Design of Instrumentation for Metabolic Monitoring of the Adélie Penguin

A thesis presented in partial fulfillment of
the requirements for the degree of
Master of Science
in Physics at
Massey University

by

Paul Stephen Ryland

December 2000

Abstract

The motivating question for the work described in this thesis was "How does the Adélie penguin cope with cold?" It was reasoned that the time-scale of temperature changes in Antarctica precluded all but metabolic and physiological responses. To determine these, a system capable of measuring and recording these biological variables in the penguins natural environment, was designed.

A device, based on the principles of near infrared spectroscopy, was developed that could measure the relative oxygen saturation of haemoglobin and the reduction state of cytochrome oxidase as well as heart rate and blood volume. The completed device was housed in a black, waterproof, plastic container, measuring 65mm x 92mm x 25mm and weighing 132.7g.

Co-ordination of measurements was achieved with operating system-like control software implemented in Motorola HC11 assembly code. Synchronous detection was used for signal acquisition and a pulse algorithm, implemented in assembly code, allowed real time pulse measurement from the input signals. Programs were written in Matlab and C++ to investigate the characteristics and limits of these techniques.

Preliminary testing of the device on human subjects successfully showed changes in metabolic state as a result of physical activity. The results of field testing on Adélie penguins were unable to answer the original question due to a number of physical factors. However, the success of human trials suggests that, with modification and improvement, the device has potential as a valuable research instrument, applicable to a variety of other species.

Acknowledgements

I would like to express my sincere gratitude to the following people whose contribution and assistance were invaluable to me and to the completion of this thesis. I have gained an enormous amount from their skills and expertise and know that the knowledge and experience gained will continue to be valuable in the future. Particular thanks go to my supervisor, Dr. Simon Brown, for his considerable contribution and guidance throughout the many aspects of the project. In areas ranging from remedial biology, lab work, field work, presentations, CV writing, grant applications and thesis proofing, his enthusiasm for the project and encouragement have been exceptional. I also wish to thank my co-supervisor, Associate Professor Robert O'Driscoll for the introduction to this project and whose extensive electronics expertise guided me in solving some of my most taxing hardware and software problems.

I gratefully thank the electronics guru, Robin Dykstra, for directing my circuit design efforts, providing development equipment and manuals, component advise, his stringent, NASA approved circuit-board-layout quality control and good friendship. Thanks also to Peter Lewis who helped select and order numerous electronic components, made an exceptional contribution during the time critical construction phase allowing deadlines to be met and even provided reading material for the quiet times in Antarctica. Also from the electronics workshop, I wish to thank Udo von Mulert for allowing the extensive use of the workshop facilities after hours and on weekends, and Keith Whitehead for his advise and ideas on numerous aspects of the project.

For her help in Antarctica, expert penguin handling skills and easy going personality I would like to thank Yvette Cottam. I also wish to express my gratitude to Antarctica New Zealand for their approval of the work in Antarctica with the Adélie penguin and to Massey University for the opportunity to work on this project. I thank Massey University for the approval of my Institute of Fundamental Sciences Graduate Research Fund application amounting in $1731 towards purchase of equipment and related expenses.

Particular thanks to John Pedley for his help with the use of the ECG to test the second prototype and for the loan of the exer-cycle used to test the third prototype. I would also like to recognize the many test subjects who exerted themselves in the name of science, in particular, to Jane Shierlaw whose high quality data made it to print. I would also like to thank Jane for her veterinary advice as well as thesis writing comments and ideas. Finally, I would especially like to thank Mark Hunter for the very many coffees, countless ridiculously late nights and for being someone available to discuss the more subtle points of thesis writing (some of which were relevant).

Contents

Abstract ii
Acknowledgements iii
Contents iv
List of Figures vii

Chapter 1 Introduction 1
1.1 The Problem 1
1.2 Measurement 1
1.3 Background – The Adélie Environment 3
1.4 Technology 5
1.5 Measurement Principles 5
1.5.1 Near Infrared Spectroscopy 5
1.5.2 Triple Wavelength Oxygen Saturation Measurement 8
1.5.3 Near Infrared Spectrometry 9
1.6 Thesis Overview 10

Chapter 2 Instrumentation 12
2.1 Design Specifications 12
2.2 Design Overview 13
2.3 Analogue Circuitry 15
2.3.1 The Sensor Head 15
2.3.2 The Synchronous Detector 16
2.3.3 Synchronous Detection 17
2.3.4 The Analogue Stage 21
2.4 Digital Circuitry 22
2.4.1 Sampling and Digitisation 23
2.4.2 The Microcontroller 24
2.4.2.1 Output Compare 24
2.4.2.2 Interrupts 25
2.4.2.3 Real Time Clock 25
2.4.2.4 Additional Features of the Microcontroller 25
2.4.3 Memory 26
2.4.4 Communication 26
2.4.5 Power Considerations 27
2.4.5.1 Power Supply Sources 27
2.4.5.2 Power Considerations on the Device 28
2.4.5.3 Power Saving 28
2.5 Construction 29

Chapter 3 Control Software and Algorithms 31
3.1 The Logical Model of the Device 32
3.1.1 Setting Write Through Mode and the Result Memory Pointer 34
3.1.2 Setting the System Clock 34
3.1.3 Communication Rate 34
3.1.4 Help Command 34
3.1.5 Debug Mode 34

3.2 The Microcontroller Operating System 36
3.2.1 Control Modules 36
3.2.1.1 Operating System 36
3.2.1.2 System Initialisation 36
3.2.1.3 Command 36
3.2.1.4 Interrupts 38
3.2.1.5 Sequence Execution 38
3.2.2 Hardware Modules 39
3.2.2.1 Serial Communication 39
3.2.2.2 Memory 39
3.2.2.3 Real Time Clock 40
3.2.2.4 Sampling 41
3.2.2.5 Utilities 41
3.2.3 Measurement Sequence Instructions 43
3.3 Measurement Scripting Language 45
3.4 Pulse Rate Calculation Algorithm 48
3.4.1 The Algorithm 49
3.4.2 Analysis of the Pulse Measurement Algorithm 51

Chapter 4 Prototyping and Application 57
4.1 Developmental Testing 57
4.1.1 Early Prototypes 57
4.1.2 Signal Verification using an Electrocardiogram (ECG) 58
4.1.3 Software Development System 59
4.1.4 Blood Oxygen Saturation 60
4.1.5 Pulse Rate Measurement 61
4.1.6 The Stand-alone Prototype 62
4.1.7 Testing of the Final Device 62
4.2 Field Testing 67
4.2.1 Capture Technique and Attachment 68
4.2.2 Physical Results and Observations 69
4.2.3 Biological Responses 70

Chapter 5 Conclusion 74
5.1 Evaluation 74
5.2 Future Development 76

Appendix A Derivations 77
A.1 Oxygen Saturation Derived from Double Wavelength Measurements 77
A.2 Oxygen Saturation Derived from Triple Wavelength Measurements 79
A.3 Relative Blood Volume Derived from Double Wavelength Measurements 80

