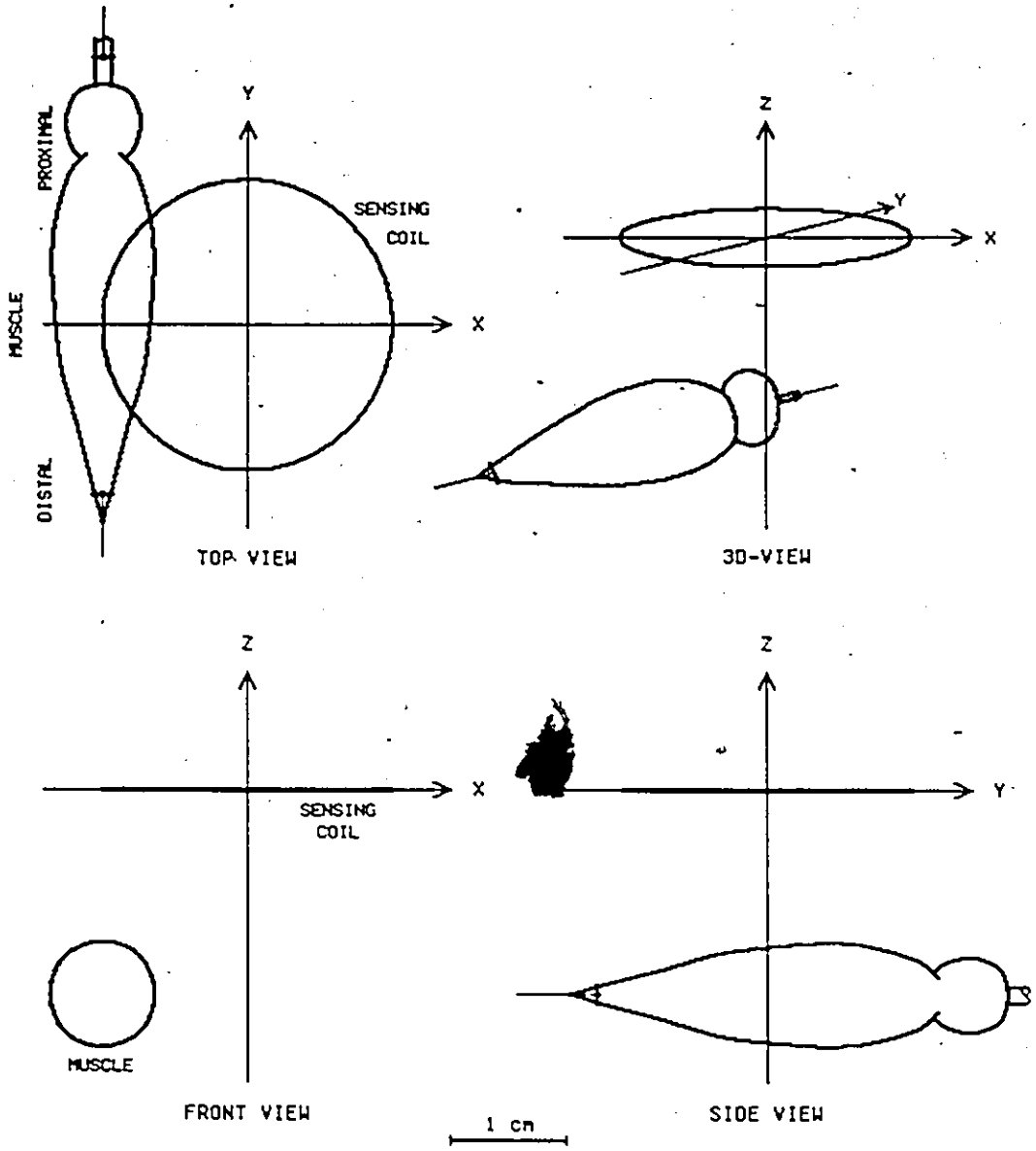


Figure 5.8 : Position and orientation of the muscle with respect to the magnetometer sensing coil for which a maximum MAF amplitude was obtained. The centre of the muscle was located at (x,y,z) = (-12mm,0,-17mm).

to the x-axis as shown in Figure 5.9a. Preliminary investigation showed that the middle position yielded the largest signal. The signal at this middle position was measured between each of the other measurements to provide a baseline for muscle fatigue effects. Thus, measurements were made with the muscle at the positions indicated in the figure in the following order: 2, 3, 2, 1, 2.

The second spatial mapping kept the muscle in the same orientation as the first, but the muscle was moved away from the sensing coil in the z direction. As shown in Figure 5.9b, the muscle was kept lined up under the edge of the sensing coil to obtain a maximum amplitude signal. Other than the change in muscle position, the experimental conditions were the same as for the first spatial mapping.

The final spatial mapping that yielded a detectable magnetic signal was with the muscle axis parallel with the y-axis and intersecting the z-axis. The muscle was moved in the -y direction. Once again, the maximum amplitude signal was obtained when the muscle was centered under the edge of the sensing coil (see Figure 5.9c). Measurements were made at this middle position between measurements at the other positions to provide a baseline for muscle fatigue effects. Thus, measurements were made at the labelled positions in the following order: 2, 3, 2, 1, 2.

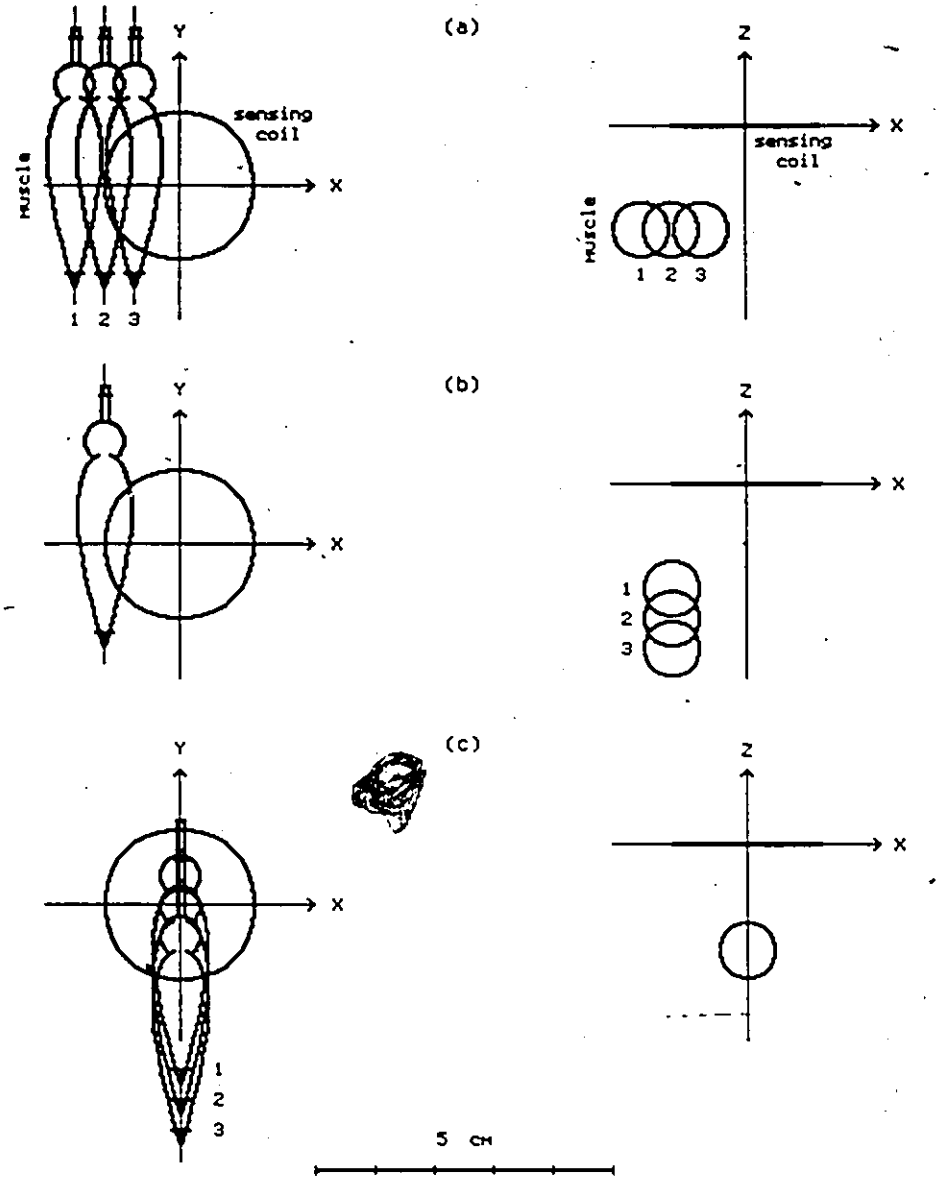


Figure 5.9 : Three spatial mappings of the muscle with respect to the sensing coil of the magnetometer. For each of these, a top view is shown on the left and a side view is shown on the right. Part (a) shows the muscle being moved horizontally in the x direction. Part (b) shows the muscle being moved down in the z direction. Part (c) shows the muscle being moved horizontally in the y direction.

5.3.6 Length Experiment

The purpose of the "length" experiment was to study changes in the MAF due to changes in muscle length. For all measurements in this experiment, the muscle was constrained to isometric contractions and kept in the position for which the MAF amplitude was a maximum (Figure 5.8). The muscle length was measured with respect to its rest length (the length for which the force transducer barely registered zero). For each muscle length, the muscle was maximally stimulated and measurements were made for three data files as for the "position" experiment. A set of measurements with the muscle at its rest length was made several times during this experiment to allow corrections for muscle fatigue.

A set of MAF and twitch force measurements was first made with the muscle set to its rest length. The muscle length syringe was then used to introduce more fluid to the force transducer syringe so that upon the next contraction, the muscle could shorten to 90 % of its rest length. The muscle length syringe was subsequently adjusted to allow measurements for muscle lengths of 95%, 105% and 110% of rest length. The actual sequence of muscle lengths for which measurements were made was: 100%, 90%, 95%, 100%, 105%, 110%, 100%.

5.3.7 Stimulus Experiment

The purpose of the "stimulus" experiment was to discover what changes occur in the MAF when the amplitude of the stimulus pulse to the sciatic nerve was varied. Since the number of motor units activated by a stimulus pulse to the nerve is not a simple function of the stimulus amplitude, the twitch force amplitude was used as a measure of the level

of muscle activation. Thus, the stimulus pulse amplitude was adjusted to yield twitch force amplitudes that were desired fractions of the maximum force response for a given muscle preparation. The isometric contractions were performed with the muscle at its rest length and in the position and orientation shown in Figure 5.8.

Measurements from a maximal stimulus were made between each of the submaximal stimulations. The stimulus pulse amplitude was adjusted so that measurements were made for muscle twitch amplitudes that were approximately 80%, 60% and 40% of the maximum possible twitch force amplitude. The sequence of measurements was thus: 100%, 80%, 100%, 60%, 100%, 40% and 100%.

5.3.8 Muscle Bath Medium Experiment

The purpose of the "bath medium" experiment was to demonstrate that the electrical properties of the bath medium can have a strong influence on the measured MAF. Two media were used in this experiment. Mineral oil was used to provide an insulating medium. MacKenzie's toad saline (McDonald, Boutilier & Towes, 1980; de la Lande, Tyler & Pridmore, 1962) is a good electrical conductor and provided a much more natural environment for the muscle. The muscle position and orientation was as shown in Figure 5.8 (optimized for maximal MAF amplitude). Maximal isometric contractions were elicited with the muscle at its rest length.

First, a set of measurements were made with the muscle suspended in the MacKenzie's saline bath medium. For the next set of measurements, the saline solution was removed and replaced with mineral oil.

The mineral oil was then removed and the final set of measurements were made with a MacKenzie's saline bath medium. The results of this experiment are presented in section 6.2 and are discussed in section 7.2.

CHAPTER 6

RESULTS: COMPOUND MAGNETIC ACTION FLUX OF THE FROG GASTROCNEMIUS

6.1 RESPONSE TO A SINGLE PULSE STIMULUS

The MAF signal-to-noise ratio was found to range from less than 0 dB to over 18 dB. From preliminary trials, it was found that the optimal signal-to-noise ratio was obtained when the muscle was in the horizontal plane and just under one edge of the magnetometer sensing coil as shown in Figure 5.8. With the muscle in this position and orientation, the signals shown in Figure 6.1 were obtained. Each of these signals is the magnetic response of the muscle to a single pulse stimulus of 100 microsecond duration and 270 millivolt amplitude. This stimulus was applied to the sciatic nerve and maximally activated the muscle (the twitch force amplitude could go no higher). The muscle was constrained to isometric contractions at its rest length. The MacKenzie's saline bath was at a temperature of 23.3 Celsius. The excised muscle had a rest length of 33 mm and had a mass of 2.32 grams. The muscle had been dissected from a leopard frog of 47.9 g and a nose-to-crotch length of 81 mm.

As can be seen from Figure 6.1, a stimulus artifact appeared almost immediately after the sciatic nerve was stimulated (time=0.0). There was a three microsecond delay between the stimulus and when the muscle started to generate its magnetic signal. The MAF was biphasic,

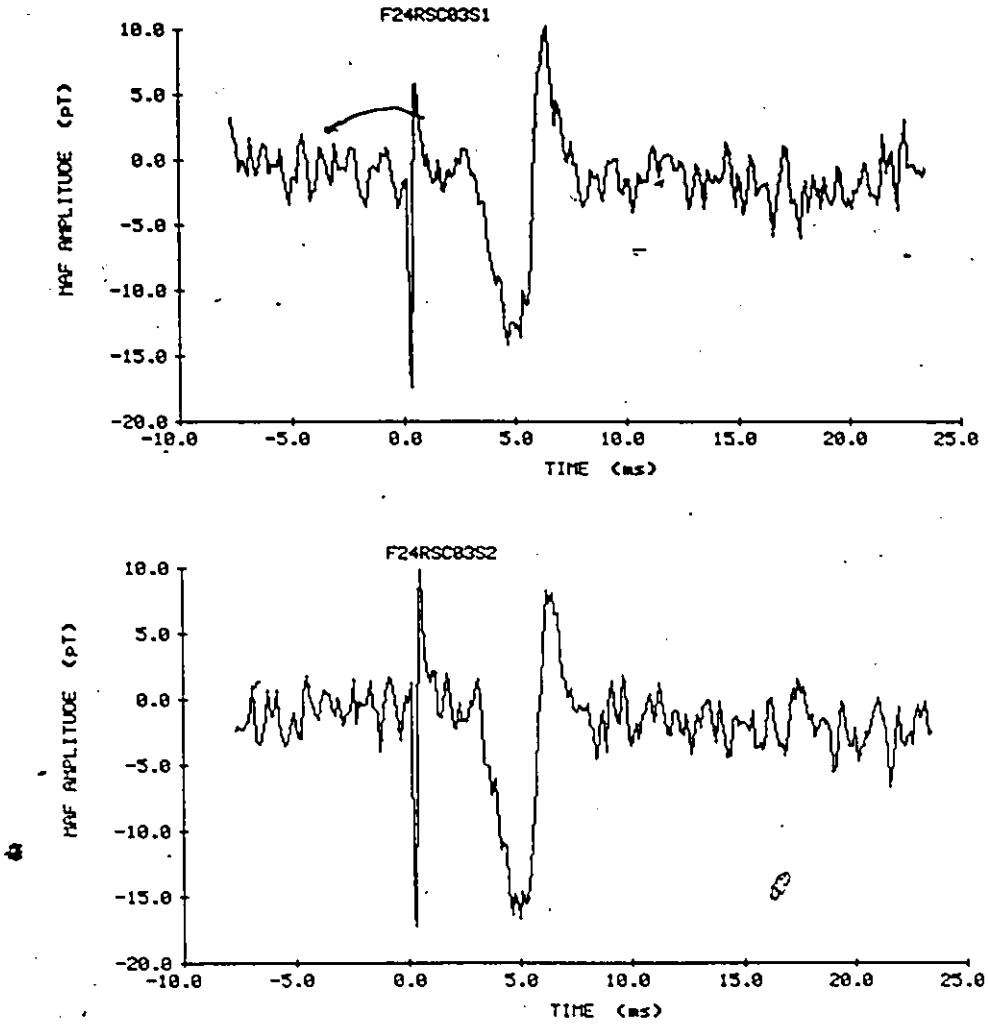


Figure 6.1 : Two recordings of the compound magnetic action flux (MAF) from an excised frog gastrocnemius in the position shown in Figure 5.8.

with the initial phase having a slightly greater amplitude and duration than the second phase. The MAF had a total duration of approximately five millisceconds.

Two MAF records were shown in Figure 6.1 to demonstrate the reproducibility of the signal. Most of the difference between the two recordings is due to relatively large environmental magnetic noise. From a collection of eight responses from this same muscle (under the conditions described above), an average MAF peak-to-peak amplitude of 35.5 picoTesla (pT) was calculated. The average MAF delay and duration were 3.48 ms and 4.12 ms respectively.

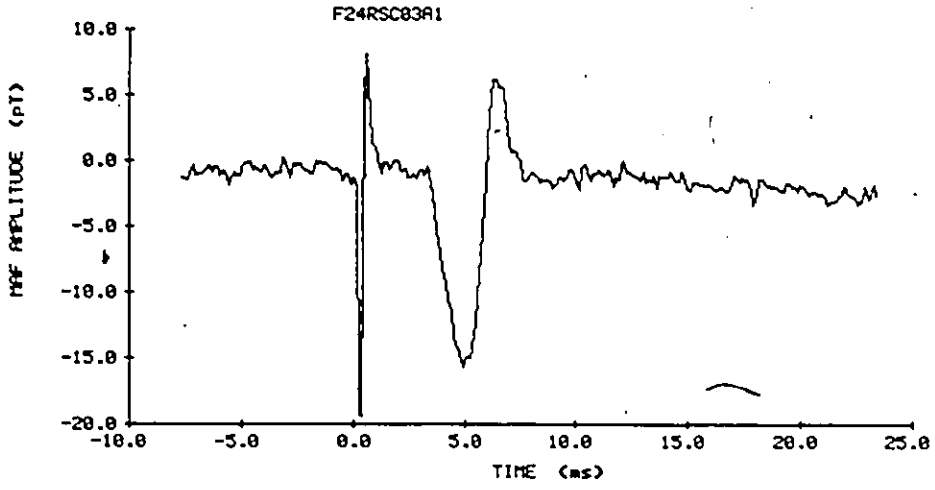


Figure 6.2 : Average of 9 successive MAF's for the same conditions as for the MAF's of Figure 6.1.

The MAF shown in Figure 6.2 is the average of 9 individual responses. This record has a greatly improved signal-to-noise ratio over those of Figure 6.1 (the random noise has largely cancelled itself while elements of the muscle's magnetic response have reinforced themselves). Four records, each an average of 9 MAF responses, were collected from

the same muscle under the same conditions as for the signals of Figures 6.1 and 6.2. The average MAF amplitude was found to be 28.3 pT with a standard deviation of 2.4 pT. The mean MAF delay and duration were 3.36 ms and 4.33 ms respectively.

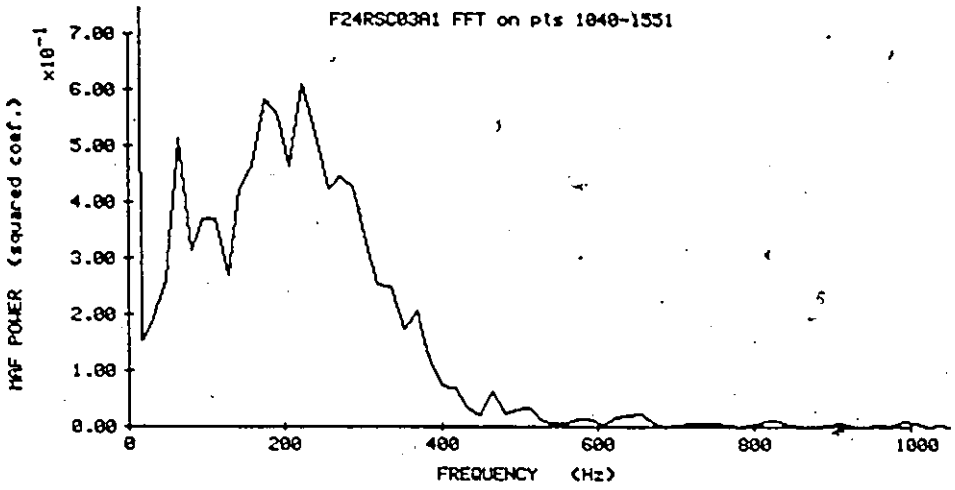


Figure 6.3 : Power spectrum of the averaged MAF of Figure 6.2. This spectrum was derived from the magnitude of the complex coefficients of a Fourier transform of 512 data points (from time 2.0 ms to 64.3 ms).

Several of the physiological variables, such as MAF amplitude and twitch force amplitude, were found to change slightly with each twitch of the muscle. A similar effect was observed by Grieve (1958) for the action potential and twitch tension in the frog sartorius muscle. This "fatigue" effect had to be removed from the data of each experiment before analysis. As described in section 4.3.3, a "fatigue slope" was calculated and used to adjust data values so that they would represent the response that would have been observed for the first twitch of the muscle. The fatigue slope represents the change in a

variable per muscle stimulation. The statistics discussed in this thesis are for data that have been corrected for fatigue. The figures present raw data that have not been corrected for the fatigue effect.

The power spectrum shown in Figure 6.3 was obtained by performing a fast Fourier transform on 512 data points from the MAF of Figure 6.2 (time 2.0 ms to 64.5 ms). The base, upon which the many irregular peaks sit, is reproducible and related to the muscle response. The individual peaks are not reproducible and appear to be due to the noise of the individual record. For example, compare the spectrum of Figure 6.3 with the spectrum of Figure 6.4. These two power spectra were each calculated from the average of 9 MAF responses (all for the same experimental conditions).

Figure 6.5 shows the 'windowed' version of the MAF of Figure 6.2 and its power spectrum. The 'windowing' process was discussed in section 4.3.4. The power spectrum of Figure 6.5 was derived from a Fourier transform of 1024 data points (125 ms) of the windowed MAF. Only 512 data points (62.5 ms) were used in the Fourier transform of the non-windowed data so that the large stimulus artifact could be excluded. The irregular peaks of Figure 6.3 have disappeared and we are left with a smooth distribution that peaks at 183 Hz and has very little power above 400 Hz. Note the DC offset indicated by the value at 0 Hz. This provides a measure of the difference in area between the two phases of the MAF. The mean peak frequency from 'windowed' versions of the four averaged records was 217 Hz with a standard deviation of 5 Hz. To see that the 'windowed' power spectra are reproducible, compare the power spectrum of Figure 6.5 with that of Figure 6.6.

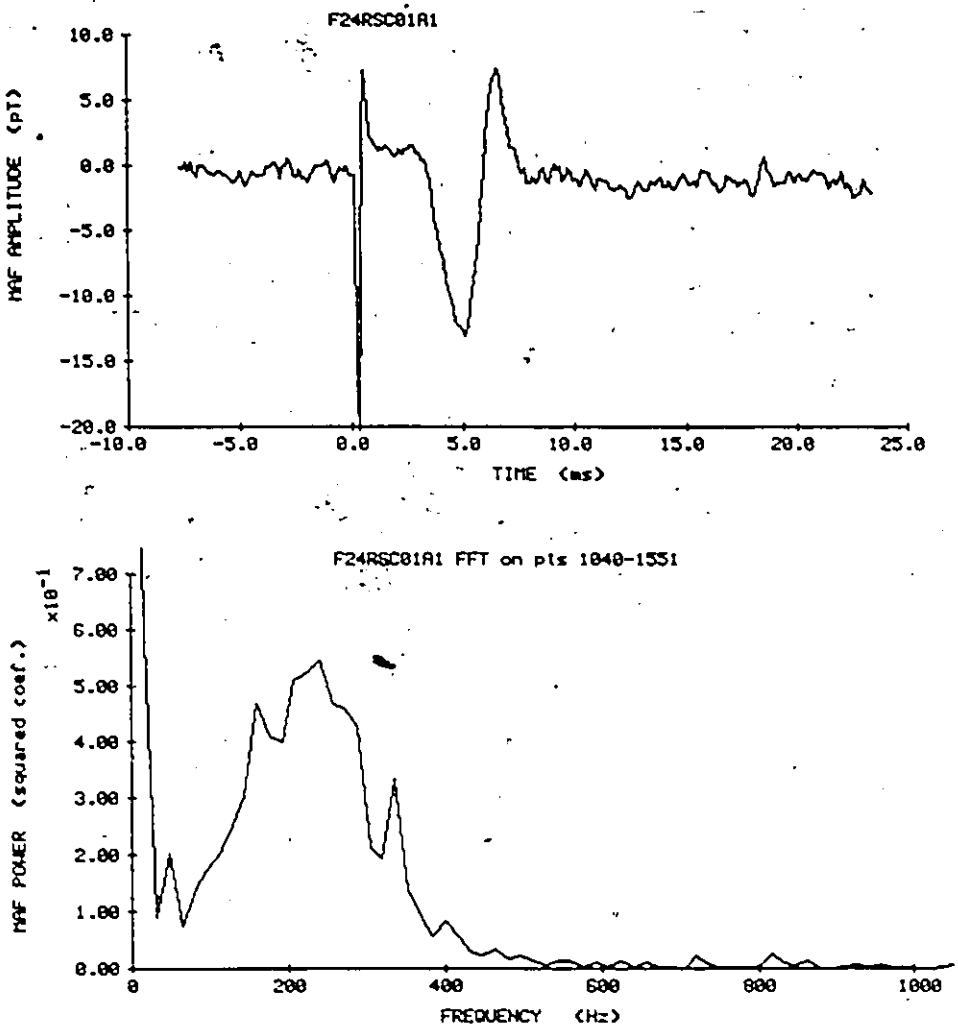


Figure 6.4 : A second average of 9 successive MAF's and the corresponding power spectrum. Experimental conditions were the same as for the first three figures of this chapter.

Figure 6.7 illustrates the relationship between the MAF and the mechanical response of the muscle. The MAF was completed well before the twitch force started to appear. The duration of the twitch was approximately 50 ms and the peak force was 262 milliNewtons (mN).

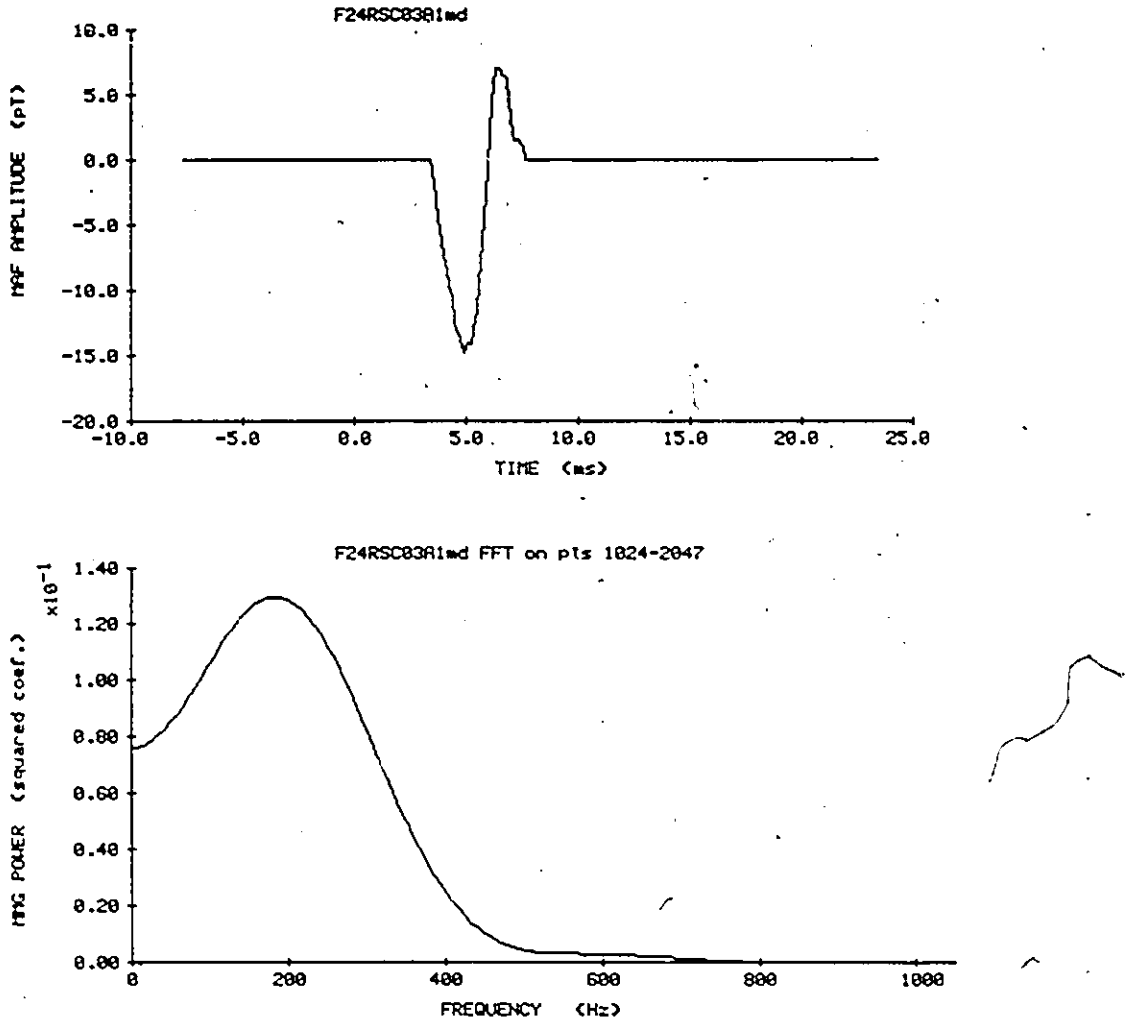


Figure 6.5 : A windowed version of the averaged MAF of Figure 6.2 and the resulting power spectrum. The Fourier transform was calculated from 1024 data points from time 0.1 ms to 125.0 ms.

corresponding to 26.7 grams. A low frequency artifact was occasionally observed at the same time as the muscle twitch. This artifact was probably due to the motion of the muscle during the twitch.