Appendix	**B**	**MatLab Programs**	**82**
	B.1	Synchronous Detector Numerical Solution	82
	B.2	Pulse Algorithm Simulation	83
	B.3	Input-Signal Drift Limitations for the Pulse Algorithm	86
	B.4	Period Measurement Limitation due to the Digital Filter	89
	B.5	Input-Signal Noise Limitations for the Pulse Algorithm	89
	B.6	Utility Routines	91
Appendix	**C**	**Circuit Diagrams**	**96**
	C.1	Overview	96
	C.2	Sensor Head	97
	C.3	Analogue Stage	98
	C.4	Digital Stage	99
	C.5	Memory	100
	C.6	Power Supply	101
	C.7	RS-232 Interface Unit	102
Appendix	**D**	**Printed Circuit Board Layouts**	**103**
Appendix	**E**	**Assembly Code**	**106**
	E.1	Pengos.a	CD
	E.2	Init.a	CD
	E.3	Equates.a	CD
	E.4	Global.a	CD
	E.5	Vectors.a	CD
	E.6	Commands.a	CD
	E.7	Exec.a	CD
	E.8	Pulse.a	CD
	E.9	LED.a	CD
	E.10	Temp.a	CD
	E.11	Delay.a	CD
	E.12	Loop.a	CD
	E.13	Intr.a	CD
	E.14	Sample.a	CD
	E.15	Mem.a	CD
	E.16	RTC.a	CD
	E.17	Serial.a	CD
	E.18	Utils.a	CD
	E.19	EEPROM.a	CD
References			107

List of Figures

Figure	Figure caption	Page
1.1	The metabolism of sugar with the cell.	2
1.2	Ross Island and breeding colonies of the Adélie penguin.	3
1.3	An Adélie penguin upon its nest of pebbles.	4
1.4	Absorption spectra of oxy- and deoxy- haemoglobin and oxidised and reduced cytochrome oxidase.	6
1.5	Interpolating to find an estimate of the background absorbance at wavelength, λ_2.	8
1.6	The two main configurations for near infrared spectroscopic systems, transmission mode and reflectance mode.	10
2.1	Adélie penguin with a control unit and sensor head fitted.	13
2.2	The connectivity and packaging of the sensor head and control unit.	14
2.3	The final realisation of the device.	14
2.4	The sensor head layout.	15
2.5	The sensor head circuit.	16
2.6	A graphical representation of the implemented synchronous detector.	17
2.7	Demodulation of an in-phase, input sine wave.	18
2.8	The Fourier components of the full-wave rectified sine wave.	19
2.9	The phase selectivity of the detector.	19
2.10	The frequency and phase response of the synchronous detector.	21
2.11	The amplification and filtering stage of the device.	22
2.12	The digital circuit components.	22
2.13	Serial communication between digital devices using the Motorola serial peripheral interface.	23
2.14	Block diagram of the 68HC11E9 microprocessor.	25
2.15	The power-supply schematic.	27
2.16	Connectivity of a computer, the serial interface unit and the device.	28
2.17	The functional regions of the PCB layout for the control unit.	29
2.18	Photographs of the completed device.	30

3.1	System overview.	31
3.2	The logical model of the device.	32
3.3	Branch vector use in operating system debugging.	35
3.4	Hierarchical arrangement of modules within the microcontroller operating system.	37
3.5	The algorithm used to make light scattering measurements.	44
3.6	A graphical representation of the pulse rate calculation process.	49
3.7	The beginning of calculation of the maximum, minimum and median values from the waveform stored in the median buffer.	50
3.8	Example waveforms that posed a problem to pulse algorithm without the use of the upper and lower quartile in generating the square wave.	50
3.9	Transition points used to obtain period estimates.	51
3.10	The boundaries for which each input signal ceases to cross the calculated median value and therefore ceases to generate the square wave.	52
3.11	Transfer function of the digital low-pass filter used in the pulse measurement algorithm.	54
3.12	The performance of the pulse measurement algorithm as the pulse frequency increased beyond the corner frequency of the digital low-pass filter	54
3.13	The limiting signal to noise ratio for a sinusoidal input signal.	55
3.14	The limiting signal to noise ratio for an asymmetric square wave input signal of similar shape to acquired human data.	56
3.15	A pulse measurement example using real data collected during testing on a human subject.	56
4.1	The second prototype.	57
4.2	The correlation between ECG data and the signal obtained from the second prototype.	58
4.3	ECG absorbance signal comparison for a subject with increased heart rate.	59
4.4	Blood oxygen saturation measurement for a subject undergoing approximately two minutes of physical exertion on an exer-cycle.	60
4.5	Resting pulse signal.	61
4.6	Pulse signal after exercise.	61
4.7	Intensity data acquired from the device.	63
4.8	Relative blood-oxygen saturation.	64
4.9	Relative cytochrome oxidase saturation.	64

4.10	Relative blood volume.	65
4.11	A pulse waveform used to calculate pulse rate.	66
4.12	Pulse rate measurement.	66
4.13	Temperature measurement.	67
4.14	Attachment of the device using tape.	69
4.15	Data collected during two potentially stressful events for an Adélie penguin.	71
4.16	Relative blood oxygen saturation calculated from the data in figure 4.15.	72
4.17	Relative oxygenation state of cytochrome oxidase calculated from the data in figure 4.15.	72
4.18	Relative blood volume calculated from the data in figure 4.15.	73
4.19	Pulse rate data acquired during the two potentially stressful events for an Adélie penguin.	73
5.1	An alternative layout for the sensor head that may give improved signal strength.	75
A2.1	The use of three wavelengths to remove non-uniform drift in oxygen saturation measurements.	79
D.1	Printed circuit board layouts with component overlays.	103
D.2	Printed circuit board layouts.	104
D.3	Serial interface unit PCB layout.	105

Chapter 1

Introduction

1.1 The problem

For the Adélie penguin (*Pygoscelis adélie*) of Antarctica there exists an intriguing biological paradox. As for any animal living in this environment, adaptation to cold and the regulation of body temperature is of primary importance. During breeding however, the Adélie penguins exhibit behaviour that seems to defy these thermal demands. For periods lasting as long as two weeks [1] they remain on their nests vigilantly guarding their eggs. During this time they fast, exhibit minimal muscular activity and no behavioural activities such as huddling. Fasting results in a reduction of resting metabolic rate thereby conserving energy. Contrary to this, an adaptive response to cold is to increase metabolic rate, producing heat from food or body fat reserves. As the penguins are fasting, this increase in metabolic rate results in the depletion of the bird's insulating body fat layer, further increasing the need for heat generation. As thermogenesis does not occur significantly by other means there seem to be conflicting metabolic demands and the question arises, 'How does the Adélie penguin cope in this environment?'

1.2 Measurement

An increase in metabolic rate implies an increase in the demand for oxygen. Processes that exhibit a response to changes in metabolic rate are the transport of oxygen via haemoglobin and the oxidation of substrates within cells by oxidative phosphorylation. Two proteins involved in these processes, haemoglobin (Hb) and cytochrome oxidase (COX), exhibit changes in their spectral characteristics depending on their oxygenation state. These spectra can be observed *in vivo* using near-infrared (NIR) spectroscopy, a technique that has been employed successfully with the human foetus, neonate and adult [2].

Oxidised and reduced haemoglobin and cytochrome oxidase exist in equilibrium in blood and in the mitochondria of cells respectively (1.1 and 1.2). The equilibrium concentrations for each of these give information about the supply and demand for oxygen at the beginning and end of the metabolic process.

$$Hb + O_2 \rightleftharpoons HbO_2 \qquad (1.1)$$

$$COX + O_2 \rightleftharpoons COXO_2 \qquad (1.2)$$

Haemoglobin, oxygen and oxyhaemoglobin are transported throughout the body via blood vessels (figure 1.1). A concentration gradient between the blood and the cell causes oxygen to diffuse through the wall of the blood vessel into the cell. Higher concentrations of oxyhaemoglobin observed in the blood imply that the supply of oxygen is greater than the demand due to decreased respiration or reduced metabolic

rate. If oxyhaemoglobin concentrations decrease, then oxygen consumption is greater than the demand as a result of increased respiration or higher metabolic rate.

Oxygen is consumed within the cell during the last stage of oxidative phosphorylation. Within the cell, sugar is broken down into a smaller molecule called pyruvate that is oxidised within the mitochondria to produce the waste products; carbon dioxide and water. This final reaction is catalysed by cytochrome oxidase that exists in equilibrium with oxygen in the mitochondrial membrane. If high levels of oxidised cytochrome oxidase are observed, this indicates that the rate of sugar metabolism is slow and, conversely, highly reduced cytochrome oxidase indicates an increased metabolic rate.

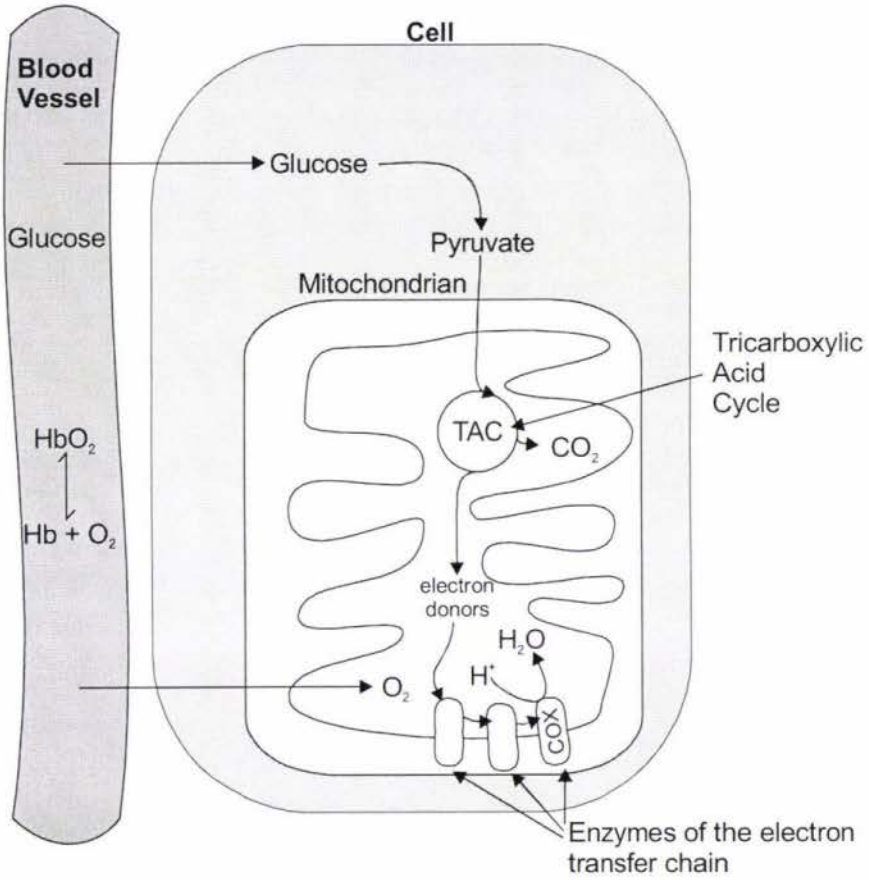


Figure 1.1: The metabolism of sugar with the cell. Oxygen exists in equilibrium with two proteins, haemoglobin and cytochrome oxidase, in blood and in the mitochondria within cells respectively. The spectral characteristics of these two proteins depend on the relative concentrations of their oxidised and reduced forms. Changes in these spectra give information about the rate of oxygen metabolism.

By developing a device capable of making oxygen saturation measurements along with pulse and temperature measurements a correlation may be observed between the environmental temperature and biological responses of the penguin. Such a device must be portable, small and lightweight so as not to inhibit the normal activities of the bird or cause stress resulting in unrealistic data. NIR spectroscopy, a non-invasive technique, is ideally suited to this problem and the development of such an instrument would allow changes in the relative oxygen saturation to be observed with minimal impact on the penguin.

1.3 Background – The Adélie Environment

Antarctica, and its surrounding oceans, form one of the most extreme, and yet habitable, environments on earth. All species that live and breed in this southern polar region face the same survival issue: the adaptation to cold and maintenance of body temperature. Each animal that lives in or visits this environment exhibits biological or behavioural adaptations that enable it to combat the extreme cold such as increased body fat, thicker skin/feather layers or group huddling behaviour. Adélie penguins spend eight months of the year living and foraging off the pack ice that forms where the polar ice cap meets the southern ocean. The birds move with the pack ice that advances and recedes seasonally, covering a distance of over 1300km [3]. Starting around mid October, the Adélie penguins make a trek, often travelling 80km or more inland, to their annual breeding sites located on the shores of the Antarctic mainland or on many of the Antarctic islands. Some nesting colonies can number in the tens of thousands and on Ross Island (figure 1.2), where there are six colonies [4], a major nesting site of approximately 60,000 Adélie penguins is located at Cape Bird (Barton, K. J., personal communication).