The MAF was found to occur approximately simultaneously with the

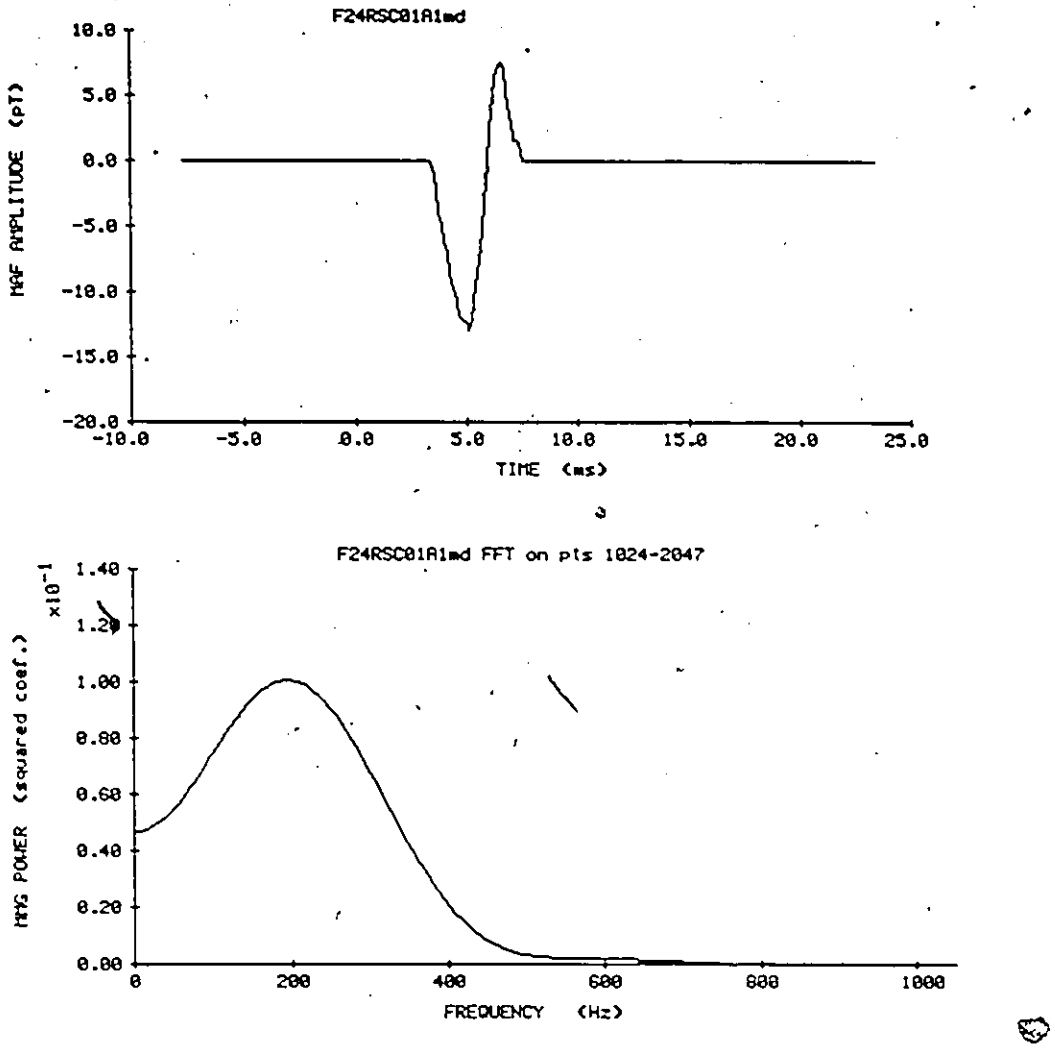


Figure 6.6 : A second power spectrum from a windowed average of 9 successive MAF recordings. This MAF signal is a windowed version of the signal shown in Figure 6.4.

whole muscle action potential (WMAP). Figure 6.8 shows a WMAP from a different leopard frog muscle preparation than for the signals already discussed. From a collection of 9 WMAP records from several muscle preparations, the mean delay and duration were 2.90 ms and 4.33 ms

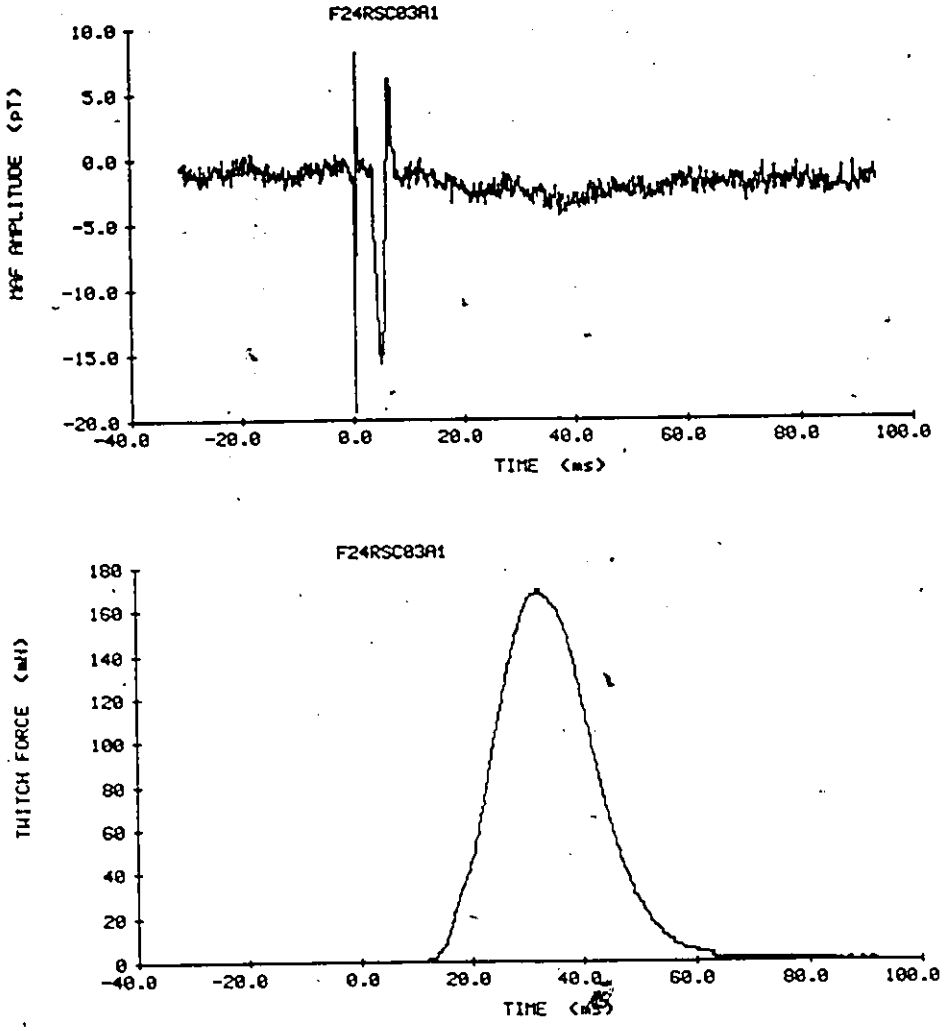


Figure 6.7 : Temporal relationship between the MAF and the muscle twitch force. Part (a) shows the averaged MAF of Figure 6.2. Part (b) shows the averaged muscle twitch force. Both plots are on the same time scale.

respectively. The standard deviations were 0.12 ms and 0.68 ms respectively. The duration of 4.33 ms was for the main biphasic WMAP response. If the small fluctuations after the main biphasic signal are

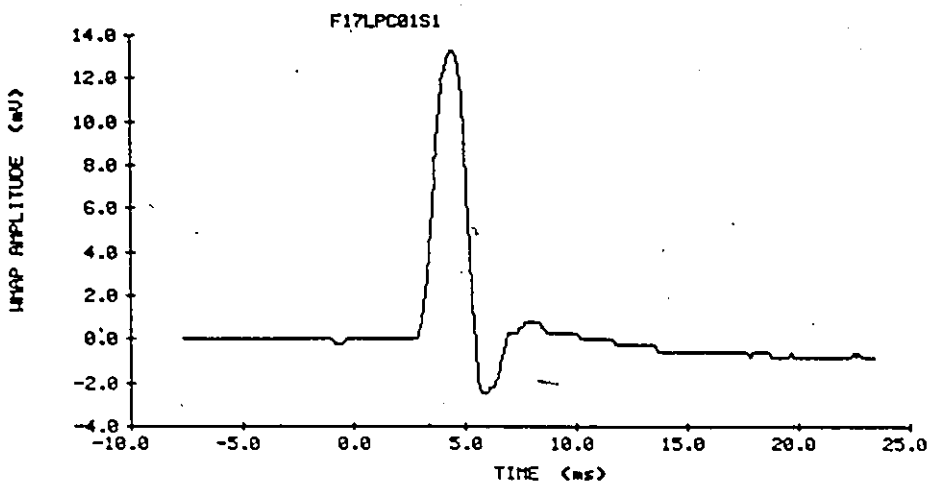


Figure 6.8 : Whole muscle action potential (WMAP). This signal was measured from a different muscle preparation than for all of the previous figures of this chapter.

included, the mean WMAF duration is 7.13 ms (with a standard deviation of 0.61 ms).

A summary of the statistics of the MAF, the magnetic stimulus artifact, the WMAP and the twitch force is presented in Table 6.1. Note that the force "time of peak" is the time from when the nerve was stimulated until the twitch force reached its maximum value, and includes the 13 ms delay. For quantities that were found to change significantly with muscle fatigue, the mean values were corrected so that they reflect the response that would have occured for the very first stimulus. A "fatigue slope" has also been included to indicate how much the quantity changed with each stimulation.

The data from which the statistics of Table 6.1 (except for the WMAP) were derived were obtained from a single leopard frog muscle preparation. To demonstrate the reproducibility of the MAF from frog-

MUSCLE SIGNAL VARIABLES		SINGLE RESPONSES				RECORDS OF 9 AVERAGED RESPONSES			
		N	MEAN	SD	FATIGUE SLOPE	N	MEAN	SD	FATIGUE SLOPE
MAF	AMPLITUDE (pT)	8	35.5	2.1	-0.1	4	28.3	2.4	-0.06
	DELAY (ms)	8	3.48	0.27	0	4	3.36	0.07	0
	DURATION (ms)	8	4.12	0.38	0	4	4.33	0.12	0
	PEAK FREQUENCY (Hz)	-	-	-	-	4	217	5	-0.25
	DC/AMP. (x0.001)	-	-	-	-	4	3.4	0.1	0.02
MSA	AMPLITUDE (pT)	8	28.8	4.2	0	4	27.9	0.5	0
	DELAY (ms)	8	0.12	0	0	4	0.12	0	0
	DURATION (ms)	8	0.92	0.07	0	4	0.95	0.06	0
WMAP	DELAY (ms)	9	2.90	0.12	0	-	-	-	-
	DURATION (ms)	9	4.33	0.68	0	-	-	-	-
FORCE	AMPLITUDE (mN)	8	229	19	-0.4	4	216	2	-0.3
	DELAY (ms)	8	13.2	0.4	0	4	13.2	0.8	0
	DURATION (ms)	8	28.6	1.8	0.12	4	28.39	0.04	0.19
	TIME OF PEAK (ms)	8	29.7	0.9	0.02	4	29.7	0.4	0.02

Table 6.1 : Summary of signal statistics for a leopard frog muscle preparation (MAF: magnetic action flux; MSA: magnetic stimulus artifact; WMAP: whole muscle action potential; N: number of samples; SD: standard deviation). The "fatigue slope" indicates the change in a variable per muscle twitch.

to-frog, the MAF records of two bullfrogs have been presented in Figure 6.9. Each of these two records is an average of 9 successive

magnetic responses. These records appear to be very similar to the averaged leopard frog MAF of Figure 6.2. Only the amplitudes are noticeably different. A summary of the frog-to-frog variations of the MAF, magnetic stimulus artifact and twitch force variables is presented in Table 6.2. The variation in the MAF amplitude due to different muscle sizes was removed by dividing by the muscle mass.

6.2 EFFECT OF THE MUSCLE BATH MEDIUM

In this experiment, it was demonstrated that the electrical properties of the muscle bath medium can have a strong effect on the MAF amplitude. Two media were used: MacKenzie's saline (similar to Ringer's) and mineral oil (liquid paraffin). Superior WMAPs were obtained with the mineral oil medium because it is a good electrical insulator. The saline solution tended to short-circuit the action potential electrodes. The saline, however, provides the muscle with a more natural environment. Most important, to this research, was the fact that the MAF amplitude was greatly reduced when the mineral oil bath medium was used.

Figures 6.10, 6.11 and 6.12 show measurements made with the muscle bathed in MacKenzie's saline, then mineral oil and finally MacKenzie's saline again. Single MAF responses are shown in the top plot of each of these figures, while averages of 25 successive responses are shown in the bottom plots. From the averaged data, an MAF peak-to-peak amplitude of 14.8 pT was initially observed with the MacKenzie's

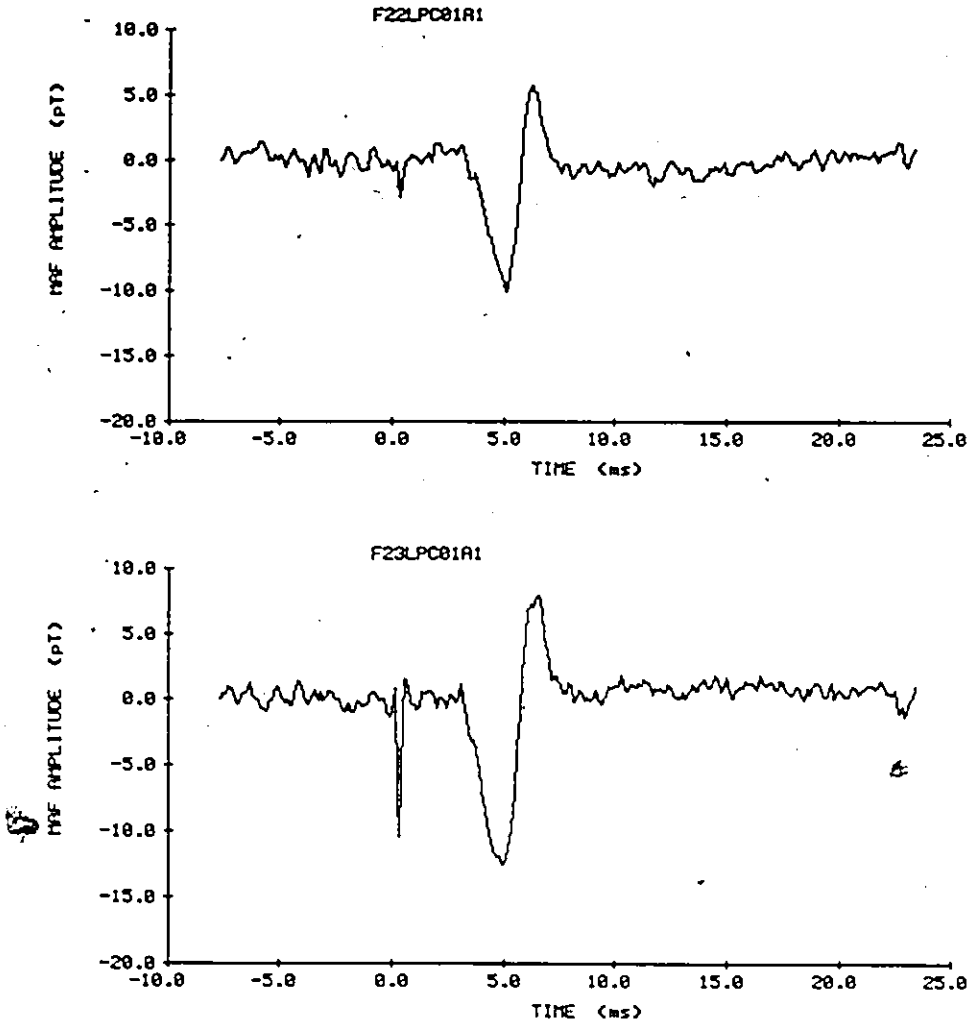


Figure 6.9 : Averaged MAF records from two bullfrog muscle preparations. These signals are very similar to the signal obtained from a leopard frog (see Figure 6.2).

saline bath medium (Figure 6.10). When changed to mineral oil, the amplitude immediately dropped by a factor of approximately 6.5 (Figure 6.11). When MacKenzies saline was reintroduced, the MAF amplitude immediately jumped back up (Figure 6.12).

MUSCLE SIGNAL VARIABLES		BULLFROG 1		BULLFROG 2		LEOPARD FROG		COMBINED DATA	
		MEAN	SD	MEAN	SD	MEAN	SD	MEAN	SD
MAF	AMPLITUDE (pT/g)	10.6	3.3	11.2	0.3	12.2	1.0	11.5	1.7
	DELAY (ms)	3.21	0.08	3.05	0	3.36	0.07	3.22	0.15
	DURATION (ms)	3.66	0.42	4.51	0.25	4.33	0.12	4.19	0.44
MSA	AMPLITUDE (pT)	6.4	4.5	15.1	3.5	27.9	0.5	17.6	9.9
	DELAY (ms)	0.12	0	0.12	0	0.12	0	0.12	0
	DURATION (ms)	0.77	0.19	0.85	0.13	0.95	0.06	0.87	0.14
FORCE	AMPLITUDE (mN/g)	-	-	324	8	93	1	192	124
	DELAY (ms)	-	-	10.7	0.9	13.2	0.8	12.2	1.5
	DURATION (ms)	-	-	58.0	2.6	20.4	0.04	36.5	20.2

Table 6.2 : Frog-to-frog variation (mean and standard deviation) in the MAF, magnetic stimulus artifact (MSA) and twitch force data. The "combined" statistics were calculated from the individual data values of the three frogs rather than from the mean values listed in this table.

The MAF amplitude did not return to its original value because of fatigue of the muscle. The muscle had been stimulated 232 times (over a period of an hour) by the time the data for the last record for Figure 6.12 was collected. Muscle twitch force amplitudes (not corrected for fatigue) have been listed with each of the plots of Figures 6.10, 6.11 and 6.12 to show that the muscle was just as active in the mineral oil as it was in the MacKenzie's saline. Note the gradual decline in force amplitude with muscle fatigue.

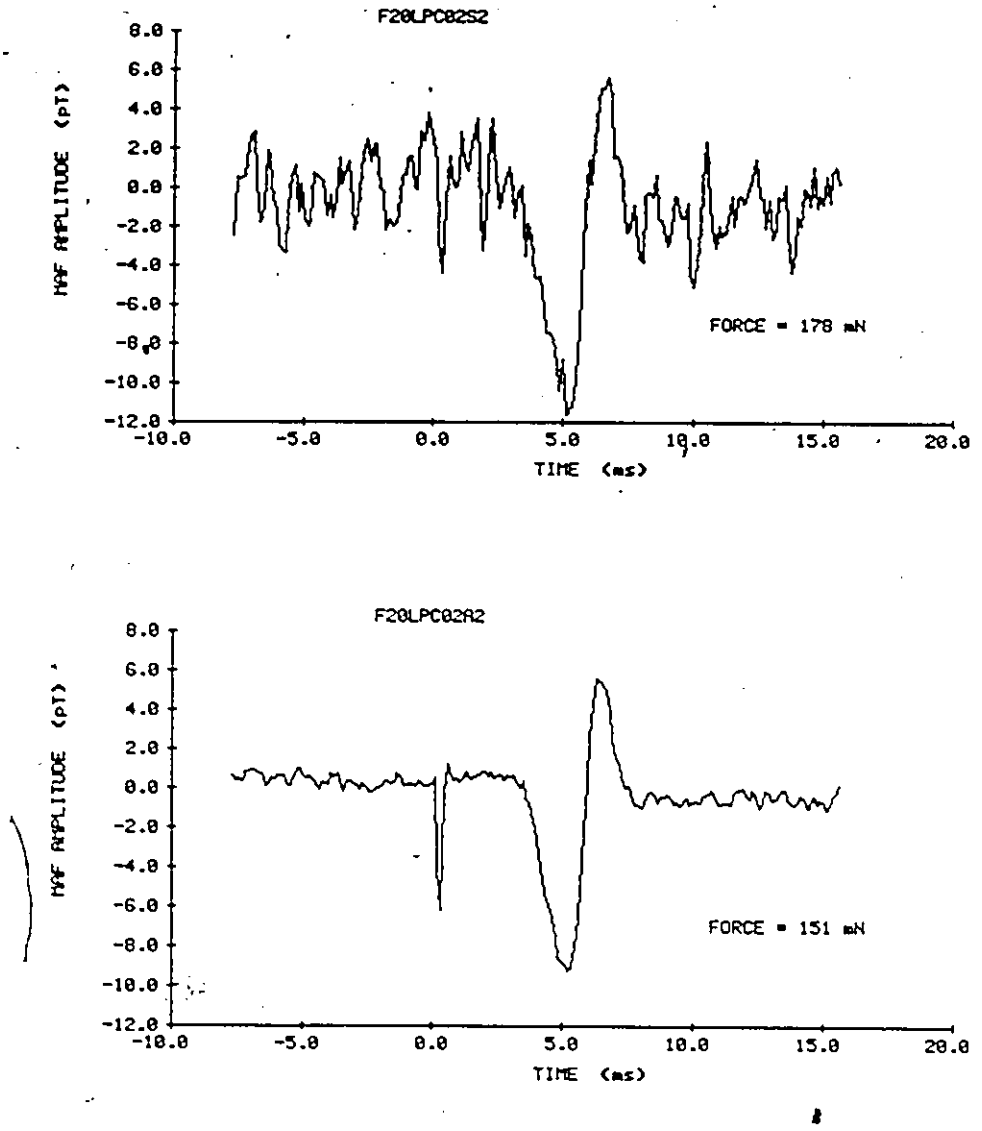


Figure 6.10 : Recordings of the MAF for a muscle preparation suspended in a MacKenzie's saline bath medium. A single response is shown in the upper plot and an average of 25 successive MAF's is shown in the lower plot.

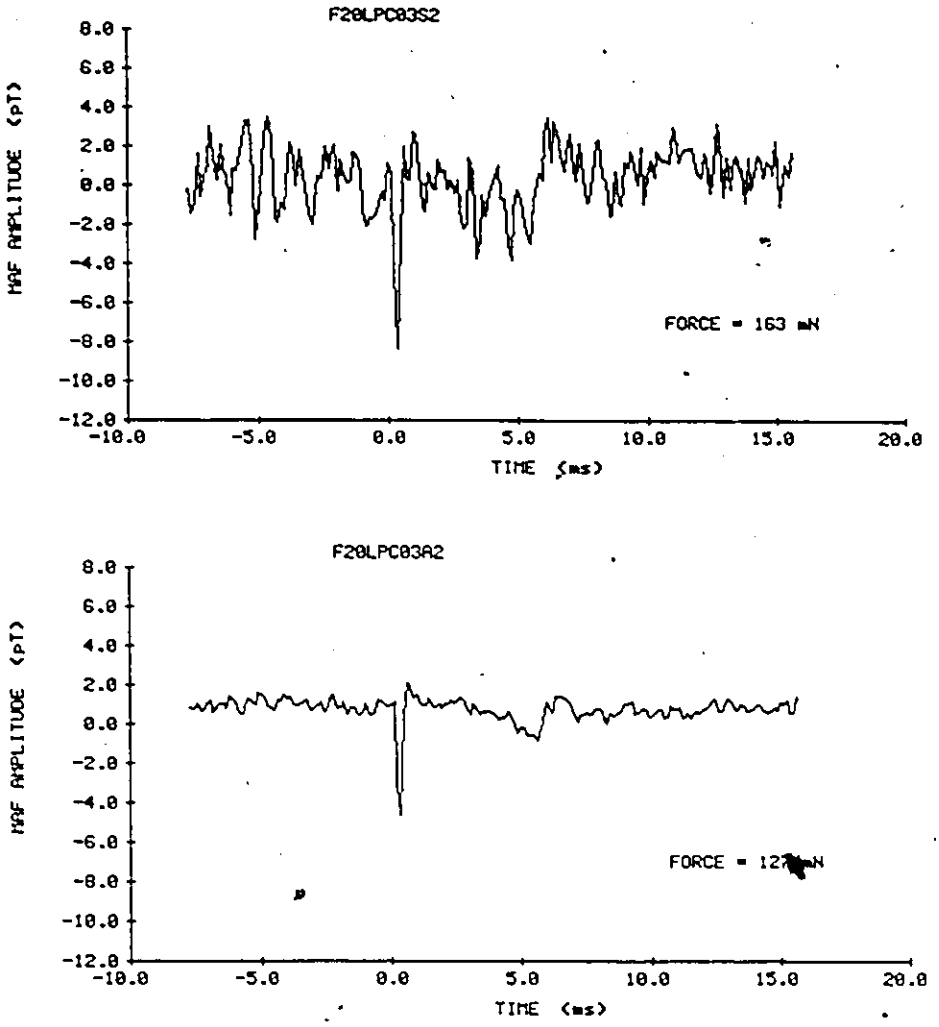


Figure 6.11 : Recordings of the MAF for the same muscle as for Figure 6.10, but with the MacKenzie's saline bath medium replaced with mineral oil. A single response is shown in the upper plot and an average of 25 successive MAF's is shown in the lower plot.

6.3 SPATIAL ASPECTS OF THE MAF

The first of the three spatial mappings was as shown in Figure 5.9a. Of all the variables that were studied, only the MAF

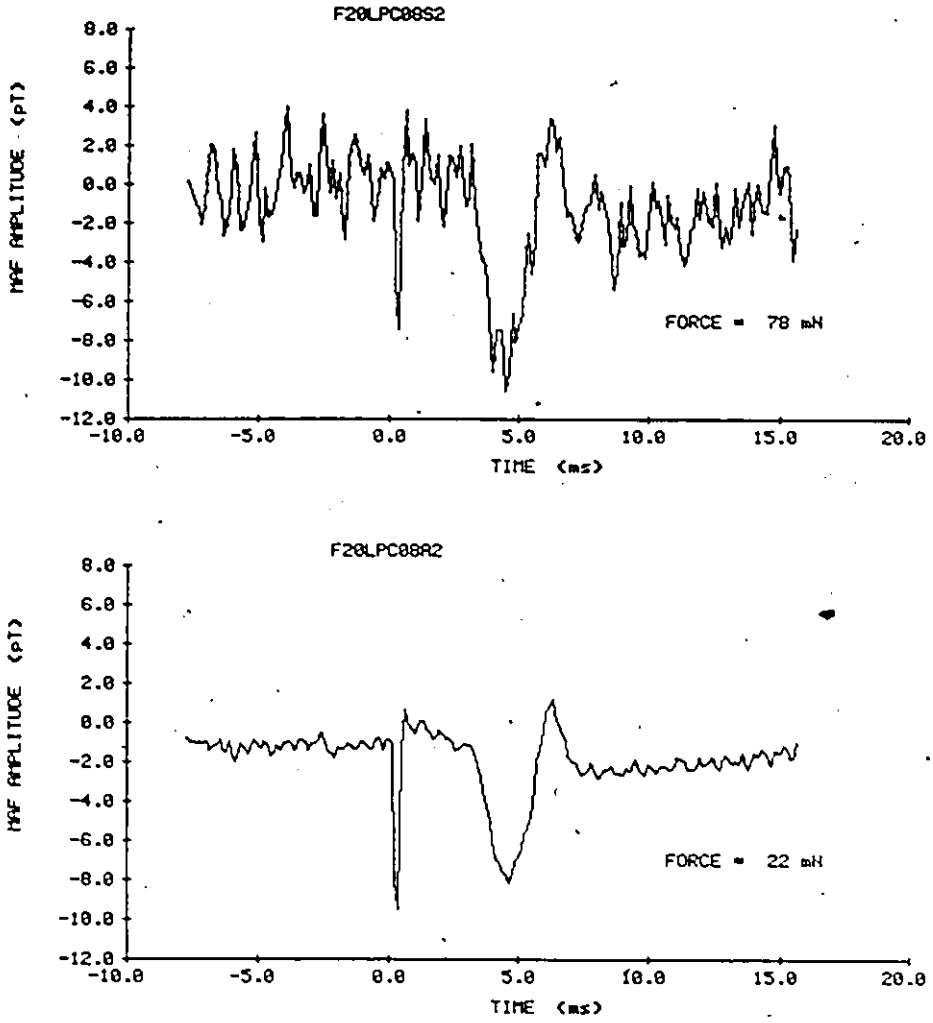


Figure 6.12 : Recordings of the MAF for the same muscle as for Figures 6.10 and 6.11, but with the bath medium switched back to the original MacKenzie's saline. A single response is shown in the upper plot and an average of 25 successive MAF's is shown in the lower plot.

amplitude had a definite and repeatable relation to position. The magnetic signal amplitude was a maximum when the muscle was directly under the detector coil windings. A plot of the magnetic signal amplitude

(corrected for muscle fatigue) is shown in Figure 6.13 as a function of position.

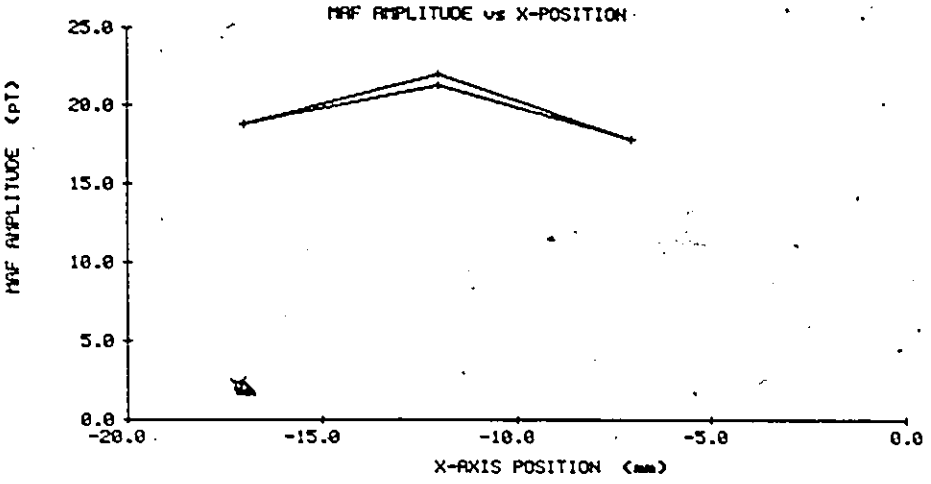


Figure 6.13 : Effect of the x direction position on the MAF amplitude.

For the second spatial mapping, as shown in Figure 5.9b, the muscle position was moved away from the detector coil plane in the -z direction. Both the stimulus artifact and MAF amplitude were found to decrease in amplitude. The stimulus artifact amplitude and MAF amplitude (corrected for muscle fatigue) have been plotted in Figure 6.14 as functions of distance from the magnetometer detector coil.

The final spatial mapping that yielded a detectable magnetic signal was as shown in Figure 5.9c. Once again, the maximum amplitude signal was obtained when the muscle was centered under the edge of the

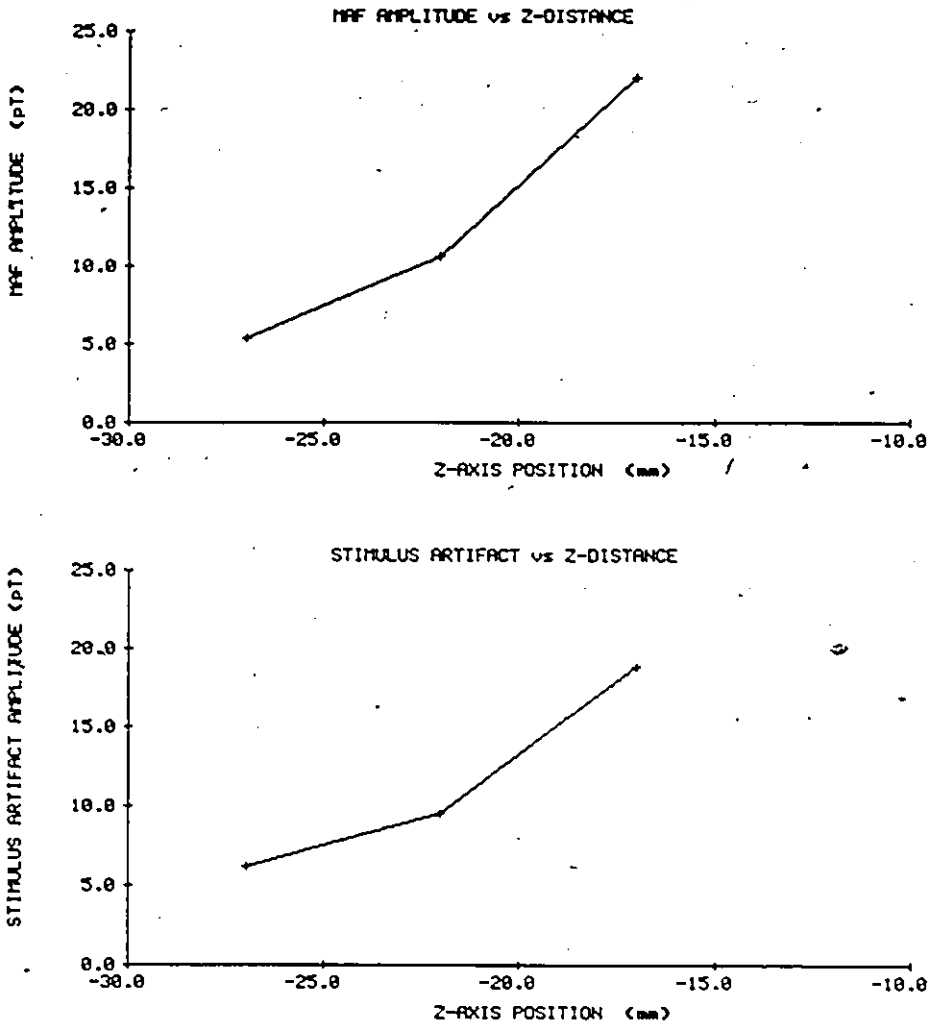


Figure 6.14 : Effect of the distance from the muscle to the magnetometer sensing coil (in the z direction) on the MAF and stimulus artifact amplitudes.

pick-up coil (see Figure 6.15). This maximum amplitude was only half the maximum amplitude of the first spatial mapping.