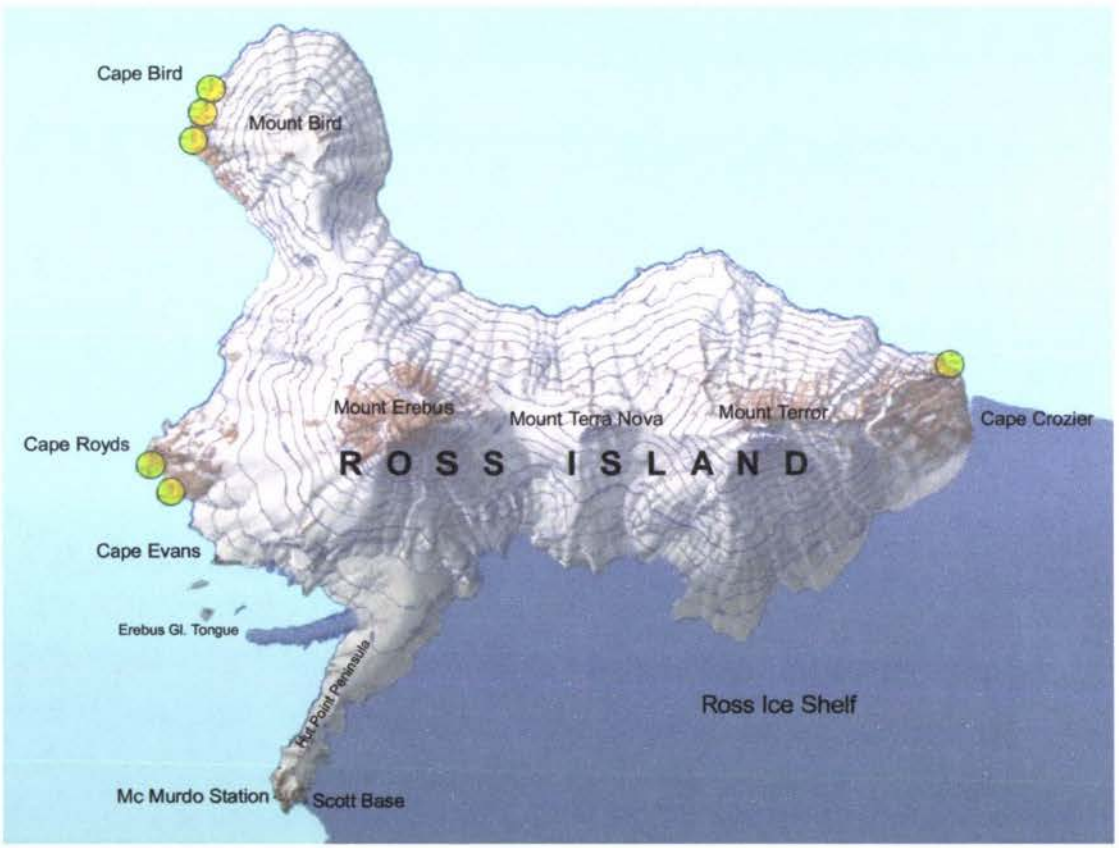


Figure 1.2: Ross Island and breeding colonies of the Adélie penguin. The Adélie penguin rookeries are indicated in yellow. At Cape Bird an Adélie colony numbering approximately 60,000 forms every year from early November to mid January.

Upon arrival at the nesting site the males, who arrive earlier than the females, begin constructing nests. Open windswept mounds and ridges are the usual location for the nests as these snow-free areas are all that is available when the Adélies arrive in early spring. The Adélies collect stones ranging in size from 1cm to 5cm and place them around the edge of a depression in the ground forming a doughnut shaped wall on which the penguin sits (figure 1.3).

Figure 1.3: An Adélie penguin upon its nest of pebbles. The nest consists of pebbles between 1cm and 5cm in diameter arranged around a depression in the ground. Nests are constructed on snow-free mounds to avoid streams and puddles when surrounding snow and ice melts. Unfortunately these regions are also exposed to harsh weather conditions.

This choice of nest location has both advantages and disadvantages. As spring turns to summer the surrounding snow and ice melts forming streams and puddles that these raised regions avoid. Unfortunately, these raised areas are also exposed to wind and during the early stages of brooding the Adélies have to contend with harsh spring weather conditions. In the early weeks of November the male Adélies may endure temperature fluctuations of approximately 20°C brought on by increased wind chill due to blowing snow. More surprisingly, on still days the Adélie is faced with a heat dissipation problem due to the zero humidity of Antarctic air. Under constant sunlight, local air temperatures can rise well above zero and on these 'hot' days nesting Adélies will lie with flippers and feet outstretched in an attempt to dissipate heat. For birds without eggs to protect, the overheating problem is solved by lying in or eating snow.

The total incubation period for Adélie eggs is between 33 and 39 days and, in 88% of cases, the male incubates the eggs for the first 14 days. During this time he fasts [1]. Since there are few other options, thermo-regulation must occur on a systemic level through variation in heart rate, metabolic rate, respiratory rate and vasoconstriction. The goal of this thesis was to develop a system capable of measuring these responses and provide an insight into the homeostatic mechanisms of the Adélie penguin.

1.4 Technology

The basic requirements of NIR spectroscopy are a monochromatic light source in the red and infrared region of the electromagnetic spectrum and a photo-detector sensitive enough to respond to the subtle changes in scattered light intensity. The options considered for light sources were either laser diodes or LEDs as other sources were impracticably large. An attractive aspect of using laser diodes includes increased incident illumination and temporal coherence. However, temperature instability, cost, power consumption and the lack of availability over a range of frequencies prohibited their use. Recent development in LED technology has seen a dramatic increase in the intensity and range of available frequencies. Combined with their cost, weight, power consumption and acceptable coherence (typical linewidth of 20nm), they were selected as the most suitable light sources for the device.

The options available as detectors included photodiodes of varying areas and construction or a photo-multiplier. The later was eliminated for cost and size reasons and of the photodiodes, a large area ($7.5mm^2$) silicon detector was selected as its cost, sensitive range and temperature stability made it favourable. Other possibilities were hybrid photo-detector/preamplifier devices, however their expense precluded their use.

Basic improvement to a NIR spectroscopic system is achieved by either increasing the intensity of the incident light or increasing the effective sensitivity of the detector. The factors considered when designing the device also included cost, weight, size, temperature stability and power consumption.

1.5 Measurement Principles

1.5.1 NEAR INFRARED SPECTROSCOPY

The biological and medical value of near infrared spectroscopy arises from the relative transparency of tissue to light in the red and near infrared regions of the electromagnetic spectrum and the presence of two natural chromophores that exhibit oxygenation dependent absorption at these wavelengths (figure 1.4). These chromophores are haemoglobin, which is present in red blood cells and is therefore an indicator of blood oxygenation, and cytochrome oxidase, which is the terminal enzyme in the mitochondrial electron transfer chain and therefore an indicator of tissue oxygenation [5]. The goal of NIR spectroscopy is to obtain absolute quantitative absorption spectra through observed changes in detected scattered light. However, differences from subject to subject in physical attributes such as skin opacity, skin thickness, blood circulation and temperature preclude single wavelength measurements due to an inability to calibrate the system. Using double wavelength techniques similar to that used by Shiga [6] and Mendelson [7], qualitative oxygen saturation data are obtained through normalisation of the absorbance data.

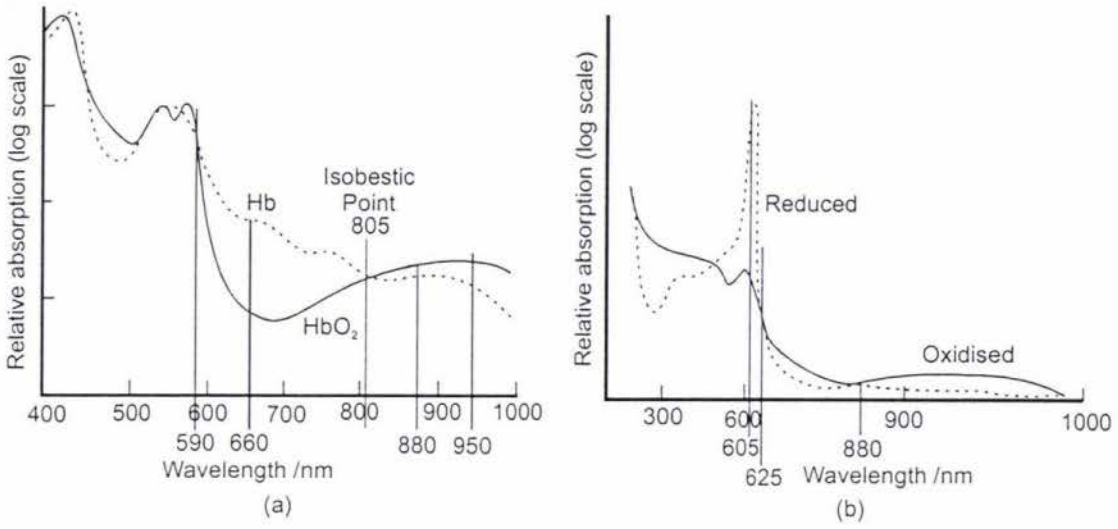


Figure 1.4: (a) Absorption spectra of oxy- and deoxy- haemoglobin and (b) oxidised and reduced cytochrome oxidase [8]. 805nm is the isobestic wavelength for oxy- and deoxy- haemoglobin, other specified wavelengths indicate the frequencies of the available light sources (LEDs).

In radiation transport, light is comprised of discrete photons that are either elastically scattered or totally absorbed according to the coefficients ε (absorption coefficient) and σ (scattering coefficient) for constituents within the tissue [8]. The Beer-Lambert law (1.3) describes the total absorbance as the sum of the absorption coefficients multiplied by the concentration of each absorber [9]. The total absorbance is related to the detected light intensity by the logarithm of the incident and transmitted light (1.4)

$$A = \sum_i \varepsilon_\lambda^{X_i} [X_i] L \tag{1.3}$$

$$A = \log_{10}(I_0 / I) \tag{1.4}$$

where I_0 and I are the intensity of the incident and transmitted light respectfully, $\varepsilon_\lambda^{X_i}$ are the absorption coefficients (at wavelengths λ) for the various absorbers (X_i) in the tissue, $[X_i]$ are the concentration of the absorbers and L is the optical path length. The following result that relates blood oxygen saturation to the measured light intensity is calculated from absorbance data measured at the two wavelengths, 660nm and 880nm. At 660nm, reduced haemoglobin absorbs considerably more than oxyhaemoglobin and at 880nm the absorbance due to oxy- and deoxy- haemoglobin is comparable. The tissue oxygenation result is derived in the same manner using the same assumptions but shorter wavelengths of 605nm and 626nm. The general oxygen saturation derivation using double wavelength measurements is given in appendix A.1.

At 660nm and 880nm it can be assumed that the contribution to the absorption by chromophores other than haemoglobin is small and, on the time scale of an observation, their contribution remains constant [6]. These terms along with optical loss and the sensitivity of the detector can be incorporated into an attenuation constant such that (1.3) may be rewritten as

$$A \propto \left(\varepsilon_\lambda^{Hb}[Hb] + \varepsilon_\lambda^{HbO_2}[HbO_2]\right) L \tag{1.5}$$

Equations (1.4) and (1.5) can be combined resulting in an equation that describes the observed intensity, I, as a function of the optical path length, the oxy- and deoxy-haemoglobin absorption coefficients, and concentrations which vary in a complementary fashion.

In (1.5) the absorbance, A, depends on the optical path length, which is unknown. Work using time-resolved or frequency-domain reflectance spectrometry has been carried out by a number of researchers (Wilson *et al.* [8], Liu *et al.* [10]) to obtain absolute, quantitative absorption data. These techniques however, have large computational and hardware requirements that are unsuitable for this application.

Given the relative transparency of tissue to red and near infrared light it can be assumed that the concentration of scatterers is much greater than the concentration of absorbers and that the degree of scattering varies insignificantly between 660nm and 880nm. That is,

$$(S >> A) \; and \; (S_{660nm} \approx S_{880nm}) \qquad (1.6)$$
$$where \; S = \sum_i \sigma_{X_i} [X_i]$$

Under these conditions, the average optical path length for both wavelengths is approximately equal (i.e. $\langle L_{660nm} \rangle \approx \langle L_{880nm} \rangle$) and, by taking the ratio of absorbances, the optical path length term may be eliminated [8].

$$\frac{A_{660}}{A_{880}} = \frac{\varepsilon_{660}^{Hb}[Hb] + \varepsilon_{660}^{HbO_2}[HbO_2]}{\varepsilon_{880}^{Hb}[Hb] + \varepsilon_{880}^{HbO_2}[HbO_2]} \qquad (1.7)$$

Using the complementary relationship between the oxy- and deoxy- haemoglobin concentration and recalling that the absorbance is proportional to the intensity signal, $A \propto \log(I_0/I)$, an equation that describes the relationship between measured light intensity and oxygen saturation is found (appendix A.1).

$$\frac{[HbO_2]}{[Hb_{total}]} = \frac{\varepsilon_{880}^{Hb} \dfrac{\log(I_{0,660}/I_{660})}{\log(I_{0,880}/I_{880})} - \varepsilon_{660}^{Hb}}{(\varepsilon_{880}^{Hb} - \varepsilon_{880}^{HbO_2}) \dfrac{\log(I_{0,660}/I_{660})}{\log(I_{0,880}/I_{880})} + \varepsilon_{660}^{HbO_2} - \varepsilon_{660}^{Hb}} \qquad (1.8)$$

The reduction state of cytochrome oxidase and measured light intensity is calculated using the same analytical method, however in this case, the difference in absorption is observed for light of wavelength, 605nm (figure 1.4).

From the absorbance relationship (1.5) an equation describing relative blood volume can also be derived by considering the absorbance at two different wavelengths

$$A_{660} \propto \varepsilon_{660}^{Hb}[Hb]L + \varepsilon_{660}^{HbO_2}[HbO_2]L \qquad (1.9)$$
$$A_{880} \propto \varepsilon_{880}^{Hb}[Hb]L + \varepsilon_{880}^{HbO_2}[HbO_2]L \qquad (1.10)$$

Obtaining either (1.9) or (1.10) in terms of $[Hb]$ and $[HbO_2]$ then combining the results gives an equation relating the total haemoglobin concentration to the absorbance (1.11). Assuming that the total haemoglobin concentration in the blood remains approximately constant, the relationship between blood volume and measured light intensity is found by substituting the absorbance relationship, $A \propto \log(I_0/I)$, into (1.11). The general derivation of relative blood volume is given in appendix A.3.

$$[Hb_{total}] \propto \frac{1}{L}\left(\frac{A_{660}\left(\varepsilon_{880}^{HbO_2} - \varepsilon_{880}^{Hb}\right) + A_{880}\left(\varepsilon_{660}^{Hb} - \varepsilon_{660}^{HbO_2}\right)}{\varepsilon_{660}^{Hb}\varepsilon_{880}^{HbO_2} - \varepsilon_{880}^{Hb}\varepsilon_{660}^{HbO_2}} \right) \tag{1.11}$$

1.5.2 TRIPLE WAVELENGTH OXYGEN SATURATION MEASUREMENT

In general, the opacity of tissue reduces for light of increasing wavelength. Within an absorption band (e.g. 590nm to 880nm for oxyhaemoglobin) the background absorbance can be estimated by interpolating between two wavelengths at which the absorbances of the oxidised and reduced states are comparable (figure 1.5). Normalising the acquired absorbance with this predicted reference point reduces the error due to non-uniform base line drift and improves the validity of assumptions made for the constant attenuation assumed in equation (1.5).