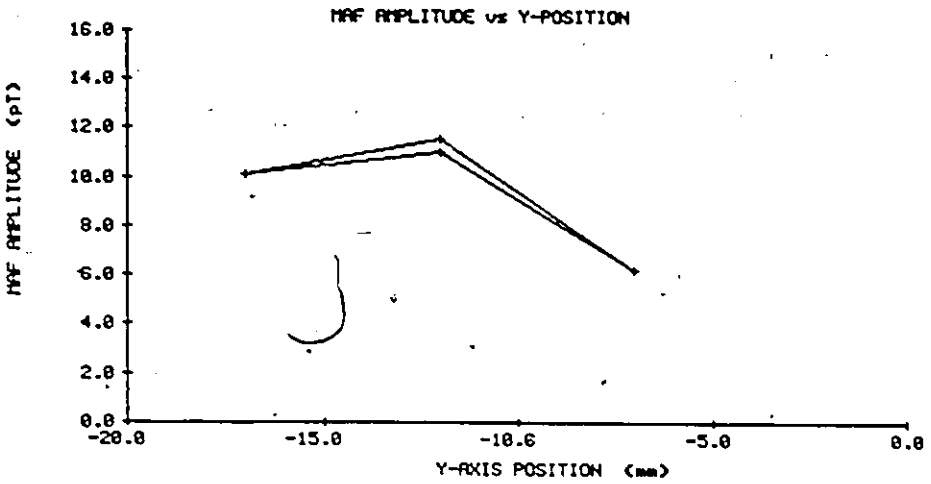


Figure 6.15 : Effect of the y direction position on the MAF amplitude.

6.4 MUSCLE LENGTH EXPERIMENT

Of all the variables that were calculated from the MAF and force data vectors for different muscle lengths, only the MAF and force amplitudes showed any definite trends. The MAF amplitude was found to decrease slightly with increasing muscle length. Figure 6.16 shows this dependence on muscle length. The twitch force amplitude was found to change greatly with the 5% and 10% changes in muscle length. With a 5% reduction from rest length, the muscle was unable to generate any force. As can be seen from Figure 6.16, a large MAF was detected for the reduced muscle lengths despite the absence of the twitch force. A 5% increase from rest length caused the force to more than double the initial 250 mN and go out of range of the analogue-to-digital converter. This same change in muscle length introduced a baseline tension of only 12 mN.

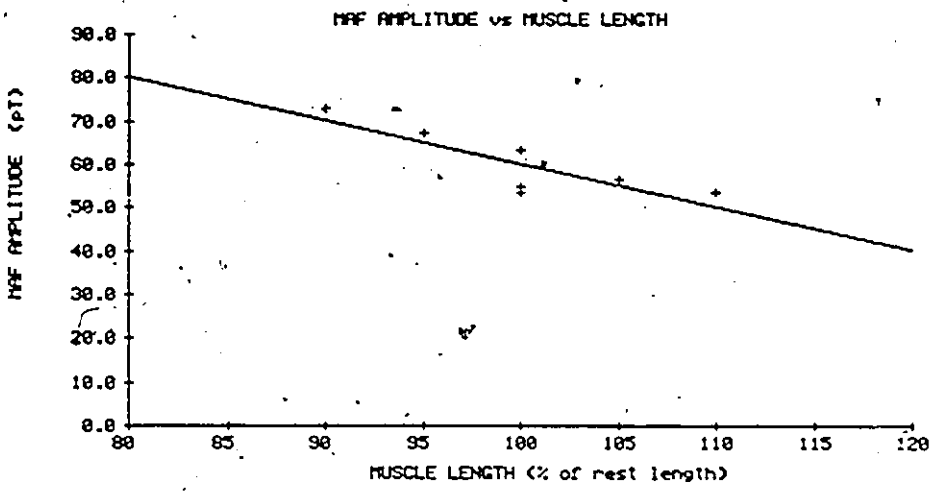


Figure 6.16 : The MAF amplitude as a function of muscle length.

6.5 STIMULUS AMPLITUDE EXPERIMENT

As expected, the MAF peak-to-peak amplitude was found to increase as the stimulus was increased toward the maximal stimulus. Figure 6.17 shows that the functional relationship is nonlinear.

It was also observed that the biphasic MAF response became increasingly offset as the stimulus was increased. That is, the initial phase of the magnetic signal developed an increasing amplitude and duration compared with the following phase. This was quantified by plotting (as a function of twitch force amplitude) the DC offset of the

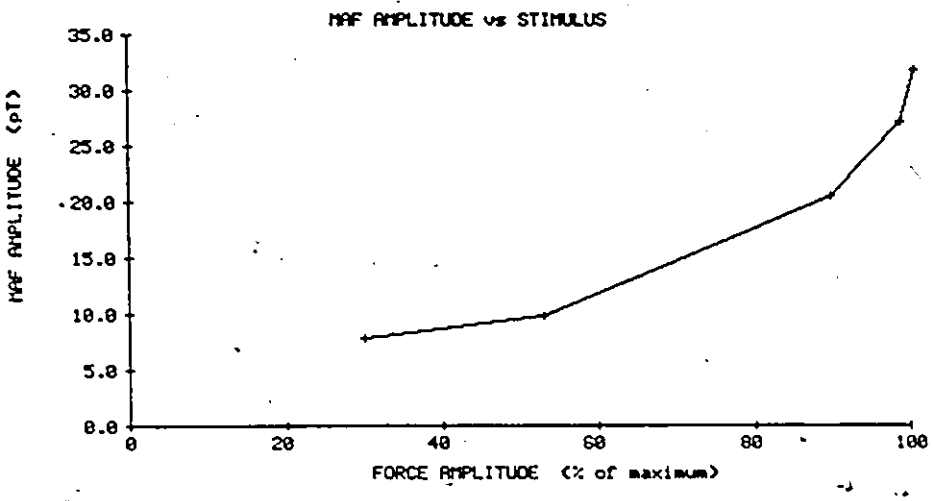


Figure 6.17 : The MAF amplitude as a function of twitch force amplitude (stimulus amplitude).

"windowed" MAF divided by its peak-to-peak amplitude. This is shown in Figure 6.18.

The time from when the stimulus was applied to when the peak twitch force was generated, called the 'Peak Force Delay', was found to decrease slightly with increasing stimulus. Figure 6.19 shows this gradual decline. All other variables failed to show any definite connection with the level of muscle activation (stimulus amplitude).

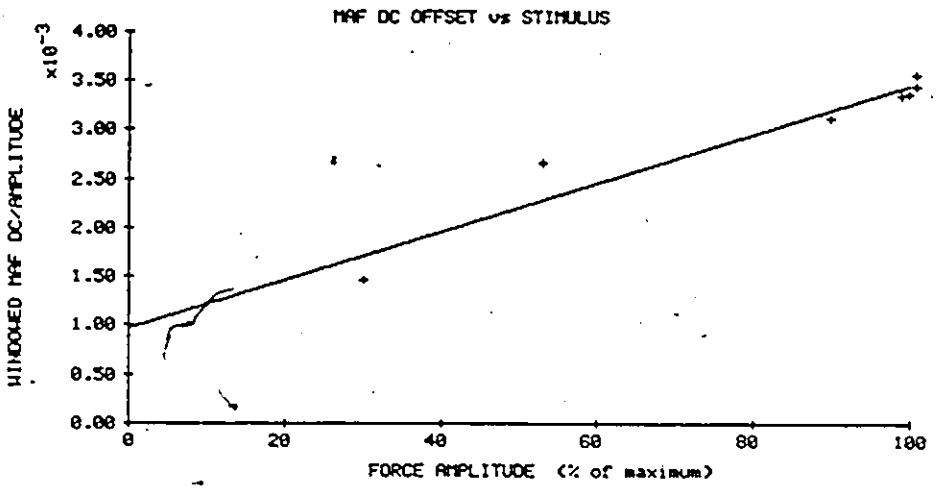


Figure 6.18 : The change in MAF DC-offset with different levels of muscle activation.

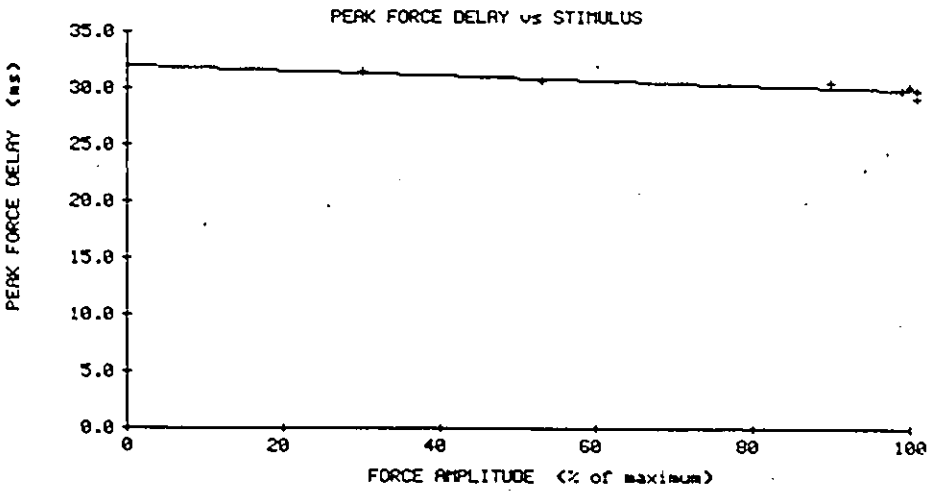


Figure 6.19 : The gradual decline in the "peak force delay" with increasing muscle activation. The peak force delay is the time from nerve stimulation until the twitch force reaches its maximim value. The twitch force amplitude was used as a measure of the level of activation.

CHAPTER 7

DISCUSSION

Presented in chapter 6, for the first time, was a set of measurements of the evoked magnetic action flux (MAF) of a whole excised skeletal muscle for a range of controlled physiological conditions. The frog gastrocnemius muscles were stimulated indirectly by single square voltage pulses applied to the sciatic nerve. It was the magnetic signal associated with the maximal isometric contractions that was measured. In chapter 6, the signal was described for optimal measurement conditions and statistics for a small sample of data records were presented. The electrical properties of the muscle bath medium were shown to have a strong effect on the measured MMG amplitude. Preliminary results of several physiological experiments were presented. Further experimentation will be necessary to confirm these early results.

7.1 THE MAF UNDER OPTIMAL CONDITIONS

The MAF from the frog gastrocnemius muscle was found to be a biphasic signal with a delay of 3.4 ms, a duration of 4.2 ms and an amplitude of 30 pT (see Table 6.1). The MAF was found to occur at the same time as the EMG and well before the mechanical response of the muscle (twitch force) was generated.

It might be thought possible that the magnetometer was not measuring a magnetic field, but was picking up the action potential by capacitive coupling. SQUID magnetometers are not sensitive to electric fields. The pickup coil of a SQUID magnetometer is part of a superconducting flux transformer. Because this circuit is superconducting, it has no resistance and cannot support a potential difference due to an external charge distribution.

Several observations confirm that the magnetometer signal could not have been a capacitively coupled action potential. First, the MAF was usually found to have a large stimulus artifact associated with it while the whole muscle action potential usually had none. Also, when the sciatic nerve was removed from the stimulating electrodes, both the muscle response and the stimulus artifact disappeared from the magnetometer signal. If the magnetometer signal were a capacitively coupled action potential, then the stimulus artifact should have remained because the stimulus pulse would still be present at the stimulating electrodes.

Could the measured magnetic signal be a motion artifact? This is not possible because the MAF was observed to be complete well before the muscle started to generate the twitch force. A small motion artifact was detected in a few of the magnetometer signals, but this occured after the MAF and at the same time as the muscle twitch.

The timing of the MAF and stimulus artifact in the magnetometer signal were as expected. The stimulus artifact was thought to come from a small source loop made up of the nerve stimulation electrodes, a short length of the leads to these electrodes (from where they separate from

being twisted together) and the 4 mm length of nerve bridging the two electrodes. Although the stimulation pulse was only 0.1 ms in duration, the 0.9 ms stimulus artifact was expected because of the 'smearing' effect of the low-pass second-order Butterworth filter on the magnetometer output (2048 Hz cutoff frequency). The 3.4 ms delay between the stimulus artifact and the onset of the MAF corresponds to the time taken for the action potential to propagate the 30 mm of sciatic nerve from the stimulation electrodes to the muscle. Propagation across the neuromuscular junction requires 1 ms. The remaining 2.4 ms corresponds to an action potential speed of 12.5 m/s. If we assume that the neuromuscular junctions are centered in the length of the muscle, then the 4.3 ms duration of the MAF corresponds to a propagation speed of 14mm/4.3ms = 3.3 m/s. This compares well with typical propagation speeds (Guyton, 1981; Carlson & Wilkie, 1974). The remaining delay of 5.5 ms until the twitch force started to appear was required for the calcium ions to diffuse from the sarcoplasmic reticulum and activate the contractile filaments (Gonzalez-Serratos, 1971; Vander, Sherman & Luciano, 1980).

Work of other researchers show similar results for isolated MMG pulses. A recording of a single motor unit action current from the extensor digitorum longus muscle of a rat showed a duration of approximately 3.0 ms (Gielen et al., 1986). Motor unit MMG pulses from the muscles that flex the human thumb were estimated to be 5 ms in duration (Cohen & Givler, 1972). For longer muscles such as the human brachialis, biceps and triceps, the estimate was 12 ms (Cohen & Givler, 1972). Level-triggered averaging isolated MMG pulses from the human tibialis anterior with durations of 10 to 20 ms (Koga & Nakamura, 1983). The

longer the muscle, the greater the duration of the MMG pulse and the lower the peak frequency of the power spectrum because it takes longer for the action potential to propagate the length of the muscle.

The dual current-dipole model of an action potential propagating along a cylindrical cell (presented in section 2.3) predicted a biphasic magnetic signal (Tripp, 1983; Plonsey, 1981). The MAF signals reported in section 6.1 conformed to the predictions of this model in that a biphasic signal was always observed. The measurements did not support the claim that the two current dipoles were of equal magnitude. The second phase of the MAF was consistently found to have a smaller amplitude and shorter duration than the initial phase. There are two possible reasons for this discrepancy. First, the summation of individual fibre MAF's to form the measured compound signal may have resulted in poor reinforcement of the second phase because of a variable delay between the two phases. Second, the structural complexity of a muscle fibre is not accounted for in the model of an ideal cylindrical cell.

Several of the variables of Table 6.2 were found to vary widely from one frog to the next. The large variations in force amplitude and duration were most likely due to the difference in species of the frogs. The great variation in the magnetic stimulus artifact amplitude was due to the fact that the stimulator provided a voltage pulse, and it was the resulting current that produced the magnetic field of the stimulus artifact. The current that flowed through the stimulation circuit depended on the resistance between the two stimulating electrodes. This in turn depended on the size of the nerve and the amount of connective tissue and saline solution surrounding the nerve, and should have varied

significantly from frog to frog. Also, since the nerve was wetted with saline several times during each experiment, the stimulus artifact amplitude was expected to vary between the records of a single experiment.

The EMG electrode design of Figure 5.7 was not well suited to concurrent biomagnetic measurements. If a good electrical contact was established between the two ends of the electrode wire when tying it around the muscle, a shorted turn was created. By Lenz's law, the shorted turn would have currents induced in it that would tend to cancel the axial component of the fluctuating magnetic field generated by the muscle (the azimuthal component would be unaffected). Thus, the presence of these electrodes might have significantly degraded the measured MAF. Since the saline muscle bath was found to short-circuit these electrodes, they were not used during the MAF measurements presented in this thesis. A new electrode design will be developed for future experiments.

7.2 EFFECT OF THE MUSCLE BATH MEDIUM

The 'bath-medium' experiment showed that the amplitude of the MAF was changed by a factor of approximately 6 when the bath medium was changed from McKenzie's saline to mineral oil (and back again). Since the effect was reversible (with no noticeable time delay), the mineral oil could not have affected the muscle chemically. The difference

between the two media is that the mineral oil is an electrical insulator and the saline solution is a good electrical conductor.

The conductivity of the saline bath medium was very close to that of the intracellular fluid. The boundary would have little effect on the volume currents and should not contribute significantly to the external magnetic field. By surrounding the muscle in an insulating medium, the volume currents are prevented from flowing external to the muscle. The muscle surface should then contribute significantly to the external field. In this experiment it appears that the muscle surface contribution cancelled the external field of the intracellular currents.

An alternate explanation involves the use of Ampere's law. The insulating medium confines the volume currents to a thin layer at the muscle surface. Due to the high degree of radial symmetry of the gastrocnemius and the fact that any circular path around the long axis of the muscle should have little net current passing through its cross-section, Ampere's law suggests that there should be little magnetic field outside the muscle.

7.3 SPATIAL ASPECTS OF THE MAF

Because of the large size of the dewar tail, it was difficult to get the muscle preparation close to the sensing coil of the magnetometer. Measurements could not be made with the muscle in some spatial positions and orientations. For example, the muscle could not be moved close enough to the sensing coil to detect the MAF when the muscle axis was coincident with the coil axis. It is possible that a magnetic flux

was present, but due to the symmetry no net flux linkage occurred.

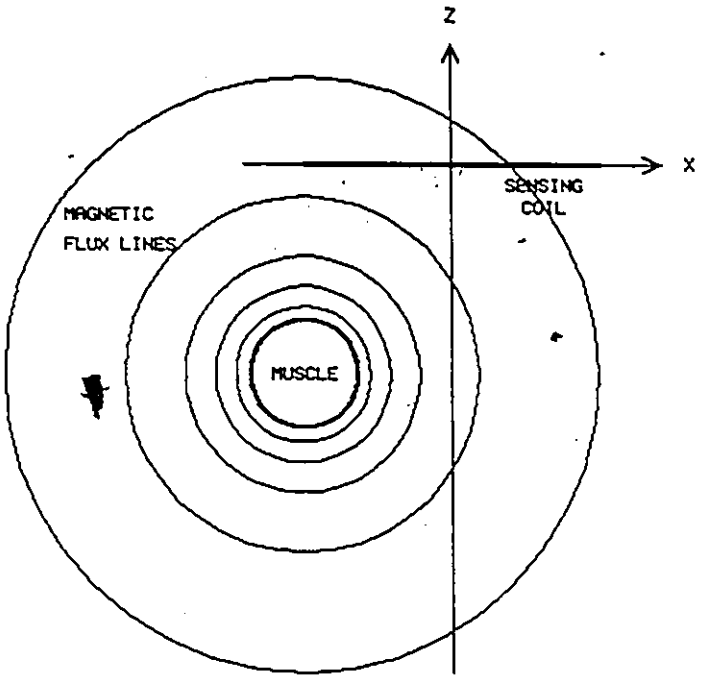


Figure 7.1 : Azimuthal magnetic flux lines surrounding the muscle. Note that if the muscle is moved to either side of its present position, fewer flux lines would link with the sensing coil.

The largest signal was observed when the muscle was parallel with the y-axis, but displaced in the x-direction (Figure 5.9a). The magnetometer detects the net flux that loops through its sensing coil. From the symmetry of the muscle, it is likely that the flux lines detected in this measurement were circular about the muscle axis as shown in Figure 7.1. This component of the MAF may be described as a biphasic azimuthal field. Such a field was predicted by the dual

current-dipole model presented in Chapter 2 for the propagation of an action potential along an isolated cylindrical fibre (Tripp, 1983). Measurements from isolated nerves have also confirmed this model's predictions (Wikswo, 1983).

The second spatial mapping (Figures 5.9b and 6.14a) showed that the magnetic flux density decreased as the distance between the muscle and sensing coil was increased. A comparison with the dual current-dipole model, which predicts that the field will fall off with the inverse cube of the distance (Tripp, 1981), will not be possible until more data have been collected and analysed.

The greater complexity of muscle structure, in comparison with an isolated cylindrical fibre, must be responsible for the presence of the 'end-to-end' magnetic field (not predicted for an isolated nerve). This field was detected during the spatial mapping of Figure 5.9c and was half the amplitude of the azimuthal field. Due to the symmetry of the recording configuration, nothing would have been recorded if the muscle's external field was purely azimuthal. The flux lines might be oriented as shown in Figure 7.2. The observed change in MAF amplitude with muscle position along the y-axis (Figure 6.15) supports this proposed flux distribution. The origin of this component of the external field is not known, but it may be due to the propagation of the action potentials through the transverse tubules of the muscle fibers.

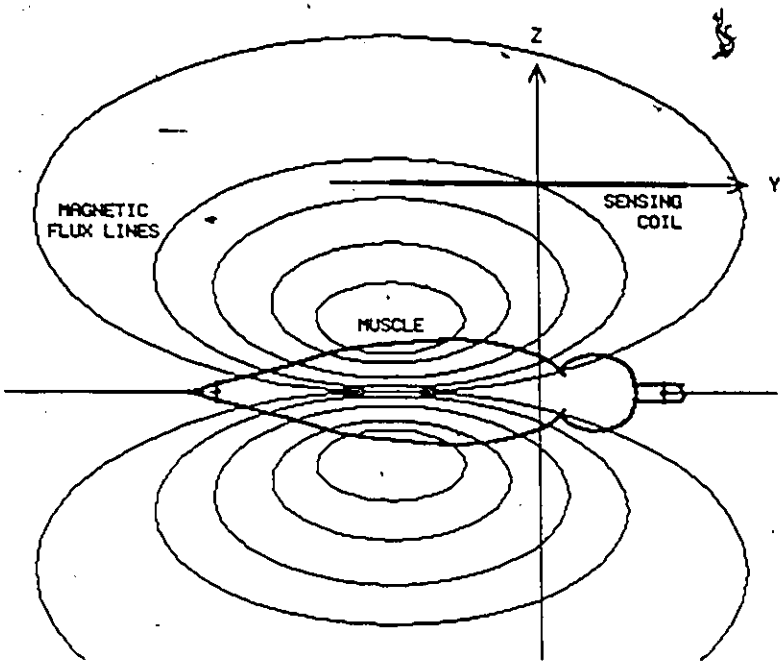


Figure 7.2 : The second component of the MAF with flux lines looping from end-to-end of the muscle. Note that fewer flux lines would link with the sensing coil if the muscle were moved to either side of its present position.

7.4 MUSCLE LENGTH EXPERIMENT

The MAF amplitude was found to decrease and the twitch force amplitude was found to increase sharply with increasing muscle length. Fulton (1925) reported that for the frog gastrocnemius, the size of the action current diminished with an increase in muscle length. The observed dependence of twitch force on muscle length is consistent with the work of Stevens, Dickinson & Jones (1980). Remember that the muscle rest length was defined as the maximum length for which the passive tension remained zero.

In the present research, part of the change in MAF amplitude may have been due to a change in position of the center of the muscle. The proximal end of the muscle was held fixed with respect to the magnetometer sensing coil. The distal end of the muscle was moved to cause the muscle to lengthen or shorten. By changing the length of a 30 mm muscle by 10%, the center of the muscle would have moved 1.5 mm with respect to the magnetometer. This effective movement of the muscle center could have caused a change in measured MMG amplitude.

From the dependence of MAF amplitude on position shown in Figures 6.13 and 6.15, it appears that the 1.5 mm change in position of the center of the muscle cannot explain the 16 % change in MAF amplitude seen in Figure 6.16. Part of the observed change in MAF amplitude must have been due to the change in muscle length. A more comprehensive set of measurements will be required to confirm this preliminary observation.

As for all of the experiments presented in this dissertation, it was difficult to find comparable experiments in the EMG literature. The early experimentation involving muscle mechanics and EMG measurements often employed multiple stimuli that brought the muscle into a tetanus. This was required because the instrumentation of the time did not have the frequency response to measure signals from single twitches without considerable distortion. Most of the more recent EMG work has been done on isolated fibres rather than on whole muscles. There is also an extensive clinical literature. Much of this work has involved measurements from muscles under voluntary control. These signals consist of the many peaks resulting from the asynchronous activity of the

individual motor units. An integrated EMG is usually presented rather than the individual action potentials. Due to this lack of comparable experiments in the EMG literature, it is difficult to evaluate the relative value of EMG and MMG measurements.

7.5 STIMULUS AMPLITUDE EXPERIMENT

The MAF peak-to-peak amplitude was found to increase nonlinearly with increasing muscle activation (Figure 6.17). A similar, but closer to linear, response was reported for action potentials from the frog sartorius (Watts, 1924). The fact that the fibres of the sartorius are more uniformly parallel than those of the gastrocnemius might explain the greater linearity of Watts' data.

Within the range of the data of Figure 6.17, an exponential function may be fitted to model the relationship between the MAF amplitude and the twitch force amplitude (stimulus amplitude). Plotted with the data, in Figure 7.3, is an exponential function of the form $y = a + cEXP(bx)$ where

x -> force amplitude (% of maximal contraction)

y -> MAF amplitude (pT)

EXP -> exponential function

a = 7.00 pT

b = 0.0453

c = 0.228 pT.

The parameter "a" was chosen by graphical means while the "b" and "c"

parameters were determined by a least squares error method. This function can only represent the data for a finite domain since the twitch force cannot go below 0% or above 100% of a maximal stimulation. Further measurements will be required to confirm this functional relationship.

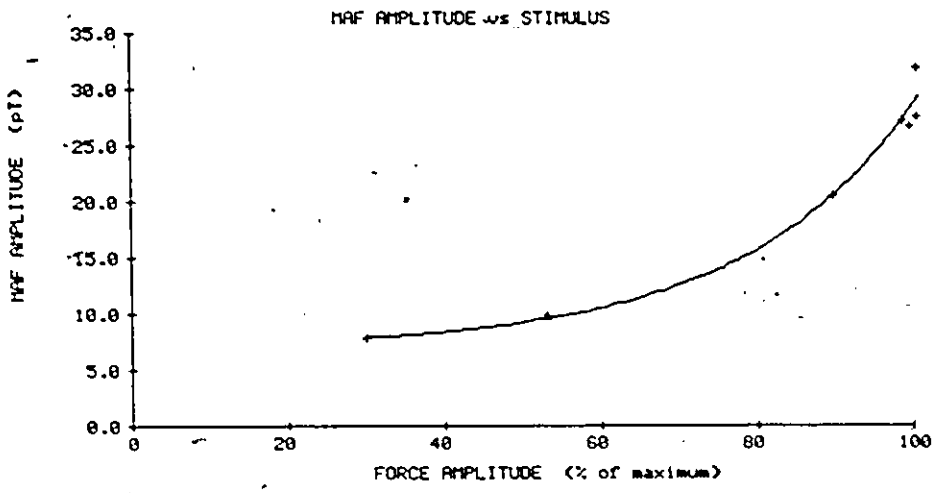


Figure 7.3 : MAF amplitude vs twitch force amplitude data fitted with an exponential function of the form: y = a + cEXP(bx).

Also, the "peak force delay" was found to decrease with increasing muscle activation. A possible reason for the "peak force delay" effect is that when the stimulus amplitude is increased, more motor units are recruited and the "slackness" in the elastic components of the muscle is taken up more quickly. The external force is consequently seen to rise more quickly.

Finally, the MAF DC offset was found to increase with increasing

muscle activation. As the sciatic nerve was stimulated with larger amplitude voltage pulses and the number of activated muscle fibres was increased, the initial phase of the biphasic MAF grew in amplitude and duration faster than the second phase. This could be explained by a distribution in the delay of the repolarization currents (that follow the leading depolarization of an action potential) throughout the fibres of the muscle. ~~The~~ depolarization currents of all the activated muscle fibers should be well synchronized. Only the small variation in the position of the motor end plates should cause "jitter" in the timing of the depolarization currents. The magnetic fields of these depolarization currents should superpose to yield an initial phase of the MAF of greater amplitude as the number of active muscle fibers is increased. If there is a significant variation in the delay until the repolarization occurs, then the magnetic fields of the repolarization phase of the MAF will not reinforce themselves as well as for the depolarization phase, and the second phase of the MAF will not grow in amplitude as fast as the initial phase. Thus, as the number of activated muscle fibers is increased, the amplitude of the initial phase of the MAF should increase faster than the amplitude of the second phase and a DC offset will result. The variation in delay of the repolarization currents may be due to distribution in muscle fibre diameters or metabolic variations.

7.6 RESEARCH OBJECTIVES

The long range objective of this research was to evaluate the potential of the MMG as a clinical and research tool. Rather than directly investigate the MMG of in vivo muscles, the compound MAF of an excised muscle was to be studied. It was felt that by developing an understanding of the physiological basis of the MAF, future investigation of the MMG would be much more fruitful. The objective of the research reported in this dissertation was therefore to perform a set of experiments that would determine the spatial and temporal characteristics of the MAF and the effect of several physiological variables upon these characteristics.

A data acquisition system involving analogue and digital hardware, computer software and a muscle environment apparatus was successfuly designed, built and debugged. Experimental procedures were also developed to allow one independent physiological variable to change, while all others were held constant at desired values. With this system, a preliminary set of measurements were made and analysed. Due to an external constraint (the SQUID magnetometer unexpectedly became inaccessible) each experiment was not repeated enough times to allow the formation of firm conclusions. Thus, the objective of determining the complete spatial and temporal characteristics of the MAF and the effect of the physiological variables upon these characteristics was only partially fulfilled.

In terms of the overall objective of determining the clinical and research potential of the MMG, substantial progress has been made.

It has been demonstrated that the magnetic signal from an excised frog muscle is reproducible and can be measured with a commercially available magnetometer. The MAF was shown to be affected by the stimulus amplitude, the muscle length and the conductivity of the surrounding medium. These experimental results support, in a general way, the expectation of being able to use the MMG to characterize the state of an active skeletal muscle.

Future work will, of course, involve an extensive set of measurements from a variety of muscles (in several species) to confirm the initial findings reported here. A magnetometer with improved spatial resolution will be used for this new set of measurements. Source modelling to explain the "end-to-end" field of the muscle will also be a high priority. This will involve consideration of the transverse tubules of skeletal muscle. Finally, instrumentation will be developed to directly measure the magnetic field vector.

CHAPTER 8

CONCLUSION

In this research, the compound magnetic action flux (MAF) was measured from the frog gastrocnemius for indirectly stimulated isometric twitches. The temporal and spatial characteristics of the MAF were investigated. The effects of several physiological variables on the MAF characteristics were also studied.

The experimental measurements showed that the muscle generated a biphasic azimuthal magnetic field as predicted by electromagnetic modelling of an ideal cylindrical cell in an homogeneous medium. A second component of the external magnetic field, not predicted by the model, was found to loop from end-to-end of the muscle. For submaximal stimulation, the MAF amplitude was found to increase nonlinearly with the twitch force amplitude. For supramaximal stimulation, the MAF amplitude was found to decrease with increasing muscle length. Finally, it was found that the electrical conductivity of the bath medium strongly affected the amplitude of the MAF signal. This was attributed to a change in the contribution from the boundary between the muscle tissue and the surrounding medium. This implies that the in vivo environment of a muscle may affect the observed signal.

These preliminary results are important for the following reasons. They have demonstrated that the MAF is a reproducible signal that may be measured in a "noisy" laboratory environment with a commercially

available SQUID magnetometer. They have also shown that the MAF is sensitive to some physiological variables and may therefore be of diagnostic value. The MAF was seen to follow a similar time course to the conventional EMG. Although the MAF is similar to the EMG in many ways, it is not yet known whether measurements of one component of the MAF can provide any information not already available in the EMG. Biomagnetic modelling has shown that the magnetic flux density and electric potential may be independent in some situations but not in others. Since skeletal muscle has not yet been adequately modelled, this issue remains unresolved.

The purpose of this research was to investigate the characteristics of the MAF and to study the effect of several physiological variables upon these characteristics. It was felt that this would be the most effective first step toward the overall objective of evaluating the potential of the magnetomyogram (MMG) as a research and clinical tool. Since the MAF is the basic component of the more complex MMG, determination of MAF characteristics should greatly aid in investigation of the MMG. This research was successful in providing a preliminary set of results that have determined some MAF characteristics. Measurements from a larger sample of muscle preparations will be required to confirm these initial results. The measurements presented in this dissertation are important because no other investigation of the effects of physiological variables on MAF or MMG characteristics has been reported in the literature.

The MMG has been neglected by biomagnetic researchers because it is thought to offer no more information than the EMG and is more difficult to measure. Investigation of the MMG was motivated by several arguments. First, biological tissues are more transparent to magnetic fields than to electric fields. Deep sources, not "visible" to surface EMG measurements, should yield measureable MMG signals. Some experimental results reported in the literature support this claim. Second, MMG measurements require no physical contact with the subject. The combination of the transparency of the body to magnetic fields and the freedom to position the sensing coil anywhere external to the body allows the opportunity for performing spatial mappings of the magnetic field and application of tomographic and imaging techniques. Determination of the magnetic vector should also be easier than determination of the electric field vector since EMG measurements can only detect components of the electric field parallel to the skin surface. MMG vector measurements might offer more information than EMG scalar measurements. From these arguments and the success of the preliminary investigation of the MAF reported here, the hypothesis remains: the MMG has the potential to become a useful and practical tool for the research and diagnosis of neuromuscular diseases.