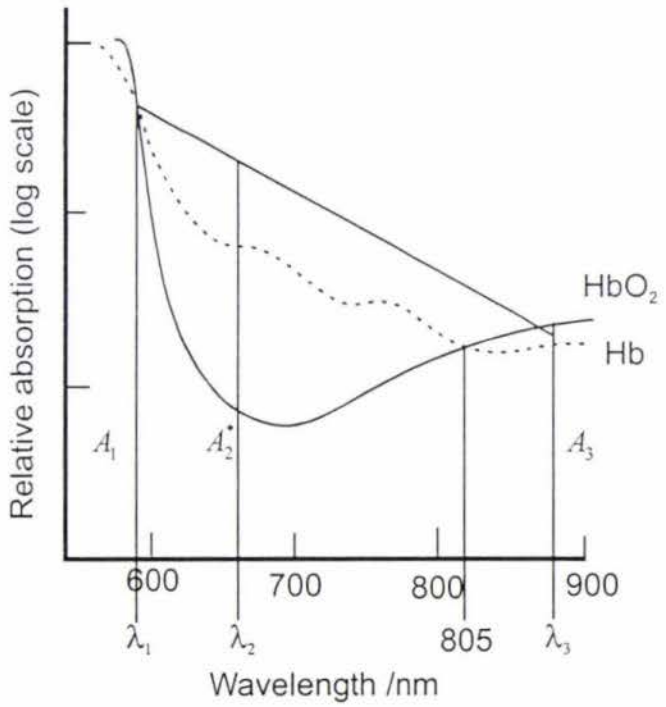


Figure 1.5: Interpolating to find an estimate of the background absorbance at wavelength, λ_2. Comparing the measured absorbance with the background estimate helps to remove the non-uniform baseline drift present in double wavelength measurements.

The equation describing oxygen saturation from triple wavelength measurements is given for the oxygenation state of haemoglobin using the wavelengths λ_1 = 590nm, λ_2 = 660nm and λ_3 = 880nm. A similar result is obtained for cytochrome oxidase

using the wavelengths $\lambda_1 = 590\text{nm}$, $\lambda_2 = 605\text{nm}$ and $\lambda_3 = 625\text{nm}$. The general derivation of triple-wavelength oxygen saturation measurement is given in appendix A.2.

Linearly interpolating between 590nm and 880nm gives an expression for the background absorbance, $A^{\bullet}_{660}$, at 660nm,

$$A^{\bullet}_{660} = \frac{A_{880} - A_{590}}{\Lambda} + A_{590} \qquad (1.12)$$

$$\text{where } \Lambda = \frac{880-590}{660-590} = 4.143$$

Defining, β, as the ratio of the measured absorbance to the background absorbance and assuming again that the mean optical path lengths are approximately equal at all three wavelengths gives the following expression

$$\beta = \frac{A_{660}}{A^{\bullet}_{660}} = \frac{\Lambda\left(\varepsilon^{Hb}_{660}[Hb] + \varepsilon^{HbO_2}_{660}[HbO_2]\right)}{\left(\varepsilon^{Hb}_{880}[Hb] + \varepsilon^{HbO_2}_{880}[HbO_2]\right) + \left(\varepsilon^{Hb}_{590}[Hb] + \varepsilon^{HbO_2}_{590}[HbO_2]\right)(\Lambda - 1)} \qquad (1.13)$$

As before, the complementary relationship between the oxy- and deoxy- haemoglobin concentration is used allowing equation (1.13) to be solved for the oxygen saturation giving,

$$\frac{[HbO_2]}{[Hb_{total}]} = \frac{\beta\left(\varepsilon^{Hb}_{880} + (\Lambda-1)\varepsilon^{Hb}_{590}\right) - \Lambda\varepsilon^{Hb}_{660}}{\Lambda\left(\varepsilon^{HbO_2}_{660} - \varepsilon^{Hb}_{660}\right) + \beta\left[(\Lambda-1)\left(\varepsilon^{Hb}_{590} - \varepsilon^{HbO_2}_{590}\right) + \varepsilon^{Hb}_{880} - \varepsilon^{HbO_2}_{880}\right]} \qquad (1.14)$$

To obtain the direct relationship between measured intensity and oxygen saturation, the absorbance relationship, $A \propto \log(I_0/I)$, is substituted into β

$$\beta = \frac{\Lambda A_{660}}{A_{880} + A_{590}(\Lambda - 1)} = \frac{\Lambda \log\left(I_{0,660}/I_{660}\right)}{\log\left(I_{0,880}/I_{880}\right) + \log\left(I_{0,590}/I_{590}\right)(\Lambda - 1)} \qquad (1.15)$$

In both the double and triple wavelength calculations, relative blood volume, blood oxygenation and tissue oxygenation are found to be functions of the intensity signal and absorption coefficients only. Using data acquired from the device and absorption information from the literature allowed the metabolic state of the subject to be described.

1.5.3 NEAR INFRARED SPECTROMETRY

Near-infrared spectroscopic systems are usually arranged in one of two configurations, transmission mode or reflectance mode (figure 1.6). Established clinical and research devices such as the Wood-Geraci ear-oximeter and the Hewlett-Packard eight-wavelength oximeter are all transmission mode devices [11]. The use of reflectance mode spectrometry was introduced by Brinkman and Zijlystra in 1949 who showed that changes in oxyhaemoglobin saturation could be recorded non-invasively from an

optical sensor attached to the forehead [7]. Reflectance mode oximeters however, have not achieved widespread commercial use due to limited accuracy and difficulties in absolute calibration. For the intended application, absolute calibration of the device was not required, as the main objective was to demonstrate a correlation between environmental conditions and the relative changes in the metabolic response of the Adélie penguin. In this device limitations in accuracy were reduced by the greater intensity of modern LED technology and calculation techniques such as the triple wavelength measurement (section 1.5.2).

Aside from the differences in construction and calibration, the physical basis for both transmission and reflectance mode spectroscopy is the same for measurements of completely diffuse light (i.e. the photon distribution within the medium retains no information about initial direction). Photon diffusion analysis by Kumar and Schmitt [12] has shown that, with a source and detector spacing of greater than 2mm, a collimated incident light source is equivalent to a diffuse source located below the surface in an optically turbid medium such as tissue. Since the Beer-Lambert law describes a measured intensity in terms of the photon path length and the incident light source may be considered diffuse, the detected signal for both transmission and reflectance mode spectroscopy is equivalent.