Future research will proceed in several directions. First, the preliminary results presented in this dissertation must be confirmed by a set of measurements from a larger sample of muscle preparations. Second, some biomagnetic modelling specific to the structure of skeletal muscle must be developed in order to fully explain the magnetic field observed in this research. Finally, instrumentation will be developed

to allow direct recording of the magnetic flux density vector as a function of time and position.

APPENDIX A

DATA ACQUISITION SOURCE CODE

Presented in this appendix is the C programming language source code for the experiment control and data acquisition programme SMDA (SQUID Magnetometer Data Acquisition). Many C language subroutine calls and HP-UX (UNIX) system calls appear in the following listings. It is assumed that usage of these calls will be familiar to other programmers or can be determined by reference to standard C and UNIX manuals. Included in these listings are all the "user defined" subroutines required by SMDA. Except for SMDA "main" and "quit" the subroutines are presented in alphabetical order in the following pages:

smda: main
quit
acquire
acquire1
acquire3s2a
beep
fkey_lbl
fkey_scan
length_exp
maxforce
muscle_dat
pause_continue
pos_exp
repeat_complete
save
sequence
set_up
single_save
stim_exp.

```
/*****************************************************************************
*
*       smda.c
*               ->  SQUID Magnetomer Data Acquisition programme (in C).
*
*               This programme  collects data  from the  DSQ-400 SQUID
*       magnetometer, an EMG amplifier and a pressure transducer and
*       saves it on the internal disc drive.  It also provides the instruc-
*       tions for the three experimental procedures used to study the MMG
*       at Erindale College:
*               1 -  Position Experiment (spatial dependence of the MMG)
*               2 -  Stimulus Experiment (vary stimulus amplitude)
*               3 -  Length Experiment  (vary the muscle length).
*
*       The data is stored on discs called /SMFDnn (SQUID Magnetometer Frog
*               Data nn), where nn is a decimal integer.
*
*       The file names are formed from: '
*               - Fnn  where nn is the frog number (2-digit integer).
*               - R or L to identify the right or left leg of the frog.
*               - P, L or S  which are codes for the 'position', 'length'
*                       and 'stimulus' experiments respectively.
*               - Cnn  where nn is a condition code that determines the
*                       experimental conditions for that measurement.
*               - S1, S2, A1, A2, or S3  which are record type specifiers.
*                       - S1, S2 and S3 contain single records.
*                       - A1 contains the average of 9 records.
*                       - A2 contains the average of 25 records.
*                               (including the A1 records)
*               - an example file name is  'F25RPC13S1'.
*               - a full path name would be '/SMFD07/F25RPC13S1'.
*
*       Variables stored in the data files are:
*               n  - the number of values in the data vectors.
*               buffer  - 64+4n bytes of character type data consisting
*                     alternately of:
*                           8-BIT offset binary EMG,
*                           8-BIT unsigned binary Force
*                           and 16-BIT binary MMG integers.
*                     (this buffer is declared with 64 extra bytes of space
*                      for the storage of the following variables)
*               r  - the data rate -- # of samples/s.
*               sn  - stimulus number -- number of stimuli since disection.
*               rt  - relative time -- time (in seconds) since
*                                               the beginning of disection.
*               x  |
*               y  |- position of muscle center w.r.t. magnetometer (mm).
*               z  |
*               ms  - muscle stimulus (% of maximum stimulus amplitude).
*               ml  - muscle length (% of resting muscle length).
*               mmgfc - 3dB cutoff frequency of digital resample filter for MMG.
*               spec  - species of disected frog.
*                     = 0 for Rana pipiens pipiens
*                     = 1 for Rana catesbeiana
*                     = 2 for 'other'.
*               sex  = 0 for MALE.  = 1 for FEMALE.
*               leg  = 0 for RIGHT.  = 1 for LEFT.
*               emgsf  - scale factor for converting emg integer to microvolts.
*               forcesf  - scale factor for converting force integer to
*                                                               microNewtons.
*               msa  - muscle stimulus amplitude (mv).
*               msd  - muscle stimulus duration (microseconds).
*               fw  - frog weight (in 10's of milligrams).
*               fl  - frog length from nose to crotch (mm).
*               mw  - gastroc muscle weight (in 10's of milligrams).
*               mrl  - resting muscle length (mm).
*               mbtmp  - muscle bath temperature (in 1/10ths of degrees Celcius)
*
```

```
*       Programme control is handled with the special function keys.
*
*       Copy this file to /tmp and compile to generate object file using
*                       /cpp/bin/fcc -c smda.c     or     cc -c smda.c
*
*       To link with the other required object files, copy only the required
*               object files to /tmp and use .
*                       /cpp/bin/fcc *.o -ldvio      or     cc *.o -ldvio
*
*               -> included in *.o must be
*                       acquire.c, acquire1.c, acquire3s2a.c, beep.c,
*                       continue.c, disc_save.c, fkey_lbl.c, fkey_scan.c,
*                       length_exp.c, maxforce.c, pos_exp.c, repeat.c,
*                       sequence.c, set_up.c, stim_exp.c.
*
*************************************************************************/

#define N_INIT  2048    /* number of samples to be acquired */
#define R_INIT  8192    /* data acquisition rate (samples/second) */
#define MMGFC_INIT  2048        /* cut-off frequency of MMG filter (Hz) */
#define BUF_SIZE  64+4*N_INIT

#include <stdio.h>
#include <fcntl.h>
#include <sys/beep.h>
#include <sys/ioctl.h>

char path[19], *klbl[9], *buffer;
int n=N_INIT, r=R_INIT, mmgfc=MMGFC_INIT;
int sn, rt, x, y, z, ms, ml, spec, sex, leg;
int xs0, ys0, zs0, fn, cn, dn;
long dt, ct;
int emgsf, forcesf, msa100, msa, msd, fw, fl, mw, mrl, mbtmp;
int sp_fd = -1, fkey, force100;
float ls0;

main()
{
   char *calloc();
   int i;

      /* allocate memory for character buffer */
   if ( (buffer = calloc(BUF_SIZE, 1)) == 0)
   {
      printf ("\nMemory allocation failure !\nClear some RAM?\n");
      exit(1);
   }

      /* initialize the data file name */
   for (i=0; i<18; i++)
      path[i] = ' ';
   path[18] = '\0';

      /* Open Speaker Device File */
   if ( (sp_fd = open ("/dev/beeper", O_RDWR)) == -1)
   {
      printf ("\nThe speaker device file did not open properly!\n");
      exit (1);
   }

      /* print start-up message */
   printf ("\n*************************");
   printf ("*************************************************\n");
   printf ("\n\tsmda\n");
   printf ("\n\t\tSQUID Magnetometer Data Acquisition Programme\n");
   printf ("\n\tThis programme collects data from the DSQ-400 SQUID");
   printf (" magnetometer,\nan EMG amplifier and a pressure transducer and");
   printf (" saves it on the internal\ndisc drive.  It also provides the");
```

```
printf (" instructions for the three experimental\nprocedures used to");
printf (" study the MMG at Erindale College:\n");
printf ("\t1 -  Position Experiment (spatial dependence of the MMG)\n");
printf ("\t2 -  Stimulus Experiment (vary stimulus amplitude)\n");
printf ("\t3 -  Length Experiment  (vary the muscle length).\n");
printf ("\nProgramme activity is controlled with the special function ");
printf ("keys.\n\n**************************");
printf ("************************************************************\n");
printf ("\n( press [CONTINUE] )\n");
pause_continue ();

printf ("\n\n*************************************************");
printf ("*********************************\n");
printf ("\n\t\tDSQ-400 SETTINGS\n");
printf ("\nMode :    Run TP \n");
printf ("DAC Filter :  1k (or OUT) \n");
printf ("DAC Output :  1/4 \n");
printf ("Digital Filter :  2k \n");
printf ("Sample Rate :  8k \n");
printf ("Status :  I.D. Code :  1110  (#14) (SW5-1,2,3,4) \n");
printf ("          Bytes Out :  0111  (SW4-1,2,3,4) \n");
printf ("          Fltr/Dac  :  10XX  (2-pole But.) (SW1-1)\n");
printf ("\n*************************************************");
printf ("*********************************\n");
printf ("\nCheck these settings carefully!\n");
printf ("\n*************************************************");
printf ("*********************************\n");
printf ("\n( press [CONTINUE] )\n");
pause_continue ();

printf ("\n\nSet up all equipment (with muscle apparatus in the shielded ");
printf ("room) \nand power up to allow thermal settling.\n");
printf ("\n\nSelect one of the special function keys labeled below:\n");

  /* main control loop of ida.c */

while (1)
{
     /* define the special function key labels */
   klbl[1] = "         SET-UP ";
   klbl[2] = "POSITIONexperimnt";
   klbl[3] = " LENGTH experimnt";
   klbl[4] = "STIMULUSexperimnt";
   klbl[5] = "  NEW    MUSCLE ";
   klbl[6] = " SINGLE  RECORD ";
   klbl[7] = "                ";
   klbl[8] = "          EXIT  ";

   fkey_lbl ();      /* write the key labels to the alpha window */
   printf ("\n**************************");
   printf ("************************************************************\n");

   fkey = fkey_scan();       /* receive function key input */
   write (1, "\33&j@", 4);   /* turn off menu */

   switch (fkey)
   {
      case 1 :
         set_up();
         break;
      case 2 :
         pos_exp();
         break;
      case 3 :
         length_exp();
         break;
      case 4 :
         stim_exp();
```

```
            break;
         case 5 :
            muscle_dat();
            break;
         case 6 :
            single_save();
            break;
         case 7 :
            break;
         case 8 :
            quit();
            break;
      }
   }
}
/*****************************************************************************/

quit()
{
   write (1, "\15\12Do you really want to quit ?!", 31);

     /* define the special function key labels */
   klbl[1] = "       YES   ";
   klbl[2] = "             ";
   klbl[3] = "             ";
   klbl[4] = "          NO ";
   klbl[5] = "             ";
   klbl[6] = "             ";
   klbl[7] = "             ";
   klbl[8] = "             ";

   fkey_lbl (); /* write the key labels to the alpha window */

   while (1)
   {
      fkey = fkey_scan();       /* receive function key input */
      if (fkey == 4)
         return;
      if (fkey == 1)
      {
         cfree (buffer);
         printf ("\nProgramme smda terminated.\n");
         printf ("\nTo restart this programme, high-light ");
         printf ("'smda' in the PAM window\nand press [Start].\n");
         printf ("\n*******************************");
         printf ("*************************************************\n");

         klbl[1] = "             ";
         klbl[4] = "             ";
         fkey_lbl ();   /* write the key labels to the alpha window */

         exit (0);
      }
   }
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       acquire.c       ->      data acquisition routine
*
*               -> this function collects 'n' samples of data from the DSQ-400
*                       SQUID magnetometer via the DIL and the HP-IB.
*
*       -> copy this file to /tmp and compile to generate object file using
*               /cpp/bin/fcc -c acquire.c       or      cc -c acquire.c
*
*       -> to link with other object files, use
*               /cpp/bin/fcc other_files.o acquire.o -ldvio      or      cc...
*
*****************************************************************************/

#include <stdio.h>
#include <fcntl.h>
#define INTER 0xE40023

acquire ()
{
   extern char *buffer;
   extern int n, sn;
   register unsigned char *clock_reg;
   int n_read,eid,MLA;
   char command[4];

      /* open HP-IB for read/write access */
   if ((eid = open("/dev/dhpib.i", O_RDWR)) == -1)
   {
      printf("\nFailure to open /dev/dhpib.i\n");
      printf("\nDoes 'load_hpib' need to be run?\n");
      return;
   }

   sn++;        /* increment the stimulus number */

      /* configure the interface for data acquisition */
   MLA=hpib_bus_status (eid, 7) + 32;   /* determine IPC listen address */
   command[0]=95;               /* UNTALK command */
   command[1]=63;               /* UNLISTEN command */
   command[2]=64+25;            /* talk address for device 25 on HP-IB */
   command[3]=MLA;              /* IPC listen address */

   hpib_send_cmnd(eid,command,4);
   io_burst(eid,1);     /* put interface in high-speed 'burst' mode */

   clock_reg = (unsigned char *) INTER; /* disable beat interupts */
   *clock_reg = (unsigned char) 0x80;

   n_read = read (eid, buffer, 4*n);    /* perform data acquisition */

   clock_reg = (unsigned char *) INTER; /* re-enable beat interupts */
   *clock_reg = (unsigned char) 2;

   io_burst(eid,0);     /* return from 'burst' mode */
   hpib_send_cmnd (eid, command, 1);    /* send 'untalk' command */

   if (n_read != 4*n)
   {
      printf("\nData acquisition incomplete!\nOnly %d bytes ", n_read);
      printf("of the requested %d bytes were read.\n", 4*n);
   }

   close (eid);
   return;
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       acquire1.c      ->      data acquisition routine
*
*               -> this function collects one sample of data from the DSQ-400
*                       SQUID magnetometer via the DIL and the HP-IB.
*
*       -> copy this file to /tmp and compile to generate object file using
*               /cpp/bin/fcc -c acquire.c       or      cc -c acquire.c
*
*       -> to link with other object files, use
*               /cpp/bin/fcc other_files.o acquire.o -ldvio      or      cc,..
*
*****************************************************************************/

#include <stdio.h>
#include <fcntl.h>
#define INTER 0xE40023

acquire1 ()
{
   extern char *buffer;
   extern int n;
   register unsigned char *clock_reg;
   int n_read,eid,MLA;
   char command[4];

      /* open HP-IB for read/write access */
   if ((eid = open("/dev/dhpib.i", O_RDWR)) == -1)
   {
      printf("\nFailure to open /dev/dhpib.i\n");
      printf("\nDoes 'load_hpib' need to be run?\n");
      return;
   }

      /* configure the interface for data acquisition */
   MLA=hpib_bus_status (eid, 7) + 32;   /* determine IPC listen address */
   command[0]=95;                       /* UNTALK command */
   command[1]=63;                       /* UNLISTEN command */
   command[2]=64+25;                    /* talk address for device 25 on HP-IB */
   command[3]=MLA;                      /* IPC listen address */
   hpib_send_cmnd(eid,command,4);

   io_burst(eid,1);     /* put interface in high-speed 'burst' mode */

   clock_reg = (unsigned char *) INTER; /* disable beat interupts */
   *clock_reg = (unsigned char) 0x80;

   n_read = read (eid, buffer, 4);   /* perform data acquisition */

   clock_reg = (unsigned char *) INTER; /* re-enable beat interupts */
   *clock_reg = (unsigned char) 2;

   io_burst(eid,0);     /* return from 'burst' mode */

   hpib_send_cmnd (eid, command, 1);    /* send 'untalk' command */

   if (n_read != 4)
   {
      printf("\nData acquisition incomplete!\nOnly %d bytes ", n_read);
      printf("of the requested %d bytes were read.\n", 4*n);
   }

   close (eid);

   return;
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*     acquire3s2a.c
*
*     This programme block looks after the collection and storage of data
*          in five disc files:
*                         S1 - single data record.
*                         S2 - single data record.
*                         A1 - 9 data records averaged.
*                         A2 - 25 data records averaged.
*                         S3 - single data record.
*
*     -> copy this file to /tmp and compile to gererate an object file using
*          /cpp/bin/fcc -c acquire3s2a.c      or      cc -c acquire3s2a.c
*
*     -> to link with other required object files, use
*          /cpp/bin/fcc files.o acquire3s2a.o    or    cc files.o acquire3s2a.o
*
*****************************************************************************/

#include <stdio.h>

acquire3s2a()
{
   extern char path[], *buffer;
   extern int n;

   char *calloc();
   int i, j;
   int *emgav, *forceav, *mmgav;

      /* Allocate memory for signal averaging */
   emgav = (int *) calloc(n, sizeof(int));
   forceav = (int *) calloc(n, sizeof(int));
   mmgav = (int *) calloc(n, sizeof(int));
   if (emgav==0 || forceav==0 || mmgav==0)
   {
      printf ("\nMemory allocation failure in 'acquire3s2a'!\n");
      printf ("Clear some memory and press [REPEAT].\n");
      return;
   }

      /* Collect the data for the 3 disc files */
   path[16] = 'S';
   path[17] = '1';
   beep (880, 25);
   acquire ();
   save ();

   path[17] = '2';
   beep (880, 25);
   acquire ();
   save ();

   path[16] = 'A';
   path[17] = '1';
   for (i=0; i<9; i++)
   {
      beep (880, 25);
      acquire ();
      sleep (2);
      for (j=0; j<n; j++)
      {
         *(emgav + j) += (((int) *(buffer + 4*j)) & 0xFF);
         *(forceav + j) += (((int) *(buffer + 1 + 4*j)) & 0xFF);
         *(mmgav + j) += (((int) *(buffer + 2 + 4*j)) << 8) |
                              (((int) *(buffer +.3 + 4*j)) & 0xFF);
      }
```

```
    }
    for (i=0; i<n; i++)
    {
       *(buffer + 4*i) = (char) ((((*(emgav + i) + 5) / 9) & 0xFF);
       *(buffer + 1 + 4*i) = (char) ((((*(forceav + i) + 5) / 9) & 0xFF);
       *(buffer + 2 + 4*i) = (char) ((((*(mmgav + i) + 5) / 9) >> 8) & 0xFF);
       *(buffer + 3 + 4*i) = (char) ((((*(mmgav + i) + 5) / 9) & 0xFF);
    }
    save ();

    cfree (emgav);
    cfree (forceav);
    cfree (mmgav);

    return;
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       beep.c
*
*               This function causes the internal speaker to sound a tone.
*       If the speaker is already busy making a sound, this function waits
*       until it is finished before initiating the new sound.  The frequency
*       (in Hz) and the duration (in 1/100ths of a second) of the tone are
*       specified by the integers 'freq' and 'dur'.
*
*                       )100 Hz < freq < 5000 Hz
*
*       Note :  This function requires that /dev/beeper already be opened,
*               with sp_fd (speaker file descriptor) defined as an external
*               integer.
*
*       -> copy this file to /tmp and compile to generate an object file using
*               /cpp/bin/fcc -c beep.c      or      cc -c beep.c
*
*       -> to link with other required object files, use
*               /cpp/bin/fcc files.o beep.o      or      cc files.o beep.o
*
*****************************************************************************/
#include <stdio.h>
#include <fcntl.h>
#include <sys/beep.h>
#include <sys/ioctl.h>

beep (freq, dur)
int freq, dur;
{
   extern int sp_fd;
   struct freqdur speakdat;

      /* test to see if the beeper is busy */
   do
      ioctl (sp_fd, BEEP_READ, &speakdat);
   while (speakdat.duration > 0);

      /* set up for the beep */
   speakdat.frequency = (unsigned short) freq;
   speakdat.duration = (unsigned short) dur;

      /* Do it! */
   ioctl (sp_fd, BEEP, &speakdat);

   return;
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       fkey_lbl.c
*                       -> writes special function key labels to the alpha
*                               window.
*                       -> the labels must be pointed to with the pointer
*                               array *klbl[9].
*
*       -> copy this file to /tmp and compile to generate object file using
*                       /cpp/bin/fcc -c fkey_lbl.c
*
*       -> to link with other object files, use
*                       /cpp/bin/fcc other_files.o fkey_lbl.o
*
*****************************************************************************/

#include <stdio.h>

fkey_lbl ()
{
   extern char *klbl[];

   char string[26];
   int i;

   for (i=1; i<=8; i++)
   {
        /* prefix the label with term0 softkey escape characters */
      sprintf (string, "\33&f2a%1dk160%16.16s", i,klbl[i]);
      write (1, string, 26);  /* write softkey definition without buffering! */
   }

   write (1, "\33&jB", 4);       /* turn on application menu */

   return;
}
/*****************************************************************************/
```

```
/*****************************************************************************/
/*
        fkey_scan.c
                        -> waits for a special function key to be pressed
                                and returns an integer indicating which key
                                was pressed.

                                -> return(i); if [fi] is pressed.

        -> copy this file to /tmp and compile to generate object file using
                        /cpp/bin/fcc -c fkey_scan.c

        -> to link with other object files, use
                        /cpp/bin/fcc other_files.o fkey_scan.o
                                                                        */
/*****************************************************************************/

#include <stdio.h>
#include <termio.h>

fkey_scan()
{
   struct termio t, savet;
   char key;

   ioctl (0, TCGETA, &savet);   /* save tty parameters */

      /* put window in raw mode */
   ioctl (0, TCGETA, &t);       /* get tty parameters */
   t.c_cc[VMIN] = 1;    /* read at least 1 characters */
   t.c_cc[VTIME] = 0;   /* do not time-out */
   t.c_lflag &= ~(ICANON | XCASE | ECHO);  /* place in raw mode */
   ioctl (0, TCSETAW, &t);      /* set the new parameters */
   write (1, "\33&s1A", 5);     /* set the transmit strap */

   while (1)
   {
      read (0, &key, 1);

      if (key == '\33')
      {
         read (0, &key, 1);

         if (('o' < key) && (key < 'x'))
         {
              /* return window to cooked mode */
            ioctl (0, TCSETAW, &savet); /* set the new parameters */
            write (1, "\33&s0A", 5);    /* reset the transmit strap */

            return (( (int) key & 0377) - 111);
         }
      }
   }
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*     length_exp.c
*
*     This programme block looks after the procedure for the muscle-length
*          experiment, the collection and storage of data and the calculation
*          of syringe plunger positions.
*
*     -> copy this file to /tmp and compile to gererate an object file using
*          /cpp/bin/fcc -c length_exp.c      or      cc -c length_exp.c
*
*     -> to link with other required object files, use
*          /cpp/bin/fcc files.o length_exp.o    or    cc files.o length_exp.o
*
*****************************************************************************/

#include <stdio.h>

length_exp ()
{
   extern char path[];
   extern int xs0, ys0, zs0, x, y, z, ml, ms, fn, cn, dn, leg;
   extern int fkey;
   extern int msa, msa100, mrl;
   extern float ls0;

   static int xdv[8] = {0,-12,-12,-12,-12,-12,-12,-12};
   static int ydv[8] = {0,0,0,0,0,0,0,0};
   static int zdv[8] = {0,0,0,0,0,0,0,0};
   static int mldv[8] = {0,100,90,95,100,105,110,100};

   dn++;        /* increment disc number */
   printf ("\nPlace disc /SMFD%02d in the internal disc drive.\n", dn);
   pause_continue ();

   ms = 100;    msa = msa100;
   printf ("\nMake sure that the stimulus amplitude is set at %d mv.\n",
                                                                  msa100);

   cn = 1;
   while (cn <= 7)
   {
      x = xdv[cn];    y = ydv[cn];    z = zdv[cn];    ml = mldv[cn];

      printf ("\nPosition the syringe plungers at:\n");
      printf ("   x -> %d mm\n", xs0-x);
      printf ("   y -> %d mm\n", ys0-y);
      printf ("   z -> %d mm\n", zs0-z);
      printf ("   muscle length syringe -> %f mm\n",
                                       ls0 - ((float) ((100-ml)*mrl))*0.01);

      printf ("\nAdjust magnetometer d.c. offset and check oscilloscope ");
      printf ("for 60 Hz noise\nbefore pressing [CONTINUE].\n");

      pause_continue ();

      sprintf (path, "/SMFD%02d/F%02d%cLC%02dS1",
                                       dn, fn, ((char) (82 - 6*leg)), cn);

      acquire3s2a ();

      sequence ();

      switch (fkey)
      {
         case 1 :
            cn++;
            break;
```

```
        case 2 :
           break;
        case 3 :
           cn--;
           break;
     }
   }

   printf ("\n\n\n**************************************************");
   printf ("*********************************\n");
   printf ("\n\t\tMUSCLE-LENGTH EXPERIMENT COMPLETE!\n");
   printf ("\n**************************************************");
   printf ("*********************************\n");
   printf ("\n\n\nSelect one of the special function keys below.\n");

   return;
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*     maxforce.c
*
*               This function extracts the maximum force integer from intbuf.
*
*     -> copy this file to /tmp and compile to gererate an object file using
*          /cpp/bin/fcc -c maxforce.c      or      cc -c maxforce.c
*
*     -> to link with other required object files, use
*          /cpp/bin/fcc files.o maxforce.o      or      cc files.o maxforce.o
*
*****************************************************************************/

maxforce ()
{
   extern char *buffer;
   extern int n;
   int i, tmp, force;

   force = ((int) *(buffer + 1)) & 0xFF;

   for (i=1; i<n; i++)
   {
      tmp = ((int) *(buffer + 1 + 4*i)) & 0xFF;
      if (tmp > force)
         force = tmp;
   }

   return (force);
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*     muscle_dat.c
*
*     This programme block looks after:
*           - the recording of individual muscle parameters.
*           - the determination of the muscle rest length.
*           - the determination of the minimum stimulus that elicits the
*                maximum twitch force.
*           - the determination of the origin of the spatial coordinate
*                system that determines the muscle position with respect
*                to the magnetometer sensing coil.
*
*     -> copy this file to /tmp and compile to gererate an object file using .
*           /cpp/bin/fcc -c muscle_dat.c      or      cc -c muscle_dat.c
*
*     -> to link with other required object files, use
*           /cpp/bin/fcc files.o muscle_dat.o     or    cc files.o muscle_dat.o
*
*****************************************************************************/

#include <stdio.h>

muscle_dat ()
{
   extern char *buffer;
   extern int n, sn, spec, sex, leg, dn;
   extern int xs0, ys0, zs0, fn;
   extern int fkey, force100;
   extern long dt;
   extern int emgsf, forcesf, msa100, msd, fw, fl, mw, mrl, mbtmp;
   extern float ls0;

   int i;

   sn = 0;      /* Reset the stmimulus number */

   printf ("\nEnter the frog leg number.\n");
   printf ("   0 = right \n   1 = left\n");
   printf ("integer [Return]\n");
   scanf_("%d", &leg);
   printf ("\nEnter the muscle weight in 1/100ths of grams.\n");
   printf ("integer [Return]\n");
   scanf ("%d", &mw);
   printf ("\nEnter the resting length of the muscle in mm.\n");
   printf ("integer [Return]\n");
   scanf ("%d", &mrl);
   printf ("\nEnter the muscle bath temperature in 1/10ths of degrees ");
   printf ("Celcius.\n");
   printf ("integer [Return]\n");
   scanf ("%d", &mbtmp);

      /* Determination of the muscle rest length */
   printf ("\n*****************************************************");
   printf ("*******************************\n");
   printf ("\nTo set the rest length of the muscle, the muscle should be ");
   printf ("stretched \nand then gradually released until the muscle tension");
   printf (" drops to zero.\nAuditory feedback in the form of two beeps will");
   printf (" be provided \nto indicate 'zero.force' and 'actual force'.\n");
   printf ("\nPress [Continue] when ready to proceed.\n");
   printf ("\n(You will receive 40 pairs of beeps)\n");

   pause_continue ();

   do
   {
      for (i=0; i<40; i++)
      {
```

```
      sleep (1);
      acquire1 ();
      beep (200, 25);
      beep (200 + 10*(((int) *(buffer + 1)) & 0xFF), 25);
   }

   repeat_complete ();
}
while (fkey == 1);

printf ("\nEnter the position of the length syringe in mm.\n");
printf ("(floating point number) [Return]\n");
scanf ("%f", &ls0);

   /* Determination of maximum stimulus */
printf ("\n*************************************************");
printf ("*****************************\n");
printf ("\nAdjust the stimulus amplitude until the minimum stimulus is ");
printf ("found \nthat elicits the maximum response.\n");
printf ("Use the [Repeat] key until the maximum stimulus has been found.\n");
printf ("\nNote that the last time [Repeat] is pressed sets the ");
printf ("value of force100.\n");

do
{
   acquire ();
   force100 = maxforce ();
   printf ("\nMaximum force in muscle twitch = %d (ADC int.)\n", force100);

   repeat_complete ();
}
while (fkey == 1);

printf ("\nEnter the stimulus amplitude in mv.\n");
printf ("integer [Return]\n");
scanf ("%d", &msa100);
printf ("\nEnter the stimulus duration in microseconds.\n");
printf ("integer [Return]\n");
scanf ("%d", &msd);

   /* Determination of Positon Origin */
printf ("\n\n*************************************************");
printf ("*****************************\n");
printf ("\nCenter the muscle under the magnetometer and enter \n");
printf ("the syringe plunger locations.\n");
printf ("\nEnter the x-syringe plunger position in millimeters.\n");
printf ("integer [Return]\n");
scanf ("%d", &xs0);
printf ("\nEnter the y-syringe plunger position in millimeters.\n");
printf ("integer [Return]\n");
scanf ("%d", &ys0);
printf ("\nEnter the z-syringe plunger position in millimeters.\n");
printf ("integer [Return]\n");
scanf ("%d", &zs0);

printf ("\n\n\n\n*************************************************");
printf ("*****************************\n");
printf ("\n\t\tSET-UP COMPLETE!\n");
printf ("\n*************************************************");
printf ("*****************************\n");
printf ("\n\n\nSelect one of the experiments listed below:\n");

return;
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*     pause_continue:c
*
*               This function makes use of the special function keys to assist
*       in the control of programme flow.  It allows the main programme to
*       pause while an experimental procedure is carried out.
*
*     -> copy this file to /tmp and compile to gererate an object file using
*          /cpp/bin/fcc -c pause_continue.c      or     cc -c pause_continue.c
*
*     -> to link with other required object files, use
*          /cpp/bin/fcc files.o pause_continue.o
*                                       or      cc files.o pause_continue.o
*
*****************************************************************************/

pause_continue ()
{
   extern char *klbl[];
   extern int fkey;

      /* define the special function key labels */
   klbl[1] = "        CONTINUE";
   klbl[2] = "                 ";
   klbl[3] = "                 ";
   klbl[4] = "                 ";
   klbl[5] = "                 ";
   klbl[6] = "                 ";
   klbl[7] = "                 ";
   klbl[8] = "                 ";

   fkey_lbl ();/* write the key labels to the alpha window */

   do
      fkey = fkey_scan();    /* receive function key input */
   while (fkey != 1);    /* test function key input */

   write (1, "\33&j@", 4);      /* turn off menu */

   return;
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*     pos_exp.c
*
*     This programme block looks after the procedure for the 'position'
*          experiment, the collection and storage of data and the calculation
*          of syringe plunger positions.
*
*     -> copy this file to /tmp and compile to gererate an object file using
*          /cpp/bin/fcc -c pos_exp.c      or      cc -c pos_exp.c
*
*     -> to link with other required object files, use
*          /cpp/bin/fcc files.o pos_exp.o      or      cc files.o pos_exp.o
*
*****************************************************************************/

#include <stdio.h>

pos_exp ()
{
   extern char path[];
   extern int xs0, ys0, zs0, x, y, z, ml, ms, fn, cn, dn, leg;
   extern int fkey;
   extern int msa100, msa;
   extern float ls0;

   static int xdv[15] = {0,-12,-7,-12,-17,-12,-12,-12,0,0,0,0,0,0,0};
   static int ydv[15] = {0,0,0,0,0,0,0,0,-12,-7,-12,-17,-12,-12,-12};
   static int zdv[15] = {0,0,0,0,0,0,-5,-10,0,0,0,0,0,-5,-10};

   dn++;        /* increment disc number */
   printf ("\nPlace disc /SMFD%02d in the internal disc drive.\n", dn);
   pause_continue ();

   ml = 100;    ms = 100;    msa = msa100;
   printf ("\nMake sure that the stimulus amplitude is set at %d mv\n", msa100);
   printf ("and the muscle length syringe at %f mm.\n", ls0);

   cn = 1;
   while (cn <= 14)
   {
      if (cn == 8)
      {
         printf ("\nMount the frog's other leg in the muscle holder\nfor the ");
         printf ("rest of the experiment.\n");
         muscle_dat ();
         pause_continue ();
      }

      x = xdv[cn];    y = ydv[cn];    z = zdv[cn];

      printf ("\nPosition the syringe plungers at:\n");
      printf ("   x -> %d mm\n", xs0-x);
      printf ("   y -> %d mm\n", ys0-y);
      printf ("   z -> %d mm\n", zs0-z);

      printf ("\nAdjust magnetometer d.c. offset and check the oscilloscope ");
      printf ("for 60 Hz noise\nbefore pressing [CONTINUE].\n");

      pause_continue ();

      sprintf (path, "/SMFD%02d/F%02d%cPC%02dS1",
                                          dn, fn, ((char) (82 - 6*leg)), cn);

      acquire3s2a ();

      sequence ();
```

```
    switch (fkey)
    {
       case 1 :
          cn++;
          break;
       case 2 :
          break;
       case 3 :
          cn--;
          break;
    }
  }
  printf ("\n\n\n**********************************************************");
  printf ("*********************************\n");
  printf ("\n\t\tPOSITION EXPERIMENT COMPLETE!\n");
  printf ("\n**********************************************************");
  printf ("*********************************\n");
  printf ("\n\n\nSelect one of the special function keys below.\n");
  return;
}
/*****************************************************************************/


/*****************************************************************************
*
*     repeat_complete.c
*
*               This function uses the special function keys to assist in the
*       control of programme flow.  It allows the repetition of a programme
*       block or continuation to the next part of the programme.
*
*     -> copy this file to /tmp and compile to gererate an object file using
*          /cpp/bin/fcc -c repeat_complete.c     or    cc -c repeat_complete.c
*
*     -> to link with other required object files, use
*        /cpp/bin/fcc files.o repeat_complete.o or cc files.o repeat_complete.o
*
*****************************************************************************/

repeat_complete ()
{
   extern char *klbl[];
   extern int fkey;

      /* define the special function key labels */
   klbl[1] = "         REPEAT ";
   klbl[2] = "                ";
   klbl[3] = "                ";
   klbl[4] = "                ";
   klbl[5] = "                ";
   klbl[6] = "                ";
   klbl[7] = "                ";
   klbl[8] = "       COMPLETE";

   fkey_lbl (); /* write the key labels to the alpha window */

   do
      fkey = fkey_scan();       /* receive function key input */
   while ( (fkey != 1) && (fkey != 8) );

   write (1, "\33&j@", 4);      /* turn off menu */

   return;
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       save.c
*                       -> this programme writes the DSQ-400 data to a file
*                               on the internal disk drive.  For example:
*                               /SMFDAT/LF24RC13A2
*
*       -> copy this file to /tmp and compile to generate object file using
*                       /cpp/bin/fcc -c save.c
*
*       -> to link with other required object files, use
*               /cpp/bin/fcc other_files.o save.o
*
*****************************************************************************/

#define N_INIT  2048            /* number of elements in each data vector */
#define BUF_SIZE  64+4*N_INIT   /* number of short-integers in intbuf */

#include <stdio.h>
#include <fcntl.h>

save ()
{
   extern char path[], *buffer;
   extern int n, r, sn, rt, x, y, z, ms, ml, mmgfc, spec, sex, leg, dn;
   extern long dt, ct;
   extern int emgsf, forcesf, msa, msd, fw, fl, mw, mrl, mbtmp;

   int fd, oflag, mode, n_written;

      /* Determine the time since the disection started */
   if ( (ct = time ((long *) 0) ) == -1)        /* get current time */
      printf ("\ntime() not successfully called in save()!\n");
   rt = (short) (ct - dt);    /* determine 'relative time' (since disection) */

      /* Load buffers with variables */
   *(buffer + 4*n) = (char) ((n >> 8) & 0xFF);
   *(buffer + 1 + 4*n) = (char) (n & 0xFF);
   *(buffer + 2 + 4*n) = (char) ((r >> 8) & 0xFF);
   *(buffer + 3 + 4*n) = (char) (r & 0xFF);
   *(buffer + 4 + 4*n) = (char) ((sn >> 8) & 0xFF);
   *(buffer + 5 + 4*n) = (char) (sn & 0xFF);
   *(buffer + 6 + 4*n) = (char) ((rt >> 8) & 0xFF);
   *(buffer + 7 + 4*n) = (char) (rt & 0xFF);
   *(buffer + 8 + 4*n) = (char) (x & 0xFF);
   *(buffer + 9 + 4*n) = (char) (y & 0xFF);
   *(buffer + 10 + 4*n) = (char) (z & 0xFF);
   *(buffer + 11 + 4*n) = (char) (ms & 0xFF);
   *(buffer + 12 + 4*n) = (char) (ml & 0xFF);
   *(buffer + 13 + 4*n) = (char) ((mmgfc >> 8) & 0xFF);
   *(buffer + 14 + 4*n) = (char) (mmgfc & 0xFF);
   *(buffer + 15 + 4*n) = (char) (spec & 0xFF);
   *(buffer + 16 + 4*n) = (char) (sex & 0xFF);
   *(buffer + 17 + 4*n) = (char) (leg & 0xFF);

   *(buffer + 18 + 4*n) = (char) ((emgsf >> 8) & 0xFF);
   *(buffer + 19 + 4*n) = (char) (emgsf & 0xFF);
   *(buffer + 20 + 4*n) = (char) ((forcesf >> 8) & 0xFF);
   *(buffer + 21 + 4*n) = (char) (forcesf & 0xFF);
   *(buffer + 22 + 4*n) = (char) ((msa >> 8) & 0xFF);
   *(buffer + 23 + 4*n) = (char) (msa & 0xFF);
   *(buffer + 24 + 4*n) = (char) ((msd >> 8) & 0xFF);
   *(buffer + 25 + 4*n) = (char) (msd & 0xFF);
   *(buffer + 26 + 4*n) = (char) ((fw >> 8) & 0xFF);
   *(buffer + 27 + 4*n) = (char) (fw & 0xFF);
   *(buffer + 28 + 4*n) = (char) (fl & 0xFF);
   *(buffer + 29 + 4*n) = (char) ((mw >> 8) & 0xFF);
   *(buffer + 30 + 4*n) = (char) (mw & 0xFF);
```

```
   *(buffer + 31 + 4*n) = (char) (mrl & 0xFF);
   *(buffer + 32 + 4*n) = (char) ((mbtmp >> 8) & 0xFF);
   *(buffer + 33 + 4*n) = (char) (mbtmp & 0xFF);

      /* Create the disc file and write the data in it */
   oflag = O_CREAT | O_WRONLY;  /* create a write-only file */
   mode = 00400 | 00200;        /* owner read and write access permission */

   while (1)
   {
      if ( (fd = open(path, oflag, mode)) != -1)        /* create the file */
      {
         if ( (write (fd, buffer, BUF_SIZE)) == BUF_SIZE)
         {
            close (fd); /* close the data file */
            return;     /* disc-save successfully completed! */
         }
         close (fd);             /* close the incomplete data file */
         printf ("\nThe %s data file could not be completely ", path);
         printf ("written!\n");
      }
      else
      {
         printf ("\nThe %s data file cannot be opened!\n", path);
         printf ("Is the /SMFD%02d disc in the disc drive?\n", dn);

      }

      printf ("\n(press [CONTINUE] when ready)\n");
      pause_continue ();
   }
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*     sequence.c
*
*     This function allows repetition of a main programme segment under present
*          conditions or under previous conditions.  If no repetition is
*          desired, then main programme execution resumes.
*
*     -> copy this file to /tmp and compile to gererate an object file using
*          /cpp/bin/fcc -c sequence.c      or      cc -c sequence.c
*
*     -> to link with other required object files, use
*          /cpp/bin/fcc files.o sequence.o      or      cc files.o sequence.o
*
*****************************************************************************/

sequence()
{
   extern char *klbl[];
   extern int fkey;

      /* define the special function key labels */
   klbl[1] = "        CONTINUE";
   klbl[2] = "         REPEAT ";
   klbl[3] = "        PREVIOUS";
   klbl[4] = "                ";
   klbl[5] = "                ";
   klbl[6] = "                ";
   klbl[7] = "                ";
   klbl[8] = "                ";

   fkey_lbl (); /* write the key labels to the alpha window */

   do
      fkey = fkey_scan();       /* receive function key input */
   while (fkey > 3);

   write (1, "\33&j@", 4);      /* turn off menu */

   return;
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*     set_up.c
*
*     This programme block looks after:
*           - the setting of analogue instrument gains.
*           - the calibration of the analogue circuits.
*           - the recording of physical frog parameters.
*           - the determination of the muscle rest length.
*           - the determination of the minimum stimulus that elicits the
*                maximum twitch force.
*           - the determination of the origin of the spatial coordinate
*                system that determines the muscle position with respect
*                to the magnetometer sensing coil.
*
*     -> copy this file to /tmp and compile to gererate an object file using
*           /cpp/bin/fcc -c set_up.c      or      cc -c set_up.c
*
*     -> to link with other required object files, use
*           /cpp/bin/fcc files.o set_up.o      or      cc files.o set_up.o
*
*****************************************************************************/

#include <stdio.h>

set_up ()
{
   extern char *buffer;
   extern int n, spec, sex, leg, dn;
   extern int xs0, ys0, zs0, fn;
   extern int fkey, force100;
   extern long dt;
   extern int emgsf, forcesf, msa100, msd, fw, fl, mw, mrl, mbtmp;
   extern float ls0;

   char *ctime();
   int emgcaln, forcecaln, calv, calm;
   int i;

   printf ("\nEnter the data disc number.\n");
   printf ("integer [Return]\n");
   scanf ("%d", &dn);
   printf ("The disc in the internal drive should have the name ");
   printf ("/SMFD%02d.\n", dn);
   dn--;        /* dn is incremented at the beginning of each experiment */

      /* Muscle Force Zero Offset */
   printf ("\n**************************************************");
   printf ("**********\n");
   printf ("\nSet the muscle force zero-offset with the aid of the follow");
   printf ("ing auditory \nfeedback.  The pitch of the first of two beeps ");
   printf ("indicates zero, \nwhile the second indicates the actual signal.\n");
   printf ("(you will be given 20 pairs of beeps)\n");
   printf ("Press [CONTINUE] to procede.\n");

   pause_continue ();

   do
   {
      for (i=0; i<20; i++)
      {
         acquire1 ();
         beep (200, 25);        /* beep frequency represents signal amplitude */
         beep (200 + 10*(((int) *(buffer + 1)) & 0xFF), 25);   /* 100Hz -> 0v */
         sleep (1);
      }
      printf ("The final value of the force signal was %d (ADC integer).\n",
                                             (((int) *(buffer + 1)) & 0xFF));
```

```
   repeat_complete ();
}
while (fkey == 1);

   /* Calibration Procedure */
printf ("\nApply the calibration voltage to the EMG pre-amp. and the \n");
printf ("calibration mass to the force transducer (after pressing ");
printf ("continue).\nThe EMG and force will be sampled 20 times, each ");
printf ("indicated by a beep.\nThe maximum values will be used as ");
printf ("calibration integers.\n");
printf ("(press [Continue] to procede)\n");

pause_continue();

do
{
   beep (440, 25);   /* 1/4 second A 440 */
   acquire1 ();
   emgcaln = (((int) *buffer) & 0xFF) - 128;
   forcecaln = ((int) *(buffer + 1)) & 0xFF;
   for (i=0; i<19; i++)
   {
      sleep (1);
      beep (440, 25);
      acquire1 ();
      if ( ((((int) *buffer) & 0xFF) - 128) > emgcaln )
         emgcaln = (((int) *buffer) & 0xFF) - 128;
      if ( (((int) *(buffer + 1)) & 0xFF) > forcecaln )
         forcecaln = ((int) *(buffer + 1)) & 0xFF;
   }
   if (emgcaln == 255)
   {
      printf ("\nThe emg signal has overflowed its range!\n");
      printf ("Adjust the gain and repeat this procedure.\n");
   }
   if (forcecaln == 255)
   {
      printf ("\nThe force signal has overflowed its range!\n");
      printf ("Adjust the gain and repeat this procedure.\n");
   }
   printf ("\nThe EMG and force calibration integers are ");
   printf ("%3d and %3d respectively.\n", emgcaln,forcecaln);

   repeat_complete ();

}
while (fkey == 1);

printf ("\nEnter the calibration voltage (in millivolts).\n");
printf ("integer [Return]\n");
scanf ("%d", &calv);
printf ("\nEnter the calibration mass (in grams).\n");
printf ("integer [Return]\n");
scanf ("%d", &calm);
emgsf = calv * 1000 / emgcaln;
forcesf = 9800 * calm / forcecaln;
printf ("\nThe EMG and force scale factors are\n");
printf ("%d microvolts and %d microNewtons respectively.\n", emgsf, forcesf);

   /* Physical Frog Data */
printf ("\n**************************************************");
printf ("*****************************\n");
printf ("\nKill the frog and return to enter physical data.\n");
printf ("\n(press [Continue] when you return)\n");

pause_continue ();

if ( (dt = time ((long *) 0) ) == -1)        /* determine disection time */
```

```
{
   printf ("\ntime() not successfully called!\n");
   return;
}
printf ("\nThe disection has been recorded as starting at ");
printf ("%s\n", ctime(&dt));

sleep (1);

printf ("\nEnter the 'frog number'.\n");
printf ("integer [Return]\n");
scanf ("%d", &fn);
printf ("\nEnter the frog species number:\n   0 = Rana pipiens pipiens");
printf ("   1 = Rana catesbeiana \n   2 = other\n");
printf ("integer [Return]\n");
scanf ("%d", &spec);
printf ("\nEnter the frog sex number:\n");
printf ("   0 = male \n   1 = female\n");
printf ("integer [Return]\n");
scanf ("%d", &sex);
printf ("\nEnter the frog weight in 1/100ths of grams.\n");
printf ("integer [Return]\n");
scanf ("%d", &fw);
printf ("\nEnter the frog length (nose to crotch) in mm.\n");
printf ("integer [Return]\n");
scanf ("%d", &fl);
printf ("\n**************************************************");
printf ("*******************************\n");
printf ("\nPerform the disection and mount the muscle in the bath.\n");
printf ("\n(press [Continue] when you return)\n");

pause_continue ();

muscle_dat ();

return;
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*     single_save.c
*
*     This programme block looks after the procedure for saving a single data
*          record that is not part of the three main experiments.
*
*     -> copy this file to /tmp and compile to gererate an object file using
*          /cpp/bin/fcc -c single_save.c      or      cc -c single_save.c
*
*     -> to link with other required object files, use
*          /cpp/bin/fcc files.o single_save.o    or   cc files.o single_save.o
*
*****************************************************************************/

#include <stdio.h>
#include <fcntl.h>

single_save ()
{
   extern char path[];
   extern int xs0, ys0, zs0, x, y, z, ml, ms, fn, cn, dn, leg;
   extern int fkey;
   extern int msa100, msa;
   extern float ls0;
   extern long dt, ct;

   printf ("\nPlace disc /SMFD%02d in the internal disc drive.\n", dn);
   pause_continue ();

   do
   {
      printf ("\nAdjust magnetometer d.c. offset and check the oscilloscope ");
      printf ("for 60 Hz noise\nbefore pressing [CONTINUE].\n");

      pause_continue ();

         /* determine current time */
      if ( (ct = time ((long *) 0) ) == -1)
         printf ("\ntime() not successfully called in single_save()!\n");

      sprintf (path, "/SMFD%02d/F%02d%c%04d$1",
                              dn, fn, ((char) (82 - 6*leg)), (ct-dt)/10);

      acquire3s2a ();

      repeat_complete ();

   }
   while (fkey == 1);

   printf ("\n\n\n**************************************************");
   printf ("*****************************\n");
   printf ("\n\t\tDATA STORAGE COMPLETE!\n");
   printf ("\n**************************************************");
   printf ("*****************************\n");
   printf ("\n\n\nSelect one of the special function keys below.\n");

   return;
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*     stim_exp.c
*
*     This programme block looks after the procedure for the muscle-stimulation
*          experiment, the collection and storage of data and the calculation
*          of syringe plunger positions.
*
*     -> copy this file to /tmp and compile to gererate an object file using
*          /cpp/bin/fcc -c stim_exp.c      or      cc -c stim_exp.c
*
*     -> to link with other required object files, use
*          /cpp/bin/fcc files.o stim_exp.o    or    cc files.o stim_exp.o
*
*****************************************************************************/

#include <stdio.h>

stim_exp ()
{
   extern char path[];
   extern int xs0, ys0, zs0, x, y, z, ml, ms, fn, cn, dn, leg;
   extern int fkey, force100;
   extern int msa, msa100;
   extern float ls0;

   static int xdv[8] = {0,-12,-12,-12,-12,-12,-12,-12};
   static int ydv[8] = {0,0,0,0,0,0,0,0};
   static int zdv[8] = {0,0,0,0,0,0,0,0};
   static int msdv[8] = {0,100,80,100,60,100,40,100};

   int forcem;

   dn++;        /* increment disc number */
   printf ("\nPlace disc /SMFD%02d in the internal disc drive.\n", dn);
   pause_continue ();

   ml = 100;
   printf ("\nMake sure that the muscle-length syringe is set at %f mm.\n",ls0);

   cn = 1;
   while (cn <= 7)
   {
      x = xdv[cn];    y = ydv[cn];    z = zdv[cn];    ms = msdv[cn];

      printf ("\nPosition the syringe plungers at:\n");
      printf ("   x -> %d mm\n", xs0-x);
      printf ("   y -> %d mm\n", ys0-y);
      printf ("   z -> %d mm\n", zs0-z);

      sleep (4);

      do
      {
         acquire ();
         forcem = maxforce ();
         printf("\nMaximum force in muscle twitch = %d (ADC integer)\n",forcem);
         printf ("Adjust the stimulus potentiometer so that the \nmaximum for");
         printf ("ce in the muscle twitch = %d\n", msa*force100/100);
         printf ("Use the [Repeat] key until the desired level is found.\n");

         repeat_complete ();
      }
      while (fkey == 1);

      printf ("\nEnter the stimulus amplitude setting in mv.\n");
      printf ("integer [Return]\n");
      scanf ("%d", &msa);
```

```
            printf ("\nAdjust magnetometer d.c. offset and check the oscilloscope ");
            printf ("for 60 Hz noise\nbefore pressing [CONTINUE]\n");

            pause_continue ();

            sprintf (path, "/SMFD%02d/F%02d%cSC%02dS1",
                                                dn, fn, ((char) (82 - 6*leg)), cn);

            acquire3s2a ();

            sequence ();

            switch (fkey)
            {
               case 1 :
                  cn++;
                  break;
               case 2 :
                  break;
               case 3 :
                  cn--;
                  break;
            }
         }

         printf ("\n\n\n\n**********************************************************");
         printf ("*****************************\n");
         printf ("\n\t\tMUSCLE-STIMULUS EXPERIMENT COMPLETE!\n");
         printf ("\n**********************************************************");
         printf ("*****************************\n");
         printf ("\n\n\nSelect one of the special function keys below.\n");

         return;
      }
      /*****************************************************************************/
```