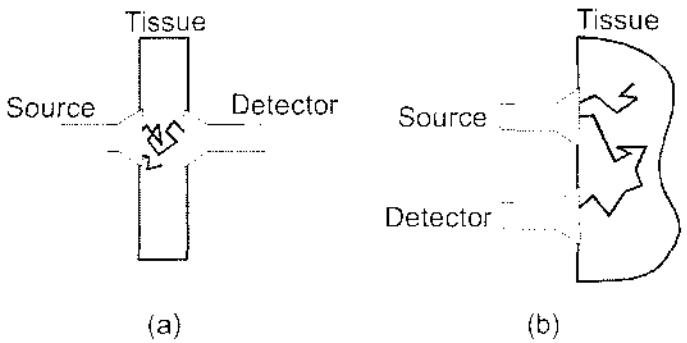


Figure 1.6: The two main configurations for near infrared spectroscopic systems, (a) transmission mode and (b) reflectance mode. For distances greater than 2mm from the incident light source the scattered light may be considered a diffuse light source below the surface. As diffuse light is independent of direction both transmission and reflectance mode spectroscopy are equivalent.

1.6 Thesis Overview

The work undertaken in this thesis involves the design and development of a NIR spectroscopic device. Using the principles and techniques described above, a system was developed that not only collected the necessary physiological data but also addressed some of the difficulties of working in Antarctica and with the Adélie penguin. In the instrumentation chapter that follows, the hardware is assembled along with justification for the components selected. A logical division between measurement and control exists that divides the hardware into analogue and digital stages respectively. The acquired signal is followed through the various analogue processes to the point of digitisation where focus is then moved to the control of the device by the digital components.

The third chapter examines the control of the device from a software perspective. It gives a description of the operating system and the interaction between measurement sequence files, the terminal emulation software and the embedded processor. The final section of this chapter describes the algorithm used to determine pulse rate from the fluctuating scattered light signal and gives analysis of the signal processing techniques used to overcome noise.

Chapter four begins by describing the incremental development of the device and the results of the validation steps taken at each stage. Reasons for each new prototype and the increased functionality that each system allowed are described in the logical order in which they were developed and the conclusion to this section gives the test results of the final prototype version of the device. The second part of chapter four describes the results obtained during field-testing. Included are the physical aspects of the experiments, such as capture, attachment and behavioural response, through to the biological results obtained in response to stress and temperature changes: oxyhaemoglobin saturation, cytochrome oxidase saturation, blood volume and pulse rate. A discussion of the acquired data follows in the conclusion chapter that then lead to an evaluation of the device, its limitations and various suggested improvements. Long-term enhancements conclude chapter five with an outlook toward the potential future of the device in environments as equally diverse as that of the Adélie penguin.

Chapter 2

Instrumentation

An optical device was designed and constructed to provide an insight into the metabolic responses of the Adélie penguin. This necessitated a compromise between the constraints arising from working with penguins and the requirements of the hardware.

2.1 Design Specifications

The biological information to be measured included pulse rate, blood oxygenation, the reduction state of cytochrome oxidase and relative blood volume. As these variables are calculated from changes in the absorption of light at different wavelengths (section 1.5.1), a system to record this absorption data was needed. Experiments designed to measure these variables could last only a few hours with continuous sampling or for several days using less frequent sampling. The device had to be equipped with timing facilities and have the ability to store the acquired data for retrieval after the measurement period.

As the device was to be fitted to an Adélie penguin, there were a number of physical restrictions that also had to be considered. The average weight of an adult Adélie penguin is approximately 3.5 – 4.5 kg [13] and the device had to weigh only 2 – 3% of this ($\leq$ 150g) to minimise restriction of the birds normal activities such as walking, jumping, stone collecting or egg incubating. For the same reasons, minimising the package dimensions and careful consideration of shape were necessary. Research into the swimming energetics of instrumented penguins shows an increased level of energy expenditure with even relatively small instruments attached (<2% of body cross-sectional area) [14]. The added complexity of waterproofing and streamlining the device packaging was avoided by conducting experiments during the penguins breeding period, where they spend the majority of their time on land. On the occasions that the penguin intended to go to sea, the bird was recaptured and the device removed. Waterproofing of the device, however, was still necessary to prevent problems from melted snow or ice.

Minimising awareness of the device, by either the individual bird or its neighbours, was important for reducing stress. Research into the most suitable package colour has shown that colours similar to that of the bird's plumage are interfered with significantly less than other colours [15] and so all exterior surfaces of the device were coloured black.

2.2 Design Overview

Measurements were made from the ulnar artery and deep ulnar vein located at the proximal end of the penguin flipper's medial side [16]. Attaching a device with the necessary functionality outlined in the design specifications to this location was not possible so a sensor head and control unit arrangement was employed (figure 2.1).

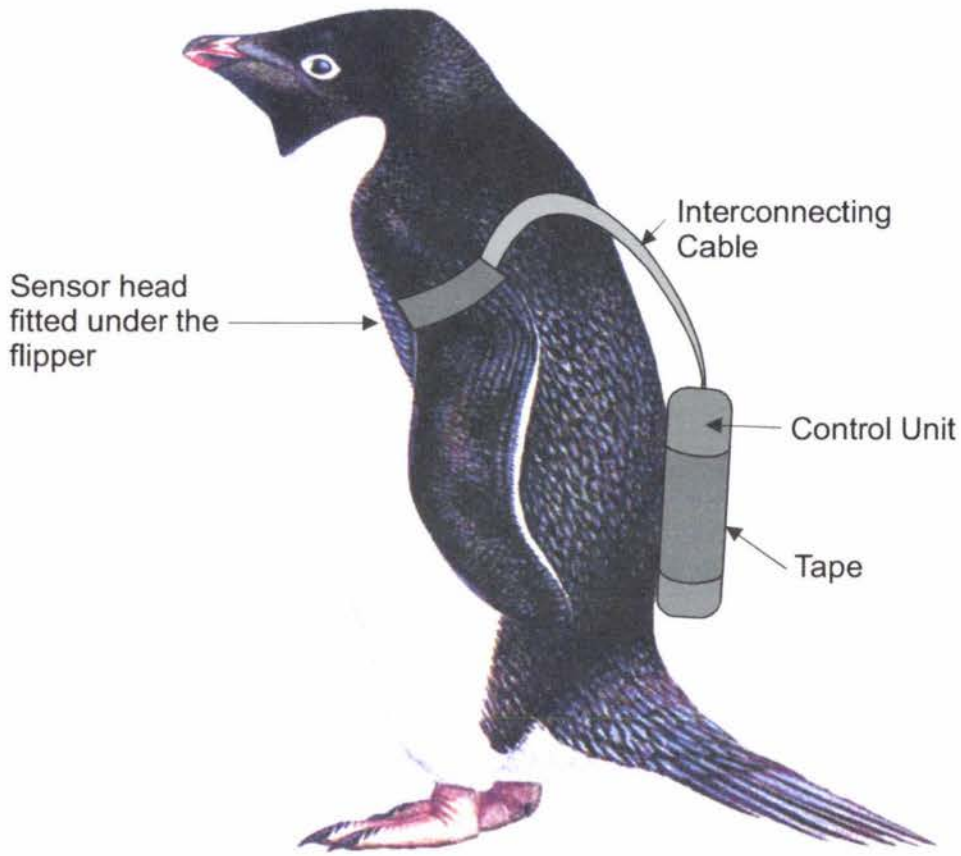


Figure 2.1: Adélie penguin with a control unit and sensor head fitted. The control unit, tape and interconnecting cable were coloured black to be less obvious to the penguins.