APPENDIX B

DATA_CHECK SOURCE CODE

Presented in this appendix is the C programming language source code for the DATA_CHECK programme. It provided facilities for printing and plotting data during an experiment. The multitasking operating system was used to run this programme simultaneously with SMDA (SQUID Magntometer Data Acquisition).

Many C language subroutine calls and HP-UX (UNIX) system calls appear in the following listings. It is assumed that usage of these calls will be familiar to other programmers or can be determined by reference to standard C and UNIX manuals. Included in these listings are all the "user defined" subroutines required by DATA_CHECK. Except for DATA_CHECK "main" and "quit", the subroutines are presented in alphabetical order in the following pages:

data_check: main
quit

acquire
cfft
display_data
fkey_lbl
fkey_scan
get_data
open_graph
plot
plot_prep
smda_graphic.

```
/*****************************************************************************
*
*       data_check.c
*               ->  interactive data-checking programme (in C).
*
*               This programme reads disc data files created by the
*       programme 'smda' and displays and plots this data.
*
*       Desired functions are selected with the special function keys.
*
*       -> copy this file to /tmp and compile to generate object file using
*               /cpp/bin/fcc -c data_check.c      or      cc -c data_check.c
*
*       -> to link with the other required object files, copy only the required
*               object files to /tmp and use
*               /cpp/bin/fcc *.o -ldvio       or      cc *.o -ldvio
*
*               -> included in *.o must be
*               open_graph.o, fkey_lbl.o, fkey_scan.o, acquire.o, cfft.o,
*               display_data.o, graph_data.o, get_data.o, plot.o, get_data.o,
*               smda_graphics.o
*
*****************************************************************************/

#define MAX_SIZE  2048  /* maximum number of elements in each data vector */
#define N_INIT    2048  /* initial number of elements in each data vector */
#define R_INIT    8192  /* initial data acquisition rate (samples/second) */

#include <stdio.h>

char path[19], *klbl[9];
short *emg, *force, *mmg;
int sn, rt, x, y, z, ms, ml, mmgfc, spec, sex, leg;
int n = N_INIT ,  r = R_INIT ;  /* initialize 'n' and 'r' */
int gfd, fkey;
int emgsf, forcesf, msa, msd, fw, fl, mw, mrl, mbtmp;

main()
{
   char *calloc();
   int i;

      /* allocate memory for emg, force and mmg */
   emg = (short *) calloc(MAX_SIZE, sizeof(short));
   force = (short *) calloc(MAX_SIZE, sizeof(short));
   mmg = (short *) calloc(MAX_SIZE, sizeof(short));
   if (emg==0 || force==0 || mmg==0)
   {
      printf ("\nMemory allocation failure !\nClear some RAM?\n");
      exit(1);
   }

   for (i=0; i<18; i++)         /* initialize the data file name */
      path[i] = ' ';
   path[18] = '\0';

      /* open the graphics window */
   if ((gfd = open_graph (512,255,0,0)) == -1)
      printf ("\nFailure to open graphics window!\n");

      /* print start-up message */
   printf ("\n*************************");
   printf ("*************************************************\n");
```

```
   /* main control loop of ida.c */

while (1)
{
      /* define the special function key labels */
   klbl[1] = "Collect   data  ";
   klbl[2] = "Display   data  ";
   klbl[3] = "                ";
   klbl[4] = "  Plot    data  ";
   klbl[5] = "                ";
   klbl[6] = "  Get     data  ";
   klbl[7] = "                ";
   klbl[8] = "  Quit          ";

   fkey_lbl ();      /* write the key labels to the alpha window */
   printf ("\nPress special function keys (f1 to f8) ");
   printf ("for desired action.\n\n*************************");
   printf ("*************************************************\n");

   fkey = fkey_scan();       /* receive function key input */
   write (1, "\33&j@", 4);   /* turn off menu */

   switch (fkey)
   {
      case 1 :
         acquire ();
         break;
      case 2 :
         display_data ();
         break;
      case 3 :
         break;
      case 4 :
         smda_graphics ();
         break;
      case 5 :
         break;
      case 6 :
         get_data ();
         break;
      case 7 :
         break;
      case 8 :
         quit();
         break;
   }
  }
}
/*****************************************************************************/

quit()
{
   extern char *klbl[];
   extern int fkey;

   printf ("\nDo you really want to quit ?!\n");

      /* define the special function key labels */
   klbl[1] = "          YES   ";
   klbl[2] = "                ";
   klbl[3] = "                ";
   klbl[4] = "           NO   ";
   klbl[5] = "                ";
   klbl[6] = "                ";
   klbl[7] = "                ";
   klbl[8] = "                ";

   fkey_lbl (); /* write the key labels to the alpha window */
```

```
   while (1)
   {
      fkey = fkey_scan();       /* receive function key input */
      if (fkey == 4)
         return;
      if (fkey == 1)
      {
         cfree (emg);
         cfree (force);
         cfree (mmg);
         printf ("\nProgramme data_check.c terminated.\n");
         printf ("\nTo restart this programme, high-light ");
         printf ("'data_check' in the PAM window\nand press [Start].\n");
         printf ("\n*********************************");
         printf ("************************************************\n");

         klbl[1] = "                ";
         klbl[4] = "                ";
         fkey_lbl ();   /* write the key labels to the alpha window */

         exit (0);
      }
   }
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       acquire.c       ->      data acquisition routine
*
*               -> this function collects 'n' samples of data from the DSQ-400
*                    SQUID magnetometer via the DIL and the HP-IB.
*
*       -> copy this file to /tmp and compile to generate object file using
*               /cpp/bin/fcc -c acquire.c       or      cc -c acquire.c
*
*       -> to link with other object files, use
*               /cpp/bin/fcc other_files.o acquire.o -ldvio      or      cc...
*
*****************************************************************************/

#include <stdio.h>
#include <fcntl.h>

#define INTER 0xE40023
```

```
acquire ()
{
   extern short *emg, *force, *mmg;
   extern int n;
   register unsigned char *clock_reg;
   int i, n_read, eid, MLA;
   char command[4], *calloc(), *buffer;

      /* allocate memory for intbuf */
   if ( (buffer = calloc (4*n, 1)) == 0)
   {
      printf ("\nMemory allocation failure in acquire() in data_check!\n");
      return;
   }

      /* open HP-IB for read/write access */
   if ((eid = open("/dev/dhpib.i", O_RDWR)) == -1)
   {
      printf("\nFailure to open /dev/dhpib.i\n");
      printf("\nDoes 'load_hpib' need to be run?\n");
      return;
   }

      /* configure the interface for data acquisition */
   MLA=hpib_bus_status (eid, 7) + 32;   /* determine IPC listen address */
   command[0]=95;                /* UNTALK command */
   command[1]=63;                /* UNLISTEN command */
   command[2]=64+25;             /* talk address for device 25 on HP-IB */
   command[3]=MLA;               /* IPC listen address */
   hpib_send_cmnd(eid,command,4);

   io_burst(eid,1);     /* put interface in high-speed 'burst' mode */

   clock_reg = (unsigned char *) INTER; /* disable beat interupts */
   *clock_reg = (unsigned char) 0x80;

   n_read = read (eid, buffer, 4*n);    /* perform data acquisition */

   clock_reg = (unsigned char *) INTER; /* re-enable beat interupts */
   *clock_reg = (unsigned char) 2;

   io_burst(eid,0);     /* return from 'burst' mode */

   hpib_send_cmnd (eid, command, 1);    /* send 'untalk' command */

   if (n_read != 4*n)
   {
      printf("\nData acquisition incomplete!\nOnly %d bytes ", n_read);
      printf("of the requested %d bytes were read.\n", 4*n);
      return;
   }

   for (i=0; i<n; i++)
   {
      *(emg + i) = (((short) *(buffer + 4*i)) & 0xFF) - 128;
      *(force + i) = ((short) *(buffer + 1 + 4*i)) & 0xFF;
      *(mmg + i) = (((short) *(buffer + 2 + 4*i)) << 8) |
                                    (((short) *(buffer + 3 + 4*i)) & 0xFF);
   }

   cfree (buffer);
   close (eid);

   return;
}
/*************************************************************************/
```

```
/*****************************************************************************
*
*       cfft.c
*               -> Complex Fast Fourier Transform
*
*       -> this function replaces 'data' with its discrete Fourier transform
*          if isign=1 or replaces 'data' with 'nn' times the inverse
*          discrete Fourier transform if 'isign'=-1.
*
*       -> 'data' is a real array of 2*nn elements with pairs of adjacent
*          elements representing the real and complex parts of each
*          data point.
*
*       -> 'nn' is the number of complex data points and must be an integer
*          power of 2.
*
*
*       -> copy this file to /tmp and compile to generate object file using
*          /cpp/bin/fcc -c cfft.c
*
*       -> to link with other required object files, use
*          /cpp/bin/fcc other_files.o cfft.o -lm
*
*****************************************************************************/

#include <math.h>

cfft (data,nn,isign)
float *data;
int nn,isign;
{
   double sin();
   int i,j,n,m,mmax,istep;
   float tempr,tempi;
   double wr,wi,wpr,wpi,wtemp,theta;

   n = nn * 2;
   j=1;

   for (i=1; i<=n; i+=2)                /* this is the BIT-reversal section */
   {
      if (j > i)
      {                                 /* exchange the two complex numbers */
         tempr = *(data + j-1);
         tempi = *(data + j);
         *(data + j-1) = *(data + i-1);
         *(data + j) = *(data + i);
         *(data + i-1) = tempr;
         *(data + i) = tempi;
      }

      m = nn;

      while ((m >= 2) && (j > m))
      {
         j -= m;
         m >>= 1;       /* divide m by 2 */
      }

      j += m;
   }

   mmax = 2;    /* here begins the Danielson-Lanczos section of the routine */

   while (n > mmax)     /* outer loop executed log2(nn) times */
   {
      istep = mmax << 1;        /* istep=mmax*2 */
      theta = 6.28318530717959 / (double)(isign * mmax);
```

```
        wpr = sin(0.5*theta);
        wpr *= wpr;
        wpr *= -2.0;
        wpi = sin(theta);
        wr = 1.0;
        wi = 0.0;

        for (m=1; m<=mmax; m+=2)  /* here are the two nested inner loops */
        {
           for (i=m; i<=n; i+=istep)
           {
              j = i + mmax;       /* this is the Danielson-Lanczos formula */
              tempr = (wr * *(data + j-1)) - (wi * *(data + j));
              tempi = (wr * *(data + j)) + (wi * *(data + j-1));
              *(data + j-1) = *(data + i-1) - tempr;
              *(data + j) = *(data + i) - tempi;
              *(data + i-1) += tempr;
              *(data + i) += tempi;
           }

           wtemp = wr;            /* trigonometric recurrence */
           wr = wr*wpr - wi*wpi + wr;
           wi = wi*wpr + wtemp*wpi + wi;
        }

        mmax = istep;
     }

     return;
}
/*************************************************************************/
```

```
/*****************************************************************************
*
*       display_data.c
*                       -> this programme writes the DSQ-400 data to the
*                               application window.
*
*       -> copy this file to /tmp and compile to generate object file using
*                       /cpp/bin/fcc -c display_data.c
*       -> to link with other required object files, use
*                /cpp/bin/fcc other_files.o display_data.o -lc
*
*****************************************************************************/

#include <stdio.h>

display_data ()
{
   extern char path[], *klbl[];
   extern short *emg, *force, *mmg;
   extern int n, r, sn, rt, x, y, z, ms, ml, mmgfc, spec, sex, leg;
   extern int fkey;
   extern int emgsf, forcesf, msa, msd, fw, fl, mw, mrl, mbtmp;
   int i, low=0, high=14;

      /* dispay first page with parameters */
   printf ("\n                                %s \n", path);
   printf ("n = %d  r = %d  sn = %d  rt = %d  x = %d  y = %d  z = %d\n",
                                                        n,r,sn,rt,x,y,z);
   printf ("ms = %d  ml = %d  mmgfc = %d  spec = %d  sex = %d  leg = %d\n",
                                                    ms,ml,mmgfc,spec,sex,leg);
   printf ("emgsf = %d  forcesf = %d  msa = %d  msd = %d \n",
                                                      emgsf,forcesf,msa,msd);
   printf ("fw = %d  fl = %d  mw = %d  mrl = %d  mbtmp = %d  \n",
                                                    fw,fl,mw,mrl,mbtmp);
   printf ("\n  sample      emg      force      mmg   \n");
   for (i=low; i<high; i++)
      printf("\n   %4d       %3d       %3d       %4d",
                        i, *(emg+i), *(force+i), *(mmg+i));
   low = 14;    /* reset 'low' */

      /* define the special function key labels */
   klbl[1] = "                ";
   klbl[2] = "Continue Display";
   klbl[3] = "                ";
   klbl[4] = "                ";
   klbl[5] = "                ";
   klbl[6] = "                ";
   klbl[7] = "                ";
   klbl[8] = "  Main    Menu  ";

   fkey_lbl (); /* write the key labels to the alpha window */

   while (low < n)      /* display only one page at a time */
   {
      if ( fkey = fkey_scan() == 8 )
         return;

      printf("\n\n  sample      emg      force      mmg\n");
      high += n-low > 21 ? 21: n-low;   /* increment 'high' */

      for (i=low; i<high; i++)
         printf("\n   %4d       %3d       %3d       %4d",
                  i, *(emg+i), *(force+i), *(mmg+i));
      low += 21;        /* increment 'low' */
   }
   return;
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       fkey_lbl.c
*                       -> writes special function key labels to the alpha
*                               window.
*                       -> the labels must be pointed to with the pointer
*                               array *klbl[9].
*
*       -> copy this file to /tmp and compile to generate object file using
*                       /cpp/bin/fcc -c fkey_lbl.c
*
*       -> to link with other object files, use
*                       /cpp/bin/fcc other_files.o fkey_lbl.o -lc
*
*****************************************************************************/

#include <stdio.h>

fkey_lbl ()
{
   extern char *klbl[];
   char string[26];
   int i;

   for (i=1; i<=8; i++)
   {
        /* prefix the label with term0 softkey escape characters */
      sprintf (string, "\33&f2a%1dk160%16.16s", i,klbl[i]);
      write (1, string, 26);  /* write softkey definition without buffering! */
   }

   write (1, "\33&j8", 4);      /* turn on application menu */

   return;
}
/*****************************************************************************/
```

```
/*****************************************************************************/
/*
        fkey_scan.c
                        -> waits for a special function key to be pressed
                                and returns an integer indicating which key
                                was pressed.

                                -> return(i); if [fi] is pressed.

        -> copy this file to /tmp and compile to generate object file using
                        /cpp/bin/fcc -c fkey_scan.c

        -> to link with other object files, use
                        /cpp/bin/fcc other_files.o fkey_scan.o -lc
                                                                        */
/*****************************************************************************/

#include <stdio.h>
#include <termio.h>

fkey_scan()
{
   struct termio t, savet;
   char key;

   ioctl (0, TCGETA, &savet);   /* save tty parameters */

      /* put window in raw mode */
   ioctl (0, TCGETA, &t);       /* get tty parameters */
   t.c_cc[VMIN] = 1;    /* read at least 1 characters */
   t.c_cc[VTIME] = 0;   /* do not time-out */
   t.c_lflag &= ~(ICANON | XCASE | ECHO);  /* place in raw mode */
   ioctl (0, TCSETAW, &t);      /* set the new parameters */
   write (1, "\33&s1A", 5);     /* set the transmit strap */

   while (1)
   {
      read (0, &key, 1);

      if (key == '\33')
      {
         read (0, &key, 1);

         if (('o' < key) && (key < 'x'))
         {
               /* return window to cooked mode */
            ioctl (0, TCSETAW, &savet); /* set the new parameters */
            write (1, "\33&s0A", 5);    /* reset the transmit strap */

            return (( (int) key & 0377) - 111);
         }
      }
   }
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       get_data.c
*                       -> this programme reads the DSQ-400 data from a file
*                             on disc SMFDAT in the internal disk drive.
*
*       -> copy this file to /tmp and compile to generate object file using
*                       /cpp/bin/fcc -c get_data.c
*
*       -> to link with other required object files, use
*              /cpp/bin/fcc other_files.o get_data.o
*
*****************************************************************************/

#include <stdio.h>
#include <fcntl.h>

get_data ()
{
   extern char path[], *klbl[];
   extern short *emg,*force,*mmg;
   extern int n, r, sn, rt, x, y, z, ms, ml, mmgfc, spec, sex, leg;
   extern int emgsf, forcesf, msa, msd, fw, fl, mw, mrl, mbtmp;
   char *calloc(), *buffer;
   int i,fd;
   char string[21];

      /* define the special function key labels */
   klbl[1] = "                ";
   klbl[2] = "                ";
   klbl[3] = "                ";
   klbl[4] = "                ";
   klbl[5] = "                ";
   klbl[6] = "                ";
   klbl[7] = "                ";
   klbl[8] = "                ";

      /* read the path name for the new file from the keyboard */
   write(1,"\15\12\12Enter the path name of the desired data file:\15\12", 50);
   fkey_lbl (); /* blank out user key labels */
   read (0, string, 21);
   for (i=0; i<18; i++)
      path[i] = string[i];
   write(1, "\33&ja", 4);       /* turn off menu */

      /* open the data file (read only mode) */
   if ((fd = open(path, O_RDONLY)) == -1)
   {
      write (1, "\15\12\12The data file cannot be opened!            ", 47);
      write (1, "\15\12Is disk 'DSQ400' in the internal disk drive?\12", 47);
      return;
   }

      /* allocate memory for character buffer */
   if ( (buffer = calloc (64+4*n, 1)) == 0)
      printf ("\nMemory allocation failure in get_data()!\n");

      /* read the data from the disk file */
   if ( read (fd, buffer, 64+4*n) != 64+4*n)
      printf ("\nIncomplete read from disc in get_data()!\n");

      /* unpack the data from the character buffer */
   for (i=0; i<n; i++)
   {
      *(emg + i) = (((short) *(buffer + 4*i)) & 0xFF) - 128;
      *(force + i) = ((short) *(buffer + 1 + 4*i)) & 0xFF;
      *(mmg + i) = (((short) *(buffer + 2 + 4*i)) << 8) |
                                    (((short) *(buffer + 3 + 4*i)) & 0xFF);
```

```
    )
    n = (((int) *(buffer + 4*n)) << 8) | (((int) *(buffer + 1 + 4*n)) & 0xFF);
    r = (((int) *(buffer + 2 + 4*n)) << 8) |
                                        (((int) *(buffer + 3 + 4*n)) & 0xFF);
    sn = (((int) *(buffer + 4 + 4*n)) << 8) |
                                        (((int) *(buffer + 5 + 4*n)) & 0xFF);
    rt = (((int) *(buffer + 6 + 4*n)) << 8) |
                                        (((int) *(buffer + 7 + 4*n)) & 0xFF);
    x = ((int) *(buffer + 8 + 4*n));
    y = ((int) *(buffer + 9 + 4*n));
    z = ((int) *(buffer + 10 + 4*n));
    ms = ((int) *(buffer + 11 + 4*n));
    ml = ((int) *(buffer + 12 + 4*n));
    mmgfc = (((int) *(buffer + 13 + 4*n)) << 8) |
                                        (((int) *(buffer + 14 + 4*n)) & 0xFF);
    spec = ((int) *(buffer + 15 + 4*n));
    sex = ((int) *(buffer + 16 + 4*n));
    leg = ((int) *(buffer + 17 + 4*n));

    emgsf = (((int) *(buffer + 18 + 4*n)) << 8) |
                                        (((int) *(buffer + 19 + 4*n)) & 0xFF);
    forcesf = (((int) *(buffer + 20 + 4*n)) << 8) |
                                        (((int) *(buffer + 21 + 4*n)) & 0xFF);
    msa = (((int) *(buffer + 22 + 4*n)) << 8) |
                                        (((int) *(buffer + 23 + 4*n)) & 0xFF);
    msd = (((int) *(buffer + 24 + 4*n)) << 8) |
                                        (((int) *(buffer + 25 + 4*n)) & 0xFF);
    fw = (((int) *(buffer + 26 + 4*n)) << 8) |
                                        (((int) *(buffer + 27 + 4*n)) & 0xFF);
    fl = ((int) *(buffer + 28 + 4*n));
    mw = (((int) *(buffer + 29 + 4*n)) << 8) |
                                        (((int) *(buffer + 30 + 4*n)) & 0xFF);
    mrl = ((int) *(buffer + 31 + 4*n));
    mbtmp = (((int) *(buffer + 32 + 4*n)) << 8) |
                                        (((int) *(buffer + 33 + 4*n)) & 0xFF);

    close (fd);          /* close the data file */
    cfree (buffer);

    return;
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       open_graph.c
*                       -> opens a graphics window for the calling process
*                               and returns an integer file descriptor.
*
*       -> copy this file to /tmp and compile to generate object file using
*                       /cpp/bin/fcc -c open_graph.c
*
*       -> to link with other object files, use
*                       /cpp/bin/fcc files.o open_graph.o -lc
*
*****************************************************************************/

#include <fcntl.h>      /* for opening the graphics window */
#include <scrn/smsysdep.h>
#include <scrn/plotem.h>        /* contains the ioctl() calls */
#include <scrn/wmcom.h>         /* contains the window structure */

int open_graph (xsize,ysize,xloc,yloc)
int xsize,ysize;        /* for full screen, xsize=512 & ysize=255 */
int xloc,yloc;  /* for upper left hand corner, xloc=0 & yloc=0 */
{
        int graph_fd;

        struct windio plot;     /* window structure for the graphics window */

        write (1, "\33&j@", 4);          /* turn off function key menu */

        ioctl (0,WMGET,&plot);  /* get the default window attributes */

        plot.w_type = PETYPE;  /* assign window as Plotter Emulator type */
        plot.w_stat |= AUTO_SH | AUTO_DEST;     /* autoshow and autodestroy */
               /* keep graphics window from being connected to the keyboard */
        plot.w_stat &= ~(AUTO_ACT | ACTIVE);
        plot.w_width = xsize;
        plot.w_height = ysize;  /* set size (in pixels) of the window */
        plot.w_xloc = xloc;
        plot.w_yloc = yloc;     /* set the location (in pixels) on screen */
        plot.w_gen1 = 0;
        plot.w_gen2 = 0;        /* use the default buffer size */

        strcat (plot.w_path,".plot");   /* optional suffix for window name */

        ioctl (0,WMCREATE,&plot);       /* create the window !  */

        graph_fd = open(plot.w_path,O_RDWR);    /* get file descriptor */

        write (graph_fd, "in; pg;", 7);   /* initialize and clear g. window */

        return (graph_fd);
}

/*****************************************************************************/
```

```
/*****************************************************************************/
/*
        plot.c
                -> plots data vector 'y' as a function of data vector 'x'
                        on a set of axes.

        -> copy this file to /tmp and compile to generate object file using
                        /cpp/bin/fcc -c plot.c

        -> to link with other object files, use
                        /cpp/bin/fcc other_files.o plot.o -lm

        variables :

                gw_fp ->  graphics window file pointer obtained from a call to
                                fopen in the calling process.

                n ->  number of data points in 'x' and 'y' to be plotted.

                l ->  contains extreme values of 'x' and 'y' data vectors for
                        the determination of plotting limits.
                        l[0] = xmin
                        l[1] = xmax
                        l[2] = ymin
                        l[3] = ymax
                        -> if xmax <= xmin then this subroutine will calculate
                                new values of xmax and xmin.
                        -> if ymax <= ymin then this subroutine will calculate
                                new values of ymax and ymin.

                px -> pointer to data vector 'x'.
                        (horizontal coordinates of data points)

                py -> pointer to data vector 'y'.
                        (vertical coordinates of data points)

                ttl -> 32 character title for the graph.

                xlbl -> 32 character label for the x-axis.

                ylbl -> 32 character label for the y-axis.

                                                                          */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>

plot(gw_fp,n,l,px,py,ttl,xlbl,ylbl)
FILE *gw_fp;
int n;
float l[4], *px, *py;
char *ttl, *xlbl, *ylbl;
{
   char *calloc();
   int intpow10();
   double pow();
   float min(), max(), xfac, yfac, xmean, ymean, xlrgfac, ylrgfac;
   int i, *pltx, *plty, xpow, ypow, xmin, xmax, ymin, ymax, xtic, ytic, xi, yi;

   pltx = (int *) calloc(n, sizeof(int));
   plty = (int *) calloc(n, sizeof(int));
   if ( pltx == 0  ||  plty == 0 )
   {
      printf("\n\nMemory allocation failure in Plot (pltx or plty).\n");
      return;
   }
```

```
fprintf (gw_fp, "in; pg;");  /* initialize and clear graphics window */
fprintf (gw_fp, "pu180,245; lb%32s\3;", ttl);        /* print title */
fprintf (gw_fp, "pu240,2; lb%32s\3;", xlbl);         /* x-axis label */
fprintf (gw_fp, "di0,1; pu10,50; lb%32s\3; di1,0;", ylbl); /* y-axis lbl */
fprintf (gw_fp, "pu68,30; pd490,30; pu70,28; pd70,240;"); /* draw axes */

    /* find the extreme data values */
if (l[0] >= l[1])
{
   l[0] = min(px,n);
   l[1] = max(px,n);
}
if (l[2] >= l[3])
{
   l[2] = min(py,n);
   l[3] = max(py,n);
}

   /* determine the scaling factors */
xpow = intpow10(l[1]-l[0]);
xfac = (float) pow(10.0, (double)(3-xpow));
ypow = intpow10(l[3]-l[2]);
yfac = (float) pow(10.0, (double)(3-ypow));

   /* determine number spacing on the axes */
if ((l[1]-l[0])*xfac < 5000)
{
   xtic = 500;
   if ((l[1]-l[0])*xfac < 2000)
      xtic = 200;
}
else
   xtic = 1000;

if ((l[3]-l[2])*yfac < 5000)
{
   ytic = 500;
   if ((l[3]-l[2])*yfac < 2000)
      ytic = 200;
}
else
   ytic = 1000;

   /* translate and scale the extrema and data vectors for plotting */
xmean = (((float)xtic)/xfac) * (int)(xfac*(l[0]+l[1])/(2.0*((float)xtic)));
ymean = (((float)ytic)/yfac) * (int)(yfac*(l[2]+l[3])/(2.0*((float)ytic)));
xmin = xtic * (int) (-0.9995 + (l[0] - xmean) * xfac / ((float) xtic));
xmax = xtic * (int) ( 0.9995 + (l[1] - xmean) * xfac / ((float) xtic));
ymin = ytic * (int) (-0.9995 + (l[2] - ymean) * yfac / ((float) ytic));
ymax = ytic * (int) ( 0.9995 + (l[3] - ymean) * yfac / ((float) ytic));
for (i=0; i<n; i++)
{
   *(pltx + i) = (int) (( *(px + i) - xmean) * xfac);
   *(plty + i) = (int) (( *(py + i) - ymean) * yfac);
}

    /* scale the plotting region */
fprintf (gw_fp, "ip70,30,490,240; sc%d,%d,%d,%d; si;", xmin,xmax,ymin,ymax);

   /* mark and number the axes */
xpow = (l[1]*l[1] > l[0]*l[0]) ? intpow10(l[1]): intpow10(l[0]);
xlrgfac = (float) pow(10.0, (double)(-xpow));
for (xi=xmin; xi<=xmax; xi+=xtic)
   fprintf (gw_fp, "pu%d,%d; xt; cp-2.5,-1.3; lb%5.2f\3;", xi, ymin,
                                    (xmean + ((float) xi)/xfac)*xlrgfac);
ypow = (l[3]*l[3] > l[2]*l[2]) ? intpow10(l[3]): intpow10(l[2]);
ylrgfac = (float) pow(10.0, (double)(-ypow));
for (yi=ymin; yi<=ymax; yi+=ytic)
```

```
      fprintf (gw_fp, "pu%d,%d; yt; cp-6.0,-0.3; lb%5.2f\3;", xmin,yi,
                                        (ymean + ((float) yi)/yfac)*ylrgfac);

       /* plot the data */
   fprintf(gw_fp, "pu%d,%d; pd;", *pltx,*plty); /* move pen to first point */
   for (i=1; i<n; i++)
   {
      fprintf(gw_fp, "pa%d,%d;", *(pltx+i),*(plty+i));  /* plot data point */
   }

      /* print scale power factors */
   fprintf(gw_fp, "sc;");       /* change from user units to graphics units */
   if (xpow != 0)
      fprintf(gw_fp, "pu240,2; cp34,0; lbx10\3; cp0,0.5; lb%-3d\3;", xpow);
   if (ypow != 0)
   {
      fprintf(gw_fp, "di0,1; pu10,50; cp26,-2.4; lbx10\3; cp0,0.5;");
      fprintf(gw_fp, "lb%-3d\3; di1,0;", ypow);
   }

   fflush (gw_fp);
   cfree (pltx);
   cfree (plty);

   return;
}
/*****************************************************************************/

float min(pt,n)
int n;
float *pt;
{
   int i;
   float min;
   min = *pt;
   for (i=1; i<n; i++)
   {
      if (*(pt+i) < min)
         min = *(pt+i);
   }
   return(min);
}
/*****************************************************************************/

float max(pt,n)
int n;
float *pt;
{
   int i;
   float max;
   max = *pt;
   for (i=1; i<n; i++)
   {
      if (*(pt+i) > max)
         max = *(pt+i);
   }
   return(max);
}
/*****************************************************************************/

intpow10(z)
float z;
{
   int xpnt;
   double log10();

   if (z == 0.0 )
      return (0);
```

```
  if (z < 0.0 )
     z = -z;

  if (z < 1.0 )
     xpnt = (int) (log10((double) z) - 1.0);
  else
     xpnt = (int) log10((double) z);

  if ( (((float) log10((double) z)) - (float) xpnt) > 0.9999 )
     xpnt++;

  return(xpnt);
}
/*****************************************************************************/
```

```
/*****************************************************************************/
/*
        plot_prep.c
                    -> this programme sets parameters so that a portion of
                          a graph may be inspected more closely.

        -> copy this file to /tmp and compile to generate an object file using
                        /cpp/bin/fcc -c plot_prep.c

        -> to link with other required object files, use
                /cpp/bin/fcc other_files.o plot_prep.o
                                                                          */
/*****************************************************************************/

#include <stdio.h>

plot_prep (gw_fp,n,x,y,ttl,xlbl,ylbl)
FILE *gw_fp;
int n;
float *x, *y;
char *ttl, *xlbl, *ylbl;
{
   extern char *klbl[];
```

```
int i,fkey,nn,noff=0;
float lim[4];

   /* define special function key labels */
klbl[1] = "                ";
klbl[2] = "  Top     half  ";
klbl[3] = "Vertical  mag.  ";
klbl[4] = " Bottom   half  ";
klbl[5] = "  Left    side  ";
klbl[6] = " Middle expanded";
klbl[7] = " Right    side  ";
klbl[8] = "  Plot    menu  ";

lim[0]=0.0; lim[1]=0.0; lim[2]=0.0; lim[3]=0.0;
nn = n;

while (1)
{

   plot(gw_fp,nn,lim,(x+noff),(y+noff),ttl,xlbl,ylbl); /* plot the data */

   sleep(4);

   fkey_lbl ();      /* write the special function key labels */
   fkey = fkey_scan();       /* read function keys */
   write (1, "\33&ja", 4);   /* turn off menu */

   switch (fkey)
   {
      case 1 :
         break;
      case 2 :
         lim[2] = (lim[2] + lim[3]) * 0.5 ;
         break;
      case 3 :
         lim[2] = 0.25 * (3.0*lim[2] + lim[3]) ;
         lim[3] = 0.25 * (3.0*lim[3] + lim[2]) ;
         break;
      case 4 :
         lim[3] = (lim[2] + lim[3]) * 0.5 ;
         break;
      case 5 :
         nn /= 2 ;
         lim[1] = (lim[0] + lim[1]) * 0.5;
         lim[2] = 0.0 ;    lim[3] = 0.0 ;
         break;
      case 6 :
         nn /= 2 ;
         noff += nn/2 ;
         lim[0] = 0.25 * (3.0*lim[0] + lim[1]) ;
         lim[1] = 0.25 * (3.0*lim[1] + lim[0]) ;
         lim[2] = 0.0 ;    lim[3] = 0.0 ;
         break;
      case 7 :
         nn /= 2 ;
         noff += nn ;
         lim[0] = (lim[0] + lim[1]) * 0.5;
         lim[2] = 0.0 ;    lim[3] = 0.0 ;
         break;
      case 8 :
         fprintf (gw_fp, "pg;");
         fflush (gw_fp);
         return;
         break;
   }
 }
}
/*****************************************************************************/
```

```
/*****************************************************************************/
/*
        smda_graphic.c
                        -> this programme sets up parameters for the plotting
                                of the emg, force, and mmg data vectors and
                                their Fourier transforms.

        -> copy this file to /tmp and compile to generate object file using
                        /cpp/bin/fcc -c smda_graphic.c

        -> to link with other required object files, use
                /cpp/bin/fcc other_files.o smda_graphic.o -lm
                                                                          */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>

smda_graphics ()
{
   extern char path[], *klbl[];
   extern short *emg,*force,*mmg;
   extern int gfd, fkey;
   extern int n,r;

   FILE *gw_fp, *fdopen();
   char *calloc();
   int i,start,num;
   float *x, *y, *fft_dat, dx;

      /* set up graph labels */
   char *ttl = "                                    ";
   char *xlbl= "                                    ";
   char *ylbl= "                                    ";
   for (i=0; i<18; i++)
      *(ttl+4+i) = path[i];

   if ((gw_fp = fdopen(gfd, "r+")) == 0)
   {
      printf ("\n\nFailure to open graphics.window in 'ida_graphics.c'.");
      return;
   }

   x = (float *) calloc(n, sizeof(float));
   y = (float *) calloc(n, sizeof(float));
   fft_dat = (float *) calloc(2*n, sizeof(float));
   if ( x == 0  ||  y == 0  ||  fft_dat == 0 )
   {
      printf ("Memory allocation failure in 'ida_graphics'.");
      return;
   }

   while (1)
   {
        /* define special function key labels */
      klbl[1] = "  Plot    EMG   ";
      klbl[2] = "  Plot   Force  ";
      klbl[3] = "  Plot    MMG   ";
      klbl[4] = "                ";
      klbl[5] = "  EMG     FFT   ";
      klbl[6] = " Force    FFT   ";
      klbl[7] = "  MMG     FFT   ";
      klbl[8] = "  Main    menu  ";

      fkey_lbl ();      /* write the special function key labels */
      fkey = fkey_scan();       /* read function keys */
      write (1, "\33&j@", 4);   /* turn off menu */
```

```
if (fkey == 8)
{
   cfree (x);
   cfree (y);
   cfree (fft_dat);
   return;
}

if (fkey < 4 )
{
   xlbl = "      TIME    (milliseconds)    ";
   dx = 1000.0 / (float) r ;
   for (i=0; i<n; i++)
      *(x+i) = dx * (float) i ;

   switch (fkey)
   {
      case 1 :
         ylbl = " EMG AMPLITUDE    (ADC integer) ";
         for (i=0; i<n; i++)
            *(y+i) = (float) *(emg + i) ;
         break;
      case 2 :
         ylbl = " Force AMPLITUDE  (ADC integer) ";
         for (i=0; i<n; i++)
            *(y+i) = (float) *(force + i) ;
         break;
      case 3 :
         ylbl = "MMG AMPLITUDE  (DSQ-400 integer)";
         for (i=0; i<n; i++)
            *(y+i) = (float) *(mmg + i) ;
         break;
   }

   plot_prep (gw_fp,n,x,y,ttl,xlbl,ylbl); /* plot the data! */
}

if (fkey > 4)
{
   printf("\n\nEnter the starting point (sample number) of the FFT ");
   printf("in the data vector \nand the number of samples to be ");
   printf("transformed (must be a power of 2):");
   printf("\nint [Return]\nint [Return]\n");
   scanf("%d", &start);
   scanf("%d", &num);
   printf("\nPlease Wait.  FFT calculations in progress.");
   if ((start+num) > n)
   {
      start=0; num=n;
   }

   xlbl = "      FREQUENCY      ( Hz )     ";
   for (i=0; i<num; i++)
      *(fft_dat + 2*i+1) = 0.0;

   switch (fkey)
   {
      case 5 :
         ylbl = "  EMG FFT MAGNITUDE  (integer)  " ;
         for (i=0; i<num; i++)
            *(fft_dat + 2*i) = (float) *(emg + start+i);
         break;
      case 6 :
         ylbl = " FORCE FFT MAGNITUDE  (integer) " ;
         for (i=0; i<num; i++)
            *(fft_dat + 2*i) = (float) *(force + start+i);
         break;
      case 7 :
```

```
                ylbl = "  MMG FFT MAGNITUDE  (integer)  " ;
                for (i=0; i<num; i++)
                   *(fft_dat + 2*i) = (float) *(mmg + start+i);
                break;
             }

             cfft (fft_dat,num,1);

             dx = ((float) r) / ((float) num) ;
             for (i=0; i<num/2; i++)
             {
                *(y + i) = (float) sqrt((double) (*(fft_dat + 2*i) *
                     *(fft_dat + 2*i) + *(fft_dat + 2*i+1) * *(fft_dat + 2*i+1)));
                *(x + i) = dx * (float) i ;
             }
             printf("\015                                                    \n");

             plot_prep (gw_fp,num/2,x,y,ttl,xlbl,ylbl); /* plot the data! */
          }
       }
    }
/*****************************************************************************/
```

APPENDIX C

DATA ANALYSIS SOURCE CODE

A set of C language programmes for analysis of the stored data records is presented in this appendix. Each programme performed a single function. The programmes were run interactively or from a shell command programme. The following programmes are presented in alphabetical order in the following pages:

asciihpltreg
btasii
cfft
cgw
efdhplt
efdtr
ffthplt
mmgmdfy
plot.

It should be noted that the above programmes that involve plotting ("plt" appears in their names) were written to drive a Hewlett-Packard 7470A plotter. Previous versions of these programmes (not listed in this document) were written to drive a graphics window (plotter emulator window).

Two examples of shell command programmes have been included at the end of these listings to show how the data analysis programmes, listed above, were used:

posexan
reptest.

```
/*****************************************************************************
*
*       asciihpltreg.c  ->  Hard-Plot with Liner Regression
*                                 |
*                                 - for HP-7470 plotter
*
*                       -> this programme plots a graph of the n-point data
*                          vector of an ASCII file of the following form:
*
*                                  graph title (32 characters)
*                                  x-axis label (32 characters)
*                                  y-axis label (32 characters)
*                                  n (number of data points)
*                                  x1 y1
*                                  x2 y2
*                                  x3 y3
*                                  .  .
*                                  .  .
*                                  .  .
*                                  xn yn
*
*                       -> a straight line is fitted to the data using the
*                          method of least-squares.
*
*       -> parameters :
*
*               -> required :   argv[1] -> name of source data file (on disc).
*
*               -> optional :   argv[2] -> xmin \
*                                                |
*                               argv[3] -> xmax  |
*                                                |- plot scale limits.
*                               argv[4] -> ymin  |
*                                                |
*                               argv[5] -> ymax /
*
*                               argv[6] -> pen speed
*                                          (integer between 0 and 100)
*                                          -> default value is 5.
*
*                               argv[7]  -> w1x -> horizontal distance of
*                                          bottom left corner of plot
*                                          from left edge of paper.
*                                          (float) from 0.5 to 10.5 inches
*                                          -> default value is 0.5 inches.
*
*                               argv[8] -> w1y -> vertical distance of
*                                          bottom left corner of plot
*                                          from bottom edge of paper.
*                                          (float) from 0.25 to 7.75 inches
*                                          -> default value is 0.5 inches.
*
*                               argv[9] -> w2x  -> horizontal distance of
*                                          top right corner of plot
*                                          from left edge of paper.
*                                          (float) from 0.5 to 10.5 inches
*                                          -> default value is 10.5 inches.
*
*                               argv[10] -> w2y -> vertical distance of
*                                          top right corner of plot
*                                          from bottom edge of paper.
*                                          (float) from 0.25 to 7.75 inches
*                                          -> default value is 5.5 inches.
*
*
*       -> compile to generate executable code using:
*
```

```
*                       cc asciihpltreg.c -lm -o asciihpltreg
*               or
*                       /cpp/bin/fcc asciihpltreg.c -lm
*
*****************************************************************************/

#include <stdio.h>

main (argc, argv)
int argc;
char *argv[];
{

   FILE *gw_fp, *d_fp, *fopen();
   char *calloc(), ttl[33], xlbl[33], ylbl[33];
   int n, i;
   int ips, w1x=0, w1y=252, w2x=10080, w2y=5292, p1x, p1y, p2x, p2y;
   float ps=30.4, psf=0.38, wtmp[4];
   float *x, *y, lim[4];

      /* test to see if data file name argument was given */

   if ( strlen(argv[1]) == 0 )
   {
      printf ("\n\nERROR!\nThe name of the source data file must be provided ");
      printf ("as an argument \nwith the command %s.\n", argv[0]);
      exit (1);
   }

   lim[0] = 0.0;   lim[1] = 0.0;   lim[2] = 0.0;   lim[3] = 0.0;

        /* consider optional command parameters */

   switch (argc)
   {
      case 11:
         sscanf (argv[7], "%f", &wtmp[0]);
         sscanf (argv[8], "%f", &wtmp[1]);
         sscanf (argv[9], "%f", &wtmp[2]);
         sscanf (argv[10], "%f", &wtmp[3]);
         w1x = (int) (1008.0 * (wtmp[0]-0.5));
         w1y = (int) (1008.0 * (wtmp[1]-0.25));
         w2x = (int) (1008.0 * (wtmp[2]-0.5));
         w2y = (int) (1008.0 * (wtmp[3]-0.25));

      case 7:
         sscanf (argv[6], "%d", &ips);
         ps = psf * (float) ips;

      case 6:
         sscanf (argv[2], "%f", &lim[0]);
         sscanf (argv[3], "%f", &lim[1]);
         sscanf (argv[4], "%f", &lim[2]);
         sscanf (argv[5], "%f", &lim[3]);
         break;
   }

      /* open the data file and plotter device file */

   if ((gw_fp = fopen("/dev/plotter", "r+")) == 0)
   {
      printf ("\n\nFailure to open graphics window in 'asciipltreg'.");
      exit (1);
   }
   if ( (d_fp = fopen(argv[1], "r")) == 0 )
   {
      printf ("\n\nFailure to open ASCII file '%s' in asciipltreg.", argv[1]);
      exit (1);
```

```
}

   /* locate and scale the plotting region */

fprintf (gw_fp, "IN; VS %f; IW %d,%d,%d,%d; SP 1;", ps,w1x,w1y,w2x,w2y);
p1x = w1x + (int) (0.13671875 * (w2x-w1x));
p1y = w1y + (int) (0.11764706 * (w2y-w1y));
p2x = w1x + (int) (0.95703125 * (w2x-w1x));
p2y = w1y + (int) (0.94117647 * (w2y-w1y));
fprintf (gw_fp, "IP %d,%d,%d,%d;", p1x,p1y,p2x,p2y);
fprintf (gw_fp, "SC 137,957,118,941;");

   /* read title, labels and n from the data file */

fscanf (d_fp, "%[^\n]\n%[^\n]\n%[^\n]\n%d", ttl, xlbl, ylbl, &n);

   /* allocate memory for floating-point data vectors */

x = (float *) calloc (n, sizeof(float));
y = (float *) calloc (n, sizeof(float));
if ( (x == 0)  ||  (y == 0) )
{
   printf ("Memory allocation failure for x or y in 'asciipltreg'.");
   exit (1);
}

   /* read data from the file */

for (i=0; i<n; i++)
   fscanf (d_fp, "%f%f", (x+i), (y+i));

     /* plot the data! */

plotreg (gw_fp,n,lim,x,y,ttl,xlbl,ylbl);

     /* tidy up memory and files */

cfree (x);
cfree (y);

fclose (gw_fp);
fclose (d_fp);

exit (0);
}
/*****************************************************************************
*
*    plot.c
*           -> plots floating-point data vector 'y' as a function of
*                 floating-point data vector 'x' on a set of axes.
*
*      -> compile to generate object file using:
*           cc -c plot.c    or      /cpp/bin/fcc -c plot.c
*
*      -> to link with other object files, use
*        cc other_files.o plot.o -lm  or  /cpp/bin/fcc other_files.o plot.o -lm
*
*      variables :
*
*           gw_fp ->  graphics window file pointer obtained from a call to
*                        fopen in the calling process.
*
*           n ->  number of data points in 'x' and 'y' to be plotted.
*
*           l ->  contains extreme values of 'x' and 'y' data vectors for
*                   the determination of plotting limits.
*                   l[0] = xmin
*                   l[1] = xmax
```

```
*                       l[2] = ymin
*                       l[3] = ymax
*                       -> if xmax <= xmin then this subroutine will calculate
*                              new values of xmax and xmin.
*                       -> if ymax <= ymin then this subroutine will calculate
*                              new values of ymax and ymin.
*
*               px -> pointer to data vector 'x' of type 'float'.
*                       (horizontal coordinates of data points)
*
*               py -> pointer to data vector 'y' of type 'float'.
*                       (vertical coordinates of data points)
*
*               ttl -> 32 character title for the graph.
*
*               xlbl -> 32 character label for the x-axis.
*
*               ylbl -> 32 character label for the y-axis.
*
*****************************************************************************/

#include <stdio.h>
#include <math.h>

plotreg(gw_fp,n,l,px,py,ttl,xlbl,ylbl)
FILE *gw_fp;
int n;
float l[4], *px, *py;
char *ttl, *xlbl, *ylbl;
{
   char *calloc();
   int intpow10(), rint();
   double pow();
   float min(), max(), xfac, yfac, xmean, ymean, xlrgfac, ylrgfac;
   int i, *pltx, *plty, xpow, ypow, xmin, xmax, ymin, ymax, xtic, ytic, xi, yi;
   float sx=0.0, sy=0.0, sxy=0.0, sx2=0.0, sy2=0.0, m, b, r2;
   int yfitmin, yfitmax;

        /* allocate memory for data vectors */
   pltx = (int *) calloc(n, sizeof(int));
   plty = (int *) calloc(n, sizeof(int));
   if ( pltx == 0  ||  plty == 0 )
   {
      printf("\n\nMemory allocation failure in 'plotreg' (pltx or plty).\n");
      return;
   }

        /* perform linear regression and print values of 'm' and 'b' */
   for (i=0; i<n; i++)
   {
      sx += *(px+i);
      sy += *(py+i);
      sxy += *(px+i) * *(py+i);
      sx2 += *(px+i) * *(px+i);
      sy2 += *(py+i) * *(py+i);
   }

   m = (((float) n)*sxy - sx*sy) / (((float) n)*sx2 - sx*sx);
   b = (sy*sx2 - sx*sxy) / (((float) n)*sx2 - sx*sx);
   r2 = m * (((float) n)*sxy - sx*sy) / (((float) n)*sy2 - sy*sy);
   printf ("\n\n\nThe fitted-line slope is %g and the y-intercept is ", m);
   printf ("%g .\nThe coefficient of determination is %f .\n\n\n", b, r2);

        /* find the extreme data values */
   if (l[0] >= l[1])
   {
      l[0] = min(px,n);
      l[1] = max(px,n);
```

```
}
if (l[2] >= l[3])
{
   l[2] = min(py,n);
   l[3] = max(py,n);
}

   /* determine the scaling factors */
xpow = intpow10(l[1]-l[0]);
xfac = (float) pow(10.0, (double)(3-xpow));
ypow = intpow10(l[3]-l[2]);
yfac = (float) pow(10.0, (double)(3-ypow));

   /* determine number spacing on the axes */
if ((l[1]-l[0])*xfac < 5000)
{
   xtic = 500;
   if ((l[1]-l[0])*xfac < 2000)
      xtic = 200;
}
else
   xtic = 1000;

if ((l[3]-l[2])*yfac < 5000)
{
   ytic = 500;
   if ((l[3]-l[2])*yfac < 2000)
      ytic = 200;
}
else
   ytic = 1000;

   /* translate and scale the extrema and data vectors for plotting */
xmean = (((float)xtic)/xfac) * (int)(xfac*(l[0]+l[1])/(2.0*((float)xtic)));
ymean = (((float)ytic)/yfac) * (int)(yfac*(l[2]+l[3])/(2.0*((float)ytic)));
xmin = xtic * (int) (-0.9995 + (l[0] - xmean) * xfac / ((float) xtic));
xmax = xtic * (int) ( 0.9995 + (l[1] - xmean) * xfac / ((float) xtic));
ymin = ytic * (int) (-0.9995 + (l[2] - ymean) * yfac / ((float) ytic));
ymax = ytic * (int) ( 0.9995 + (l[3] - ymean) * yfac / ((float) ytic));
for (i=0; i<n; i++)
{
   *(pltx + i) = (int) (( *(px + i) - xmean) * xfac);
   *(plty + i) = (int) (( *(py + i) - ymean) * yfac);
}
yfitmin = (int) ((m * (xmean + xmin/xfac) + b - ymean) * yfac);
yfitmax = (int) ((m * (xmean + xmax/xfac) + b - ymean) * yfac);

    /* print labels and scale power factors */
fprintf (gw_fp, "sr1.2,3.6;");  /* set the character size */
fprintf (gw_fp, "pu352,961; lb%-32s\3;", ttl);    /* print title */
xpow = (l[1]*l[1] > l[0]*l[0]) ? intpow10(l[1]): intpow10(l[0]);
ypow = (l[3]*l[3] > l[2]*l[2]) ? intpow10(l[3]): intpow10(l[2]);
fprintf (gw_fp, "pu320,8; lb%-32s\3;", xlbl);     /* print x-axis label */
if ( (xpow < 0) || (xpow > 3) )
   fprintf(gw_fp, "cp2,0; lbx10\3; cp0,0.4; sr1.0,3.0; lb%-3d\3;", xpow);
fprintf (gw_fp, "di0,1; sr1.2,3.6;");
fprintf (gw_fp, "pu25,50; lb%-32s\3;", ylbl);    /* print y-axis label */
if ( (ypow < 0) || (ypow > 3) )
   fprintf(gw_fp,"pu65,810; lbx10\3; cp0,0.4; sr1.0,3.0; lb%-3d\3;",ypow);

    /* scale the plotting region and draw the axes */
fprintf (gw_fp, "di1,0; sr1.0,3.0; sc%d,%d,%d,%d;", xmin,xmax,ymin,ymax);
fprintf (gw_fp, "pu%d,%d; pd%d,%d; pd%d,%d;", xmin,ymax,xmin,ymin,xmax,ymin);

   /* mark and number the axes */
xlrgfac = (float) pow(10.0, (double)(-xpow));
fprintf (gw_fp, "tl0.94,0.94;");
switch (xpow)
```

```
{
   case 1:
      for (xi=xmin; xi<=xmax; xi+=xtic)
         fprintf (gw_fp, "pu%d,%d; xt; cp-3.2,-1.0; lb%5.1f\3;", xi, ymin,
                                    (xmean + ((float) xi)/xfac)*xlrgfac*10.0);
      break;
   case 2:
      for (xi=xmin; xi<=xmax; xi+=xtic)
         fprintf (gw_fp, "pu%d,%d; xt; cp-1.2,-1.0; lb%2d\3;", xi, ymin,
                        (rint ((xmean + ((float) xi)/xfac)*xlrgfac*100.0)));
      break;
   case 3:
      for (xi=xmin; xi<=xmax; xi+=xtic)
         fprintf (gw_fp, "pu%d,%d; xt; cp-1.2,-1.0; lb%2d\3;", xi, ymin,
                       (rint ((xmean + ((float) xi)/xfac)*xlrgfac*1000.0)));
      break;
   default:
      for (xi=xmin; xi<=xmax; xi+=xtic)
         fprintf (gw_fp, "pu%d,%d; xt; cp-2.2,-1.0; lb%5.2f\3;", xi, ymin,
                                       (xmean + ((float) xi)/xfac)*xlrgfac);
      break;
}
ylrgfac = (float) pow(10.0, (double)(-ypow));
fprintf (gw_fp, "tl0.47,0.47;");
switch (ypow)
{
   case 1:
      for (yi=ymin; yi<=ymax; yi+=ytic)
         fprintf (gw_fp, "pu%d,%d; yt; cp-6.0,-0.3; lb%5.1f\3;", xmin, yi,
                                    (ymean + ((float) yi)/yfac)*ylrgfac*10.0);
      break;
   case 2:
      for (yi=ymin; yi<=ymax; yi+=ytic)
         fprintf (gw_fp, "pu%d,%d; yt; cp-6.0,-0.3; lb%5d\3;", xmin, yi,
                        (rint ((ymean + ((float) yi)/yfac)*ylrgfac*100.0)));
      break;
   case 3:
      for (yi=ymin; yi<=ymax; yi+=ytic)
         fprintf (gw_fp, "pu%d,%d; yt; cp-6.0,-0.3; lb%5d\3;", xmin, yi,
                       (rint ((ymean + ((float) yi)/yfac)*ylrgfac*1000.0)));
      break;
   default:
      for (yi=ymin; yi<=ymax; yi+=ytic)
         fprintf (gw_fp, "pu%d,%d; yt; cp-6.0,-0.3; lb%5.2f\3;", xmin, yi,
                                       (ymean + ((float) yi)/yfac)*ylrgfac);
      break;
}

    /* plot the data */
fprintf(gw_fp, "sm";");      /* select the data-point character */
fprintf(gw_fp, "pu%d,%d;", *pltx,*plty); /* move pen to first point */
for (i=1; i<n; i++)
{
   fprintf(gw_fp, "pa%d,%d;", *(pltx+i),*(plty+i));  /* plot data point */
}

    /* plot regression line */
fprintf(gw_fp, "sm;pu%d,%d;", xmin, yfitmin); /* pen to beginning of line */
fprintf(gw_fp, "pd%d,%d;", xmax, yfitmax); /* move pen to end of line */

fprintf(gw_fp, "pu%d,%d; sp;", xmin,ymin);

fflush (gw_fp);
cfree (pltx);
cfree (plty);

return;
}
```

```
/*****************************************************************************/

float min(pt,n)
int n;
float *pt;
{
   int i;
   float min;
   min = *pt;
   for (i=1; i<n; i++)
   {
      if (*(pt+i) < min)
         min = *(pt+i);
   }
   return(min);
}
/*****************************************************************************/

float max(pt,n)
int n;
float *pt;
{
   int i;
   float max;
   max = *pt;
   for (i=1; i<n; i++)
   {
      if (*(pt+i) > max)
         max = *(pt+i);
   }
   return(max);
}
/*****************************************************************************/

intpow10(z)
float z;
{
   int xpnt;
   double log10();

   if (z == 0.0 )
      return (0);

   if (z < 0.0 )
      z = -z;

   if (z < 1.0 )
      xpnt = (int) (log10((double) z) - 1.0);
   else
      xpnt = (int) log10((double) z);

   if ( (((float) log10((double) z)) - (float) xpnt) > 0.9999 )
      xpnt++;

   return(xpnt);
}
/*****************************************************************************/

int rint(x)     /* return a Rounded INTeger (not truncated) */
float x;
{
   return ( (int) ( (x < 0.0) ? (x - 0.5) : (x + 0.5) ) );
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       btascii.c       -> Binary-To-ASCII
*
*                       -> this programme reads one of the RAM data files
*                               (/emg, /force, /mmg) and writes the data in
*                               ASCII form to the standard output.  A portion
*                               of the 2048-point data vector may be specified
*                               if printing of the whole vector is not desired.
*                               The first column of the output specifies the
*                               position in the data vector of the first
*                               number in that line (0 to 1023).
*
*       -> parameters :
*
*               -> required :   argv[1] -> name of source data file.
*
*               -> optional :   argv[2] -> position of data value in data vector
*                                       to start printing with.
*                                       (integer 0 to 2047)
*                               argv[3] -> number of data points to print.
*                                       (integer <= 2048)
*
*
*       -> compile to generate executable code using:
*               cc btascii.c -o btascii         or    /cpp/bin/fcc btascii.c
*
*****************************************************************************/

#define N_INIT  2048    /* number of elements in the data vector */

#include <stdio.h>

main (argc, argv)
int argc;
char *argv[4];
{

  FILE *d_fp, *fopen();
  char *calloc(), str1[50], str2[20];
  short *ybuf;
  int n=N_INIT, i, j, offset=0;
  float *fftdat;

     /* test to see if data file name argument was given */

  if ( strlen(argv[1]) == 0 )
  {
     printf ("\n\nERROR!\nThe name of the source data file must be provided ");
     printf ("as an argument \nwith the command btascii.\n");
     exit (1);
  }

       /* consider optional command parameters */

  if ( argc==4 )
  {
     sscanf (argv[2], "%d", &offset);
     sscanf (argv[3], "%d", &n);
  }
  if ( n+offset > N_INIT )
  {
     printf ("\nData vector length exceeded with argv[2] and argv[3]!\n");
     printf ("Default values of argv[2]=0 and argv[3]=N_INIT were used.\n");
     offset = 0 ;    n = N_INIT ;
  }

     /* open the data file */
```

```
   if ( (d_fp = fopen(argv[1], "r")) == 0 )
   {
      printf ("\n\nFailure to open ASCII RAM file in 'graph'.");
      exit (1);
   }

      /* allocate memory for integer buffer */

   ybuf = (short *) calloc (N_INIT, 2);
   fftdat = (float *) calloc (2*n, 4);
   if ( (ybuf == 0) || (fftdat == 0) )
   {
      printf ("Memory allocation failure in 'btascii'.");
      exit (1);
   }

      /* read data from the RAM file */

   if ( (strcmp("/fft", argv[1])) == 0)
   {
      fread (fftdat, 4, 2*(offset+n), d_fp);
      strcat (str1, "FFT Coefficients");
      strcat (str2, "(real imaginary)");
   }
   else
   {
      fread (ybuf, 2, N_INIT, d_fp);
      fscanf (d_fp, "%[^\n]\n%s", str1, str2);
   }

      /* print the data */

   printf ("\t\t\t\t%s  %s\n", str1,str2);

   if ( (strcmp("/fft", argv[1])) == 0)
   {
      for (i=offset; i<(offset+n); i+=4)
      {
         printf ("%4d    ", i);
         for (j=0; j<8; j++)
            printf ("%6.1f  ", *(fftdat+(2*i)+j));
         printf ("\n");
      }
   }
   else
   {
      for (i=offset; i<(offset+n); i+=8)
      {
         printf ("%4d    ", i);
         for (j=0; j<8; j++)
            printf ("%6d  ", *(ybuf+i+j));
         printf ("\n");
      }
   }

      /* tidy up memory and files */

   cfree (ybuf);
   cfree (fftdat);

   fclose (d_fp);

   exit (0);
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       cfft.c
*               -> Complex Fast Fourier Transform
*
*       -> this function replaces 'data' with its discrete Fourier transform
*               if isign=-1 or replaces 'data' with the inverse discrete
*               Fourier transform if 'isign'=+1.
*
*       -> 'data' is a real array of 2*nn elements with pairs of adjacent
*               elements representing the real and complex parts of each
*               data point.
*
*       -> 'nn' is the number of complex data points and must be an integer
*               power of 2.
*
*
*       -> copy this file to /tmp and compile to generate object file using
*               /cpp/bin/fcc -c cfft.c
*
*       -> to link with other required object files, use
*               /cpp/bin/fcc other_files.o cfft.o -lm
*
*****************************************************************************/

#include <math.h>

cfft (data,nn,isign)
float *data;
int nn,isign;
{
   double sin();
   int i,j,n,m,mmax,istep;
   float tempr,tempi,nninv;
   double wr,wi,wpr,wpi,wtemp,theta;

   n = nn * 2;
   j=1;

   for (i=1; i<=n; i+=2)                /* this is the BIT-reversal section */
   {
      if (j > i)
      {                                 /* exchange the two complex numbers */
         tempr = *(data + j-1);
         tempi = *(data + j);
         *(data + j-1) = *(data + i-1);
         *(data + j) = *(data + i);
         *(data + i-1) = tempr;
         *(data + i) = tempi;
      }

      m = nn;

      while ((m >= 2) && (j > m))
      {
         j -= m;
         m >>= 1;       /* divide m by 2 */
      }

      j += m;
   }

   mmax = 2;    /* here begins the Danielson-Lanczos section of the routine */

   while (n > mmax)     /* outer loop executed log2(nn) times */
   {
      istep = mmax << 1;        /* istep=mmax*2 */
      theta = 6.28318530717959 / (double)(isign * mmax);
```

```
        wpr = sin(0.5*theta);
        wpr *= wpr;
        wpr *= -2.0;
        wpi = sin(theta);
        wr = 1.0;
        wi = 0.0;

        for (m=1; m<=mmax; m+=2)  /* here are the two nested inner loops */
        {
           for (i=m; i<=n; i+=istep)
           {
              j = i + mmax;       /* this is the Danielson-Lanczos formula */
              tempr = (wr * *(data + j-1)) - (wi * *(data + j));
              tempi = (wr * *(data + j)) + (wi * *(data + j-1));
              *(data + j-1) = *(data + i-1) - tempr;
              *(data + j) = *(data + i) - tempi;
              *(data + i-1) += tempr;
              *(data + i) += tempi;
           }

           wtemp = wr;            /* trigonometric recurrence */
           wr = wr*wpr - wi*wpi + wr;
           wi = wi*wpr + wtemp*wpi + wi;
        }

        mmax = istep;
     }

     if (isign == -1)     /* scale data by 1/nn */
     {
        nninv = 1.0 / (float) nn;
        for (i=0; i<(nn*2); i++)
           *(data + i) *= nninv;
     }

     return;
  }
  /*****************************************************************************/
```

```
/*****************************************************************************
*
*       cgw.c   ->  Create Graphics Window
*
*                       -> creates a graphics window with name 'plot'.
*
*       -> compile to generate executable code using
*               /cc cgw.c -o cgw        or      /cpp/bin/fcc cgw.c
*
*****************************************************************************/

#define XSIZE  512
#define YSIZE  255      /* window will fill the screen */
#define XLOC  0
#define YLOC  0         /* upper left hand corner of window on screen */

#include <fcntl.h>      /* for opening the graphics window */
#include <scrn/smsysdep.h>
#include <scrn/plotem.h>        /* contains the ioctl() calls */
#include <scrn/wmcom.h>         /* contains the window structure */

main ()
{
   int xsize=XSIZE, ysize=YSIZE, xloc=XLOC, yloc=YLOC;
   struct windio plot;  /* window structure for the graphics window */

   write (1, " \033&j@", 5);    /* turn off softkey menu */

   ioctl (0,WMGET,&plot);       /* get the default window attributes */

   plot.w_type = PETYPE;        /* assign window as Plotter Emulator type */
   plot.w_stat &= -RECOVER ;    /* prevent window from being destroyed */
        /* keep graphics window from being connected to the keyboard */
   plot.w_stat &= ~(AUTO_ACT | ACTIVE);
   plot.w_width = xsize;
   plot.w_height = ysize;       /* set size (in pixels) of the window */
   plot.w_xloc = xloc;
   plot.w_yloc = yloc;  /* set the location (in pixels) on screen */
   plot.w_gen1 = 0;
   plot.w_gen2 = 0;     /* use the default buffer size */

   strcpy (plot.w_path, "/dev/screen/plot_window");    /* define window name */

   ioctl (0,WMCREATE,&plot);    /* create the window !  */

   exit (0);
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       efdhplt.c       -> Erindale Frog Data Hard Plot
*                                       ^
*                                       |
*                                       - for HP-7470 plotter
*
*                       -> this programme plots a graph of the 2048-point data
*                               vector of any of the following binary RAM files
*                               generated by 'efdtr':
*                                       /emg
*                                       /force
*                                       /mmg
*
*       -> parameters :
*
*               -> required :   argv[1] -> name of source data file (on disc).
*
*               -> optional :   argv[2] -> number of data value in data vector
*                                          to start plotting with.
*                               argv[3] -> number of data points to plot
*                                          (starting at argv[2]).
*
*                               argv[4] -> xmin \
*                                           |
*                               argv[5] -> xmax  |
*                                            |- plot scale limits.
*                               argv[6] -> ymin  |
*                                           |
*                               argv[7] -> ymax /
*
*                               argv[8] -> pen speed
*                                          (integer between 0 and 100)
*                                          -> default value is 80.
*
*                               argv[9]  -> w1x -> horizontal distance of
*                                          bottom left corner of plot
*                                          from left edge of paper.
*                                          (float) from 0.5 to 10.5 inches
*                                          -> default value is 0.5 inches.
*
*                               argv[10] -> w1y -> vertical distance of
*                                          bottom left corner of plot
*                                          from bottom edge of paper.
*                                          (float) from 0.25 to 7.75 inches
*                                          -> default value is 0.5 inches.
*
*                               argv[11] -> w2x -> horizontal distance of
*                                          top right corner of plot
*                                          from left edge of paper.
*                                          (float) from 0.5 to 10.5 inches
*                                          -> default value is 10.5 inches.
*
*                               argv[12] -> w2y -> vertical distance of
*                                          top right corner of plot
*                                          from bottom edge of paper.
*                                          (float) from 0.25 to 7.75 inches
*                                          -> default value is 5.5 inches.
*
*
*       -> compile to generate executable code using:
*               cc efdhplt.c -lm -o efdhplt    or    /cpp/bin/fcc efdhplt.c -lm
*
*****************************************************************************/

#define N_INIT  2048    /* number of elements in the data vector */
#define R_INIT  8192    /* data collection rate (samples/second) */
```

```
#include <stdio.h>

main (argc, argv)
int argc;
char *argv[];
{

   FILE *gw_fp, *dvfp, *pfp, *fopen();
   char *calloc(), str1[50], str2[20], ttl[33], xlbl[33], ylbl[33];
   short *ybuf, *param;
   int n=N_INIT, nn=N_INIT, r=R_INIT, fd, i, offset=0;
   int ips, w1x=0, w1y=252, w2x=10080, w2y=5292, p1x, p1y, p2x, p2y;
   float ps=30.4, psf=0.38, wtmp[4];
   float *x, *y, dx, lim[4], fac=0.0;

      /* test to see if data file name argument was given */

   if ( strlen(argv[1]) == 0 )
   {
      printf ("\n\nERROR!\nThe name of the source data file must be provided ");
      printf ("as an argument \nwith the command %s.\n", argv[0]);
      exit (1);
   }

   lim[0] = 0.0;   lim[1] = 0.0;   lim[2] = 0.0;   lim[3] = 0.0;

        /* consider optional command parameters */

   switch (argc)
   {
      case 13:
         sscanf (argv[9], "%f", &wtmp[0]);
         sscanf (argv[10], "%f", &wtmp[1]);
         sscanf (argv[11], "%f", &wtmp[2]);
         sscanf (argv[12], "%f", &wtmp[3]);
         w1x = (int) (1008.0 * (wtmp[0]-0.5));
         w1y = (int) (1008.0 * (wtmp[1]-0.25));
         w2x = (int) (1008.0 * (wtmp[2]-0.5));
         w2y = (int) (1008.0 * (wtmp[3]-0.25));

      case 9:
         sscanf (argv[8], "%d", &ips);
         ps = psf * (float) ips;

      case 8:
         sscanf (argv[4], "%f", &lim[0]);
         sscanf (argv[5], "%f", &lim[1]);
         sscanf (argv[6], "%f", &lim[2]);
         sscanf (argv[7], "%f", &lim[3]);

      case 4:
         sscanf (argv[2], "%d", &offset);
         sscanf (argv[3], "%d", &nn);
         if ( nn+offset > N_INIT )
         {
            printf ("Data vector length exceeded with argv[2] and argv[3]!\n");
            printf ("Default values of argv[2]=0 and argv[3]=N_INIT were ");
            printf ("used.\n");
            offset = 0 ;    nn = N_INIT ;
         }
         break;
   }

      /* open the data files and plotter device file */

   if ( (dvfp = fopen(argv[1], "r")) == 0 )
   {
      printf ("\n\nFailure to open %s in 'efdplt'.", argv[1]);
```

```
   exit (1);
}
if ( (pfp = fopen("/parameters", "r")) == 0 )
{
   printf ("\n\nFailure to open /parameters in 'efdplt'.");
   exit (1);
}
if ((gw_fp = fopen("/dev/plotter", "r+")) == 0)
{
   printf ("\n\nFailure to open graphics window in 'efdplt'.");
   exit (1);
}

   /* locate and scale the plotting region */

fprintf (gw_fp, "IN; VS %f; IW %d,%d,%d,%d; SP 1;", ps,w1x,w1y,w2x,w2y);
p1x = w1x + (int) (0.13671875 * (w2x-w1x));
p1y = w1y + (int) (0.11764706 * (w2y-w1y));
p2x = w1x + (int) (0.95703125 * (w2x-w1x));
p2y = w1y + (int) (0.94117647 * (w2y-w1y));
fprintf (gw_fp, "IP %d,%d,%d,%d;", p1x,p1y,p2x,p2y);
fprintf (gw_fp, "SC 137,957,118,941;");

   /* allocate memory for integer buffer and floating-point data vectors */

ybuf = (short *) calloc (n, 2);
param = (short *) calloc (32, 2);
x = (float *) calloc (nn, sizeof(float));
y = (float *) calloc (nn, sizeof(float));
if ( (ybuf == 0) || (param == 0) || (x == 0)  ||  (y == 0) )
{
   printf ("Memory allocation failure for ybuf, param, x or y in 'efdplt'.");
   exit (1);
}

   /* read data from RAM files */

fread (ybuf, 2, n, dvfp);
fscanf (dvfp, "%s%s", str1, str2);
fread (param, 2, 32, pfp);

   /* set up graph labels and the vertical scale factor */

strcpy (ttl, str1);
strcpy (xlbl, "          TIME  (ms)          ");
if ( (strcmp("/emg", argv[1])) == 0 )
{
   fac = ((float) *(param + 13)) * 0.001;
   strcpy (ylbl, "      EMG AMPLITUDE  (mV)     ");
}
if ( (strcmp("/force", argv[1])) == 0 )
{
   fac = ((float) *(param + 14)) * 0.001;
   strcpy (ylbl, "       TWITCH FORCE  (mN)     ");
}
if ( (strcmp("/mmg", argv[1])) == 0 )
{
   fac = 0.006134;
   strcpy (ylbl, "       MAF AMPLITUDE  (pT)     ");
}
if ( (strcmp("/mmgmod", argv[1])) == 0 )
{
   fac = 0.006134;
   strcpy (ylbl, "       MAF AMPLITUDE  (pT)     ");
}

   /* prepare for plotting */
```

```
    dx = 1000.0 / (float) r ;
    for (i=0; i<nn; i++)
    {
       *(x+i) = dx * ((float) (i - 1023 + offset)) ;
       *(y+i) = ( (float) *(ybuf+i+offset) ) * fac ;
    }

        /* plot the data! */

    plot (gw_fp,nn,lim,x,y,ttl,xlbl,ylbl);

        /* tidy up memory and files */

    cfree (ybuf);
    cfree (param);
    cfree (x);
    cfree (y);

    fclose (gw_fp);
    fclose (dvfp);

    exit (0);
}
/*************************************************************************/
```

```
/*************************************************************************
*
*       efdtr.c -> Erindale Frog Data To RAM
*
*               -> this programme reads the DSQ-400 'packed-character' data
*                       from a disc file and writes it to 4 binary RAM files:
*                                       /emg
*                                       /force
*                                       /mmg
*                                       /parameters
*
*       -> required parameter argv[1] is the name of the source data file
*               (disc data file).
*
*       -> compile to generate executable code using
*               cc efdtr.c -o efdtr     or      /cpp/bin/fcc efdtr.c
*
*************************************************************************/

#define N_INIT  2048    /* number of elements in each data vector */

#include <stdio.h>
#include <fcntl.h>

main (argc, argv)
```

```
int argc;
char *argv[2];
{
   FILE *efp, *ffp, *mfp, *pfp, *fopen();
   char *calloc(), *buffer;
   short *emg, *force, *mmg, *parameters;
   int n=N_INIT;
   int i, fd;

      /* test to see if data file name argument was given */

   if ( strlen(argv[1]) == 0 )
   {
      printf ("\n\nERROR!\nThe name of the source data file must be provided ");
      printf ("as an argument \nwith the command efdtr.\n");
      exit (1);
   }

      /* open the disc data file (read only mode) */

   if ((fd = open(argv[1], O_RDONLY)) == -1)
   {
      printf ("\n\nThe data file %s cannot be opened!\n", argv[1]);
      exit (1);
   }

      /* open RAM files */

   efp = fopen ("/emg", "w");
   ffp = fopen ("/force", "w");
   mfp = fopen ("/mmg", "w");
   pfp = fopen ("/parameters", "w");
   if ( (efp==NULL) || (ffp==NULL) || (mfp==NULL) || (pfp==NULL) )
   {
      printf ("\n\nFailure to open ASCII RAM files in efdtr!\n");
      exit (1);
   }

      /* allocate memory for character and integer buffers */

   buffer = calloc (64+4*n, 1);
   emg = (short *) calloc (n, 2);
   force = (short *) calloc (n, 2);
   mmg = (short *) calloc (n, 2);
   parameters = (short *) calloc (32, 2);
   if ( (buffer==0) || (emg==0) || (force==0) || (mmg==0) || (parameters==0) )
   {
      printf ("\n\nMemory allocation failure for buffers in efdtr!\n");
      exit (1);
   }

      /* read the data from the disk file */

   if ( read (fd, buffer, 64+4*n) != 64+4*n)
   {
      printf ("\n\nIncomplete read from disc in efdtr!\n");
   }

      /* unpack data from the character buffer and write to binary RAM files */

   for (i=0; i<n; i++)
   {
      *(emg + i) = (((short) *(buffer + 4*i)) & 0xFF) - (short) 128 ;
      *(force + i) = ((short) *(buffer + 1 + 4*i)) & 0x00FF ;
      *(mmg + i ) = (((short) *(buffer + 2 + 4*i)) << 8) |
                              (((short) *(buffer + 3 + 4*i)) & 0x00FF) ;
   }
   *(parameters) = (((short) *(buffer + 4*n)) << 8) |
```

```
                                        (((short) *(buffer + 1 + 4*n)) & 0xFF);
    *(parameters + 1) = (((short) *(buffer + 2 + 4*n)) << 8) |
                                        (((short) *(buffer + 3 + 4*n)) & 0xFF);
    *(parameters + 2) = (((short) *(buffer + 4 + 4*n)) << 8) |
                                        (((short) *(buffer + 5 + 4*n)) & 0xFF);
    *(parameters + 3) = (((short) *(buffer + 6 + 4*n)) << 8) |
                                        (((short) *(buffer + 7 + 4*n)) & 0xFF);
    *(parameters + 4) = (short) *(buffer + 8 + 4*n);
    *(parameters + 5) = (short) *(buffer + 9 + 4*n);
    *(parameters + 6) = (short) *(buffer + 10 + 4*n);
    *(parameters + 7) = (short) *(buffer + 11 + 4*n);
    *(parameters + 8) = (short) *(buffer + 12 + 4*n);
    *(parameters + 9) = (((short) *(buffer + 13 + 4*n)) << 8) |
                                        (((short) *(buffer + 14 + 4*n)) & 0xFF);
    *(parameters + 10) = (short) *(buffer + 15 + 4*n);
    *(parameters + 11) = (short) *(buffer + 16 + 4*n);
    *(parameters + 12) = (short) *(buffer + 17 + 4*n);
    *(parameters + 13) = (((short) *(buffer + 18 + 4*n)) << 8) |
                                        (((short) *(buffer + 19 + 4*n)) & 0xFF);
    *(parameters + 14) = (((short) *(buffer + 20 + 4*n)) << 8) |
                                        (((short) *(buffer + 21 + 4*n)) & 0xFF);
    *(parameters + 15) = (((short) *(buffer + 22 + 4*n)) << 8) |
                                        (((short) *(buffer + 23 + 4*n)) & 0xFF);
    *(parameters + 16) = (((short) *(buffer + 24 + 4*n)) << 8) |
                                        (((short) *(buffer + 25 + 4*n)) & 0xFF);
    *(parameters + 17) = (((short) *(buffer + 26 + 4*n)) << 8) |
                                        (((short) *(buffer + 27 + 4*n)) & 0xFF);
    *(parameters + 18) = (short) *(buffer + 28 + 4*n);
    *(parameters + 19) = (((short) *(buffer + 29 + 4*n)) << 8) |
                                        (((short) *(buffer + 30 + 4*n)) & 0xFF);
    *(parameters + 20) = (short) *(buffer + 31 + 4*n);
    *(parameters + 21) = (((short) *(buffer + 32 + 4*n)) << 8) |
                                        (((short) *(buffer + 33 + 4*n)) & 0xFF);

      /* write the buffered binary data to the RAM files */

    fwrite (emg, 2, n, efp);
    fwrite (force, 2, n, ffp);
    fwrite (mmg, 2, n, mfp);
    fwrite (parameters, 2, 32, pfp);

      /* write disc file name and data vector name to RAM files */

    fprintf (efp, "%s\nEMG\n", argv[1]);
    fprintf (ffp, "%s\nForce\n", argv[1]);
    fprintf (mfp, "%s\nMAF\n", argv[1]);
    fprintf (pfp, "%s\nParameters\n", argv[1]);

      /* tidy up files and memory */

    close (fd);
    fclose (efp);
    fclose (ffp);
    fclose (mfp);
    fclose (pfp);

    cfree (buffer);
    cfree (emg);
    cfree (force);
    cfree (mmg);
    cfree (parameters);

    exit (0);
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       ffthplt.c       -> Fast Fourier Transform and Hard-Plotting programme.
*                                                       |
*                                                       - for HP-7470 plotter
*
*               -> this programme calculates the Fourier Transform of
*                       a specified portion of a 2048-element data
*                       vector from one of the binary RAM files (/emg,
*                       /force, /mmg) and plots the resulting power
*                       spectrum.  The complex Fourier coefficients are
*                       written to the binary RAM file /fft.
*
*
*       -> parameters :
*
*               -> required :   argv[1] -> name of source data file.
*                               (if argv[1]="/fft" then these Fourier
*                                coefficients will be plotted.
*                                 -> no FFT new calculations )
*
*               -> optional :   argv[2] -> starting position number of data
*                               value in vector for FFT calculation.
*                               argv[3] -> number of data points to be trans-
*                               formed (starting at argv[2]).
*                               -> must be an integer power of 2.
*
*                               argv[4] -> xmin \
*                                                |
*                               argv[5] -> xmax  |
*                                                |- plot scale limits.
*                               argv[6] -> ymin  |
*                                                |
*                               argv[7] -> ymax /
*
*                               argv[8] -> pen speed
*                                               (integer between 0 and 100)
*                                               -> default value is 80.
*
*                               argv[9]  -> w1x -> horizontal distance of
*                                               bottom left corner of plot
*                                               from left edge of paper.
*                                               (float) from 0.5 to 10.5 inches
*                                               -> default value is 0.5 inches.
*
*                               argv[10] -> w1y -> vertical distance of
*                                               bottom left corner of plot
*                                               from bottom edge of paper.
*                                               (float) from 0.25 to 7.75 inches
*                                               -> default value is 0.5 inches.
*
*                               argv[11] -> w2x -> horizontal distance of
*                                               top right corner of plot
*                                               from left edge of paper.
*                                               (float) from 0.5 to 10.5 inches
*                                               -> default value is 10.5 inches.
*
*                               argv[12] -> w2y -> vertical distance of
*                                               top right corner of plot
*                                               from bottom edge of paper.
*                                               (float) from 0.25 to 7.75 inches
*                                               -> default value is 5.5 inches.
*
*
*       -> compile to generate executable code using:
*               cc ffthplt.c -lm -o ffthplt     or     /cpp/bin/fcc ffthplt.c -lm
*
```

```
*****************************************************************************/

#define N_INIT  2048    /* number of elements in the data vector */
#define R_INIT  8192    /* data collection rate (samples/second) */

#include <stdio.h>

main (argc, argv)
int argc;
char *argv[];
{

   FILE *gw_fp, *dvfp, *fftfp, *pfp, *fopen();
   char *calloc(), str1[33], str2[20], str3[22], ttl[33], xlbl[33], ylbl[33];
   short *ybuf, *param;
   int n=N_INIT, nn=N_INIT, r=R_INIT, fd, i, offset=0;
   int ips, w1x=0, w1y=252, w2x=10080, w2y=5292, p1x, p1y, p2x, p2y;
   float ps=30.4, psf=0.38, wtmp[4];
   float *x, *y, dx, lim[4], *fftdat, fac=0.0;

      /* test to see if data file name argument was given */

   if ( strlen(argv[1]) == 0 )
   {
      printf ("\n\nERROR!\nThe name of the source data file must be provided ");
      printf ("as an argument \nwith the command %s.\n", argv[0]);
      exit (1);
   }

   lim[0] = 0.0;   lim[1] = 0.0;   lim[2] = 0.0;   lim[3] = 0.0;

       /* consider optional command parameters */

   switch (argc)
   {
      case 13:
         sscanf (argv[9], "%f", &wtmp[0]);
         sscanf (argv[10], "%f", &wtmp[1]);
         sscanf (argv[11], "%f", &wtmp[2]);
         sscanf (argv[12], "%f", &wtmp[3]);
         w1x = (int) (1008.0 * (wtmp[0]-0.5));
         w1y = (int) (1008.0 * (wtmp[1]-0.25));
         w2x = (int) (1008.0 * (wtmp[2]-0.5));
         w2y = (int) (1008.0 * (wtmp[3]-0.25));

      case 9:
         sscanf (argv[8], "%d", &ips);
         ps = psf * (float) ips;

      case 8:
         sscanf (argv[4], "%f", &lim[0]);
         sscanf (argv[5], "%f", &lim[1]);
         sscanf (argv[6], "%f", &lim[2]);
         sscanf (argv[7], "%f", &lim[3]);

      case 4:
         sscanf (argv[2], "%d", &offset);
         sscanf (argv[3], "%d", &nn);
         if ( nn+offset > N_INIT )
         {
            printf ("Data vector length exceeded with argv[2] and argv[3]!\n");
            printf ("Default values of argv[2]=0 and argv[3]=N_INIT were ");
            printf ("used.\n");
            offset = 0 ;    nn = N_INIT ;
         }
         break;
   }
```

```
   /* open the data file and plotter device driver */

if ((gw_fp = fopen("/dev/plotter", "r+")) == 0)
{
   printf ("\n\nFailure to open plotter device driver in 'fftplt'.");
   exit (1);
}

if ( (dvfp = fopen(argv[1], "r")) == 0)
{
   printf ("\n\nFailure to open source data file in 'fftplt'.");
   exit (1);
}

if ( (pfp = fopen("/parameters", "r")) == 0)
{
   printf ("\n\nFailure to open /parameters file in 'ffthplt'.");
   exit (1);
}

   /* locate and scale the plotting region */

fprintf (gw_fp, "IN; VS %f; IW %d,%d,%d,%d; SP 1;", ps,w1x,w1y,w2x,w2y);
p1x = w1x + (int) (0.13671875 * (w2x-w1x));
p1y = w1y + (int) (0.11764706 * (w2y-w1y));
p2x = w1x + (int) (0.95703125 * (w2x-w1x));
p2y = w1y + (int) (0.94117647 * (w2y-w1y));
fprintf (gw_fp, "IP %d,%d,%d,%d;", p1x,p1y,p2x,p2y);
fprintf (gw_fp, "SC 137,957,118,941;");

   /* allocate memory for data vectors */

x = (float *) calloc (nn, sizeof(float));
y = (float *) calloc (nn, sizeof(float));
fftdat = (float *) calloc (2*nn, sizeof(float));
ybuf = (short *) calloc (n, 2);
param = (short *) calloc (32, 2);
if ( (x == 0)  ||  (y == 0) || (fftdat == 0) || (ybuf == 0) || (param == 0) )
{
   printf ("Memory allocation failure in 'ffthplt'.");
   exit (1);
}

   /* test source data file name and perform FFT if required */

if ( ( strcmp ("/fft", argv[1]) ) == 0 )
{
   if ( (fftfp = fopen ("/fft", "r")) == 0 )
   {
      printf ("\n\nFailure to open file '/fft' in 'fftplt'.");
      exit (1);
   }
   if ( (fread (fftdat, 4, 2*nn, fftfp)) != 2*nn )
   {
      printf ("Incomplete read from /fft in 'fftplt'.");
      exit (1);
   }
   fscanf (fftfp, "%[^\n]%s", str1, str2);
}

else
{
   fread (ybuf, 2, n, dvfp);
   fscanf (dvfp, "%s%s", str1, str2);
   fread (param, 2, 32, pfp);
   sprintf (str3, " FFT on pts %d-%d", offset, (offset+nn-1) );
   strcat (str1, str3);
```

```
      /* determine ADC scale factor */
   if ( (strcmp("/emg", argv[1])) == 0 )
      fac = ((float) *(param + 13)) * 0.001;
   if ( (strcmp("/force", argv[1])) == 0 )
      fac = ((float) *(param + 14)) * 0.001;
   if ( (strcmp("/mmg", argv[1])) == 0 )
      fac = 0.006134;
   if ( (strcmp("/mmgmod", argv[1])) == 0 )
      fac = 0.006134;

   for (i=0; i<nn; i++)
   {
      *(fftdat + 2*i) = fac * (float) *(ybuf + offset + i);
      *(fftdat + 2*i + 1) = 0.0;
   }

   cfft (fftdat, nn, -1);

   if ( (fftfp = fopen ("/fft", "w+")) == 0 )
   {
      printf ("\n\nFailure to open file '/fft' in 'fftplt'.");
      exit (1);
   }
   fwrite (fftdat, 4, 2*nn, fftfp);
   fprintf (fftfp, "%s\n%s\n", str1, str2);
}

   /* set up graph labels */

strcpy (ttl, str1);
strcpy (xlbl, "          FREQUENCY  (Hz)      ");
strcpy (ylbl, "    ");
strcat (ylbl, str2);
strcat (ylbl, " POWER  (squared coef.)");

   /* prepare for plotting */

dx = ((float) r) / ((float) nn);
for (i=0; i<(nn/2); i++)
{
   *(x+i) = dx * (float) i;
   *(y+i) = 4.0 * ( *(fftdat + 2*i) * *(fftdat + 2*i) +
                    *(fftdat + 2*i + 1) * *(fftdat + 2*i + 1) );
}

    /* plot the data! */

plot (gw_fp,(nn/2),lim,x,y,ttl,xlbl,ylbl);

    /* tidy up memory and files */

cfree (ybuf);
cfree (x);
cfree (y);
cfree (fftdat);

fclose (gw_fp);
fclose (dvfp);
fclose (fftfp);
fclose (pfp);

exit (0);
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       mmgmdfy.c       -> MMG Modify
*
*                       -> this programme reads the MMG RAM data file, subtracts
*                               the d.c. offset of the first 1024 points from
*                               the muscle response and surrounds the muscle
*                               response with zeros.  The beginning and end of
*                               the muscle response is determined interactively.
*
*
*       -> compile to generate executable code using:
*               cc mmgmdfy.c -o mmgmdfy         or    /cpp/bin/fcc mmgmdfy.c
*
*****************************************************************************/

#define N_INIT  2048    /* number of elements in the data vector */

#include <stdio.h>

main ()
{
   FILE *mfp, *mmfp, *fopen();
   char *calloc(), str1[50], str2[20], c='n';
   short *mmg;
   int i, j, dco=0, ib, ie;

      /* open the source data file */

   if ( (mfp = fopen("/mmg", "r")) == 0 )
   {
      printf ("\n\nFailure to open '/mmg' RAM file in 'mmgmdfy'.");
      exit (1);
   }

      /* allocate memory for integer buffer */

   mmg = (short *) calloc (N_INIT, 2);
   if (mmg == 0)
   {
      printf ("Memory allocation failure in 'mmgmdfy'.");
      exit (1);
   }

      /* read data from the RAM file */

   fread (mmg, 2, N_INIT, mfp);
   fscanf (mfp, "%[^\n]\n%s", str1, str2);

      /* calculate d.c. offset of first 1024 points */

   for (i=0; i<1024; i++)
      dco += *(mmg + i);
   dco /= 1024;

      /* print the data */

   printf ("\t\t\t\t%s  %s\n", str1,str2);
   printf ("  i        MMG --->\n\n");
   for (i=992; i<(992+128); i+=8)
   {
      printf ("%4d    ", i);
      for (j=0; j<8; j++)
         printf ("%6d  ", *(mmg+i+j));
      printf ("\n");
   }

   while ( (c != 'y') && (c != 'Y') )  /* execute loop while response is 'no' */
```

```
{
      /* read location of muscle response from keyboard */

   printf ("\nThe pre-stimulus d.c. offset is %d.    ->  ", dco);
   printf ("Enter the 'i' values of the\n   first and last MMG values to");
   printf (" be considered part of the muscle response: \nint int\n");
   scanf ("%d %d", &ib,&ie);

      /* print the data */

   printf ("\t\t\t\t%s  %s\n", str1,str2);
   printf ("  i        MMG --->\n\n");
   for (i=992; i<(992+128); i+=8)
   {
      printf ("%4d    ", i);
      for (j=0; j<8; j++)
      {
         if (  ((i+j) >= ib)  &&  ((i+j) <= ie)  )
            printf ("%6d  ", *(mmg+i+j));
         else
            printf ("%6d  ", 0);
      }
      printf ("\n");
   }

      /* request approval of the modification to the MMG data vector */

   printf ("\nIs the modification to the MMG data vector as desired?\n");
   printf ("(answer with the single character 'y' or 'n')\n");
   scanf ("%1s", &c);
}

      /* subtract offset from muscle response and set rest of vector to zero */

for (i=0; i<ib; i++)
   *(mmg + i) = 0;
for (i=ib; i<=ie; i++)
   *(mmg + i) -= dco;
for (i=(ie+1); i<2048; i++)
   *(mmg + i) = 0;

      /* open the destination RAM data file */

if ( (mmfp = fopen("/mmgmod", "w")) == 0 )
{
   printf ("\n\nFailure to create '/mmgmod' RAM file in 'mmgmdfy'.");
   exit (1);
}

      /* write modified mmg data vector to RAM file */

fwrite (mmg, 2, N_INIT, mmfp);
fprintf (mmfp, "%smd\nMMG\n", str1);

      /* tidy up memory and files */

cfree (mmg);

fclose (mfp);
fclose (mmfp);

exit (0);
}
/*****************************************************************************/
```

```
/*****************************************************************************
*
*       plot.c
*               -> plots floating-point data vector 'y' as a function of
*                       floating-point data vector 'x' on a set of axes.
*
*       -> compile to generate object file using:
*               cc -c plot.c    or      /cpp/bin/fcc -c plot.c
*
*       -> to link with other object files, use
*         cc other_files.o plot.o -lm  or  /cpp/bin/fcc other_files.o plot.o -lm
*
*       variables :
*
*               gw_fp ->  graphics window file pointer obtained from a call to
*                               fopen in the calling process.
*
*               n -> number of data points in 'x' and 'y' to be plotted.
*
*               l ->  contains extreme values of 'x' and 'y' data vectors for
*                       the determination of plotting limits.
*                       l[0] = xmin
*                       l[1] = xmax
*                       l[2] = ymin
*                       l[3] = ymax
*                       -> if xmax <= xmin then this subroutine will calculate
*                               new values of xmax and xmin.
*                       -> if ymax <= ymin then this subroutine will calculate
*                               new values of ymax and ymin.
*
*               px -> pointer to data vector 'x' of type 'float'.
*                       (horizontal coordinates of data points)
*
*               py -> pointer to data vector 'y' of type 'float'.
*                       (vertical coordinates of data points)
*
*               ttl -> 32 character title for the graph.
*
*               xlbl -> 32 character label for the x-axis.
*
*               ylbl -> 32 character label for the y-axis.
*
*****************************************************************************/

#include <stdio.h>
#include <math.h>

plot(gw_fp,n,l,px,py,ttl,xlbl,ylbl)
FILE *gw_fp;
int n;
float l[4], *px, *py;
char *ttl, *xlbl, *ylbl;
{
   char *calloc();
   int intpow10(), rint();
   double pow();
   float min(), max(), xfac, yfac, xmean, ymean, xlrgfac, ylrgfac;
   int i, *pltx, *plty, xpow, ypow, xmin, xmax, ymin, ymax, xtic, ytic, xi, yi;

        /* allocate memory for data vectors */
   pltx = (int *) calloc(n, sizeof(int));
   plty = (int *) calloc(n, sizeof(int));
   if ( pltx == 0  ||  plty == 0 )
   {
      printf("\n\nMemory allocation failure in Plot (pltx or plty).\n");
      return;
   }
```

```
    /* find the extreme data values */
if (l[0] >= l[1])
{
   l[0] = min(px,n);
   l[1] = max(px,n);
}
if (l[2] >= l[3])
{
   l[2] = min(py,n);
   l[3] = max(py,n);
}

   /* determine the scaling factors */
xpow = intpow10(l[1]-l[0]);
xfac = (float) pow(10.0, (double)(3-xpow));
ypow = intpow10(l[3]-l[2]);
yfac = (float) pow(10.0, (double)(3-ypow));

   /* determine number spacing on the axes */
if ((l[1]-l[0])*xfac < 5000)
{
   xtic = 500;
   if ((l[1]-l[0])*xfac < 2000)
      xtic = 200;
}
else
   xtic = 1000;

if ((l[3]-l[2])*yfac < 5000)
{
   ytic = 500;
   if ((l[3]-l[2])*yfac < 2000)
      ytic = 200;
}
else
   ytic = 1000;

   /* translate and scale the extrema and data vectors for plotting */
xmean = (((float)xtic)/xfac) * (int)(xfac*(l[0]+l[1])/(2.0*((float)xtic)));
ymean = (((float)ytic)/yfac) * (int)(yfac*(l[2]+l[3])/(2.0*((float)ytic)));
xmin = xtic * (int) (-0.9995 + (l[0] - xmean) * xfac / ((float) xtic));
xmax = xtic * (int) ( 0.9995 + (l[1] - xmean) * xfac / ((float) xtic));
ymin = ytic * (int) (-0.9995 + (l[2] - ymean) * yfac / ((float) ytic));
ymax = ytic * (int) ( 0.9995 + (l[3] - ymean) * yfac / ((float) ytic));
for (i=0; i<n; i++)
{
   *(pltx + i) = (int) (( *(px + i) - xmean) * xfac);
   *(plty + i) = (int) (( *(py + i) - ymean) * yfac);
}

    /* print labels and scale power factors */
fprintf (gw_fp, "sr1.2,3.6;");  /* set the character size */
fprintf (gw_fp, "pu352,961; lb%-32s\3;", ttl);    /* print title */
xpow = (l[1]*l[1] > l[0]*l[0]) ? intpow10(l[1]): intpow10(l[0]);
ypow = (l[3]*l[3] > l[2]*l[2]) ? intpow10(l[3]): intpow10(l[2]);
fprintf (gw_fp, "pu320,8; lb%-32s\3;", xlbl);     /* print x-axis label */
if ( (xpow < 0) || (xpow > 3) )
   fprintf(gw_fp, "cp2,0; lbx10\3; cp0,0.4; sr1.0,3.0; lb%-3d\3;", xpow);
fprintf (gw_fp, "di0,1; sr1.2,3.6;");
fprintf (gw_fp, "pu25,50; lb%-32s\3;", ylbl);    /* print y-axis label */
if ( (ypow < 0) || (ypow > 3) )
   fprintf(gw_fp,"pu65,810; lbx10\3; cp0,0.4; sr1.0,3.0; lb%-3d\3;",ypow);

    /* scale the plotting region and draw the axes */
fprintf (gw_fp, "di1,0; sr1.0,3.0; sc%d,%d;%d,%d;", xmin,xmax,ymin,ymax);
fprintf (gw_fp,"pu%d,%d; pd%d,%d; pd%d,%d;", xmin,ymax,xmin,ymin,xmax,ymin);

   /* mark and number the axes */
```

```
    xlrgfac = (float) pow(10.0, (double)(-xpow));
    fprintf (gw_fp, "tl0.94,0.94;");
    switch (xpow)
    {
       case 1:
          for (xi=xmin; xi<=xmax; xi+=xtic)
             fprintf (gw_fp, "pu%d,%d; xt; cp-3.2,-1.0; lb%5.1f\3;", xi, ymin,
                                    (xmean + ((float) xi)/xfac)*xlrgfac*10.0);
          break;
       case 2:
          for (xi=xmin; xi<=xmax; xi+=xtic)
             fprintf (gw_fp, "pu%d,%d; xt; cp-1.2,-1.0; lb%2d\3;", xi, ymin,
                          (rint ((xmean + ((float) xi)/xfac)*xlrgfac*100.0)));
          break;
       case 3:
          for (xi=xmin; xi<=xmax; xi+=xtic)
             fprintf (gw_fp, "pu%d,%d; xt; cp-1.2,-1.0; lb%2d\3;", xi, ymin,
                          (rint ((xmean + ((float) xi)/xfac)*xlrgfac*1000.0)));
          break;
       default:
          for (xi=xmin; xi<=xmax; xi+=xtic)
             fprintf (gw_fp, "pu%d,%d; xt; cp-2.2,-1.0; lb%5.2f\3;", xi, ymin,
                                       (xmean + ((float) xi)/xfac)*xlrgfac);
          break;
    }
    ylrgfac = (float) pow(10.0, (double)(-ypow));
    fprintf (gw_fp, "tl0.47,0.47;");
    switch (ypow)
    {
       case 1:
          for (yi=ymin; yi<=ymax; yi+=ytic)
             fprintf (gw_fp, "pu%d,%d; yt; cp-6.0,-0.3; lb%5.1f\3;", xmin, yi,
                                    (ymean + ((float) yi)/yfac)*ylrgfac*10.0);
          break;
       case 2:
          for (yi=ymin; yi<=ymax; yi+=ytic)
             fprintf (gw_fp, "pu%d,%d; yt; cp-6.0,-0.3; lb%5d\3;", xmin, yi,
                          (rint ((ymean + ((float) yi)/yfac)*ylrgfac*100.0)));
          break;
       case 3:
          for (yi=ymin; yi<=ymax; yi+=ytic)
             fprintf (gw_fp, "pu%d,%d; yt; cp-6.0,-0.3; lb%5d\3;", xmin, yi,
                          (rint ((ymean + ((float) yi)/yfac)*ylrgfac*1000.0)));
          break;
       default:
          for (yi=ymin; yi<=ymax; yi+=ytic)
             fprintf (gw_fp, "pu%d,%d; yt; cp-6.0,-0.3; lb%5.2f\3;", xmin, yi,
                                       (ymean + ((float) yi)/yfac)*ylrgfac);
          break;
    }

        /* plot the data */
    fprintf(gw_fp, "pu%d,%d; pd;", *pltx,*plty); /* move pen to first point */
    for (i=1; i<n; i++)
    {
       fprintf(gw_fp, "pa%d,%d;", *(pltx+i),*(plty+i));  /* plot data point */
    }
    fprintf(gw_fp, "pu%d,%d; sp;", xmin,ymin);

    fflush (gw_fp);
    cfree (pltx);
    cfree (plty);

    return;
}
/*****************************************************************************/

float min(pt,n)
```

```
int n;
float *pt;
{
   int i;
   float min;
   min = *pt;
   for (i=1; i<n; i++)
   {
      if (*(pt+i) < min)
         min = *(pt+i);
   }
   return(min);
}
/*****************************************************************************/

float max(pt,n)
int n;
float *pt;
{
   int i;
   float max;
   max = *pt;
   for (i=1; i<n; i++)
   {
      if (*(pt+i) > max)
         max = *(pt+i);
   }
   return(max);
}
/*****************************************************************************/

intpow10(z)
float z;
{
   int xpnt;
   double log10();

   if (z == 0.0 )
      return (0);

   if (z < 0.0 )
      z = -z;

   if (z < 1.0 )
      xpnt = (int) (log10((double) z) - 1.0);
   else
      xpnt = (int) log10((double) z);

   if ( (((float) log10((double) z)) - (float) xpnt) > 0.9999 )
      xpnt++;

   return(xpnt);
}
/*****************************************************************************/

int rint(x)     /* return a Rounded INTeger (not truncated) */
float x;
{
   return ( (int) ( (x < 0.0) ? (x - 0.5) : (x + 0.5) ) );
}
/*****************************************************************************/
```

```
#
# posexan -> prepare data for the Position Experiment Analysis.
#
# Implement by typing 'sh posexan' in the PAM window command line.
#
cp /usr/prog.d/analysis/btascii /
cp /usr/prog.d/analysis/cgw /
cp /usr/prog.d/analysis/efdplt /
cp /usr/prog.d/analysis/efdtr /
cp /usr/prog.d/analysis/fftplt /
cp /usr/bin/print_screen /
#
cgw
cd /usr/data/Mod_posx
#
for i in F23LPC??A1
do
   /efdtr $i
   echo \\014 > /dev/lp # send form-feed to printer.
   /efdplt /mmg 992 128
   /print_screen unframed window plot_window
   /btascii /mmg 992 128 > /dev/lp
   echo \\012\\012 > /dev/lp    # send 2 line-feeds to the printer.
   /efdplt /force 1024 640
   /print_screen unframed window plot_window
   echo \\014 > /dev/lp # send form-feed to printer.
   /fftplt /mmg 0 512 0.0 1000.0 0.0 5000.0
   /print_screen unframed window plot_window
   /btascii /fft 0 32 > /dev/lp
   echo \\012\\012 > /dev/lp    # send 2 line-feeds to the printer.
   /fftplt /mmg 1040 512 0.0 1000.0 0.0 5000.0
   /print_screen unframed window plot_window
   /btascii /fft 0 32 > /dev/lp
done
```

```
#
# reptest -> prepare data for the Reproducibility Test
#
# Implement by typing 'sh reptest' in the PAM window command line.
#
cp /usr/prog.d/analysis/btascii /
cp /usr/prog.d/analysis/cgw /
cp /usr/prog.d/analysis/efdplt /
cp /usr/prog.d/analysis/efdtr /
cp /usr/prog.d/analysis/fftplt /
cp /usr/bin/print_screen /
#
cgw
cd /usr/data/Mod_stimx
#
for i in F24RSC01* F24RSC03* F24RSC05* F24RSC07*
do
   /efdtr $i
   /efdplt /mmg 992 128
   echo \\014 > /dev/lp # send form-feed to printer.
   /print_screen unframed window plot_window
   /btascii /mmg 992 128 > /dev/lp
   echo \\012\\012 > /dev/lp    # send 2 line-feeds to the printer.
   /efdplt /force 1024 640
   /print_screen unframed window plot_window
   /fftplt /mmg 0 512 0.0 1000.0 0.0 5000.0
   echo \\014 > /dev/lp # send form-feed to printer.
   /print_screen unframed window plot_window
   /btascii /fft 0 32 > /dev/lp
   echo \\012\\012 > /dev/lp    # send 2 line-feeds to the printer.
   /fftplt /mmg 1040 512 0.0 1000.0 0.0 5000.0
   /print_screen unframed window plot_window
   /btascii /fft 0 32 > /dev/lp
done
```

REFERENCES

ANSI/IEEE Std 488-1978, IEEE Standard Digital Interface for Programmable Instrumentation, IEEE Inc., New York, USA.

Barry, W. H., Harrison, D. C., Fairbank, W. M., Lehrman, K., Malmivuo, J. A. V., Wikswo, J. P., Jr. (1977), "Measurement of the human heart vector", Science, vol. 198, pp. 1159-1162.

Baule, G., McFee, R. (1963), "Detection of the magnetic field of the heart", Am. Heart J., (July 1963), pp. 95-96.

Carlson, F.D. & Wilkie, D.R. (1974), Muscle Physiology, Prentice Hall, Englewood Cliffs, New Jersey.

Cohen, D. (1968), Science, vol. 161, p. 784.

Cohen, D. & Givler, E. (1972), "Magnetomyography: magnetic fields around the human body produced by skeletal muscles". Applied Physics Letters, vol. 21, no. 3, pp. 114-116.

Cohen, D. (1975), "Magnetic fields of the human body", Physics Today, (August 1975), pp. 34-44.

Cohen, D., et al. (1980), Proc. Natl. Acad. Sci. USA, vol. 77, p. 1447.

de la Lande, I.S., Tyler, M.J. & Pridmore, B.R. (1962), "Pharmacology of Tiliqua [Trachysaurus] rugosa (the sleepy lizard)", Australian Journal of Experimental Biology, vol. 40, pp. 129-138.

Duret, C., Darp, P. (1983), "Instrumentation for biomagnetism", Il Nuovo Cimento, vol. 2D, no. 2, pp. 123-141.

Engel, W. K., Irwin, R.L. (1967), "A histochemical-physiological correlation of frog skeletal muscle fibres", American Journal of Physiology, v. 213, pp. 511-518.

Fulton, J.F. (1925), "A correlation of the size of the action current of skeletal muscle with length, tension and initial heat production". Journal of Physiology, vol. 60, p. xxi.

Giallorenzi, T. G., Bucaro, J. A., Dandridge, A., Cole, J. H. (1986), "Optical-fiber sensors challenge the competition", IEEE Spectrum, v. 23, n. 9, pp. 44-49.

Gielen, F.L.H. & Wikswo, J.P., Jr. (1985), "Electric and magnetic motor unit action signals in the rat". Proceedings of the 38th Annual Conference on Engineering in Medicine and Biology, Chicago, p. 340.

Gielen, F.L.H., Roth, B.J. & Wikswo, J.P., Jr. (1986). "Capabilities of a toroid-amplifier system for magnetic measurement of current in biological tissue". IEEE Transactions on Biomedical Engineering, vol. BME-33, pp. 910-921.

Gonzalez-Serratos, H. (1971), "Inward spread of activation in vertebrate muscle fibres", Journal of Physiology, vol. 221, pp. 777-799.

Grieve, D.W. (1958), "The relation between twitch tension and action potential in fatigued and iodoacetate-poisoned frog sartorius muscle". Journal of Physiology, vol. 143, p. 36P.

Guyton, A.C. (1981), Textbook of Medical Physiology, Sixth Edition, W. B. Saunders, Philidelphia, USA, p. 129.

Halliday, D., Resnick, R. (1962), Physics, John Wiley & Sons, Inc., New York, USA.

Hodgkin, A. L., Huxley, A. F. (1952), "A quantitative description of membrane current and its application to conduction and excitation in nerve", Journal of Physiology, vol. 117, pp. 500-544.

Jackson, J. D. (1975), Classical Electrodynamics, Second Edition, John Wiley & Sons Inc., New York, USA.

Koga, S. & Nakamura, A. (1983), "Magnetic signals from human skeletal muscles". Il Nuovo Cimento, vol. 2D, no. 2, pp. 642-649.

Lekkala, J., Malmivuo, J. (1981), "Simultaneous measurement of the magnetic heart vector components with the unipositional lead system", Biomagnetism: Proceedings of the Third International Workshop, Walter de Gruyter, West Berlin, pp. 319-326.

Lemonick, M. D. (1987), "Superconductors". Time, (May 11, 1987), pp. 56-63.

MacHattie, L. E. (1972), "A highly stable current or voltage source". Journal of Physics E: Scientific Instruments, v. 5, p. 1016.

McDonald, D.G., Boutilier, R.G. & Toews, D.P. (1980), "The effects of enforced activity on ventilation, circulation and blood acid-base balance in the semi-terrestrial anuran, Bufo marinus". Journal of Experimental Biology, vol. 84, pp. 273-287.

Plonsey, R. (1981), "Generation of magnetic fields by the hyman body (theory)", Biomagnetism: Proceedings of the Third International Workshop, Walter de Gruyter, West Berlin, pp. 177-205.

Press, W. H., Flannery, B. P., Teukolsky, S. A., Fetterling, W. T. (1986), Numerical Recipes: The Art of Scientific Computing, Cambridge University Press, New York, USA, p. 394.

Reite, M., Zimmerman, J.E., Edrich, J. & Zimmerman, J. (1976), "The human magnetoencephalogram: some EEG and related correlations". Electroencephalography and Clinical Neurophysiology, vol. 40, pp. 59-66.

Smith, R. F., Ovalle, W. K., Jr., (1973), "Varieties of fast and slow extrafusal muscle fibres in amphibian hind limb muscles", Journal of Anatomy, v. 116, n. 1, pp. 1-24.

Stevens, J.C., Dickinson, V. & Jones, N.B. (1980), "Mechanical properties of human skeletal muscle from in vitro studies of biopsies", Medical & Biological Engineering & Computing, vol. 18, pp. 1-9.

Swinney, K. R., Wikswo, J. P., Jr., 1980, "A calculation of the magnetic field of a nerve action potential", Biophys. J., vol. 32, pp. 719-732.

Taccardi, B. (1983), "Electrophysiology of excitable cells and tissues, with special consideration of the heart muscle". Biomagnetism: An Interdisciplinary Approach, Plenum Press, New York, USA, pp. 141-172.

Tripp, J.H. (1981), "Biomagnetic fields and cellular current flow", Biomagnetism: Proceedings of the Third International Workshop, Walter de Gruyter, West Berlin, p. 212.

Tripp, J.H. (1983), "Physical concepts and mathematical models", Biomagnetism: An Interdisciplinary Approach, Plenum Press, New York, USA, pp. 101-139.

Vander, A.J., Sherman, J.H. & Luciano, D.S. (1980), Human Physiology: The Mechanisms of Body Function, Third Edition, McGraw-Hill, New York, USA, p. 220.

Watts, C.F. (1924), "The relation between the size of the electrical response and the tension developed in the contraction of striated muscle", Journal of Physiology, vol. 59, p. xv.

Wikswo, J.P., Jr. (1983), "Cellular action currents", Biomagnetism: An Interdisciplinary Approach, Plenum Press, New York, USA, pp. 173-207.

Williamson, S.J. & Kaufman, L. (1981), "Biomagnetism", Journal of Magnetism and Magnetic Materials, vol. 22, no. 2, pp. 129-202.

Williamson, S.J., Romani, G-L, Kaufman, L. & Modena, I. (1983), editors of: Biomagnetism: An Interdisciplinary Approach, Plenum Press, New York, USA.

Wright, A.H. & Wright, A.A. (1949), Handbook of Frogs and Toads of the United States and Canada, Comstock Publishing, Ithica, New York, USA.

Zimmerman, J. E. (1983), "Magnetic quantities, units, materials and measurements", Biomagnetism: An Interdisciplinary Approach, Plenum Press, New York, USA, pp. 17-42.