A plastic case, measuring 65mm x 92mm x 25mm and sealed by a plug fitted with an O-ring, enclosed the control unit protecting it from water and interference by the penguin (figure 2.2). The sensor head was connected to the main unit by an interconnecting cable soldered to a row of header pins and mounted into the plug using epoxy glue. The connecting cable linked the control unit, located on the lower back of the penguin, to the sensor head that was taped to the underside of the penguin flipper.

At the sensor head, LEDs of various wavelengths transmitted light into the tissue. Some of the scattered light was received by a photodiode (also mounted on the sensor head) that converted the light signal into an electrical signal. This was then amplified and filtered before being digitised by an analogue to digital converter and processed by a microcontroller in the control unit. Co-ordination of this process, measurement sequence interpretation, serial communication and power management were all done by the microcontroller. Finally, the acquired data were stored to static memory for later retrieval.

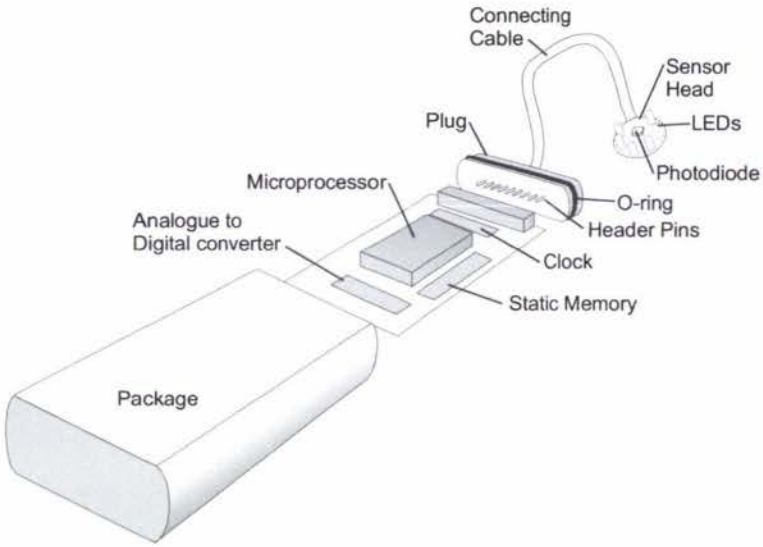


Figure 2.2: The connectivity and packaging of the sensor head and control unit.

Information was transferred between the device and a computer via an RS-232 serial port. Using this communication, measurement instructions could be downloaded into the microcontroller's memory where they were interpreted and executed (section 3.2.3). As measurement periods could last anywhere from a few hours to several days, two important features of the device were the timing and power management capabilities. A microcontroller feature was used in conjunction with a real time clock to allow the processor to switch in and out of its power saving state at particular times.

The final realisation of the device is shown in figure 2.3. The total weight of the control unit and sensor head (including the case and interconnecting cable) was 132.7g and the control unit had a frontal cross sectional area of approximately 1600 mm^2.

Figure 2.3: The final realisation of the device.

2.3 Analogue Circuitry

2.3.1 THE SENSOR HEAD

Optical measurements were made by the sensor head located over the ulnar artery under the penguin flipper. The sensor head was 28mm in diameter and 12mm thick. It carried LEDs of six different wavelengths and a temperature sensor arranged equidistantly around a photodiode (figure 2.4). As suggested by Kumar and Schmitt [12] a source – detector spacing of 5mm was used. This spacing is suitable for shallow tissue absorption measurements and provides adequate signal intensity given the power limitations of the device. Also, the feasibility of this LED-photodiode arrangement was verified by preliminary experiments. In a similar device constructed by Shiga *et al.* [6] dual wavelength LEDs were mounted 30mm from the optical detector. Shiga states that this distance is suitable for making muscle tissue measurements but requires greater power to achieve measurable signal strength.

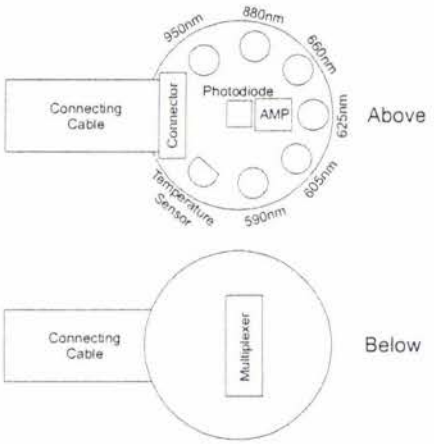


Figure 2.4: The sensor head layout. The sensor head was 28mm in diameter and 12mm thick. The view labelled 'Above' is the side that contacted the penguin flipper.

Selection of LEDs for the sensor head was based on the availability of wavelengths as close as possible to the peak differences and isobestic points of the haemoglobin and cytochrome oxidase absorption spectra (figure 1.4). The LEDs chosen had peak outputs centred at 950nm, 880nm, 660nm, 625nm, 605nm and 590nm of which the 880nm, 660nm and 590nm were used for blood oxygenation experiments and the 625nm, 605nm and 590nm were used for tissue oxygenation experiments. The 950nm LED was included as an extra wavelength for the blood oxygenation measurements since many similar systems use 660nm and 950nm for their double wavelength measurements [7].

Individual control of each LED by the microcontroller was achieved using three data lines and an eight channel surface mount multiplexer. The major advantage of this was to reduce the number of connections between the sensor head and the control unit. Using a three-bit address the microcontroller is able to select each of the six LEDs. Each LED was modulated when in use (section 2.3.2) so to achieve this, one of the eight multiplexer channels was left unconnected and by switching between the required LED channel and the unconnected channel the LED was modulated.

Light, emitted from the LEDs and scattered by the tissue, was collected by a large area photodiode. The BPW34 photodiode was chosen for its large radiant sensitive area (7.5mm^2) and high photosensitivity in the visible and infrared regions. Exposure of the detected signals to interference was minimised by mounting a current to voltage converter with gain as close as possible to the photodiode. This converter also acted as a preamplifier and was constructed using an OP07, low offset voltage and low bias current, operational amplifier. The benefit of this amplifier was its low noise characteristics making it ideal as the preamplifier.

An estimate of the local skin temperature was obtained from a temperature sensor located on the sensor head. The sensor was directly calibrated and had a linear response to changes in temperature of 10mV/K. The sensor head is summarised in figure 2.5 and a complete circuit diagram is given in appendix C.2.

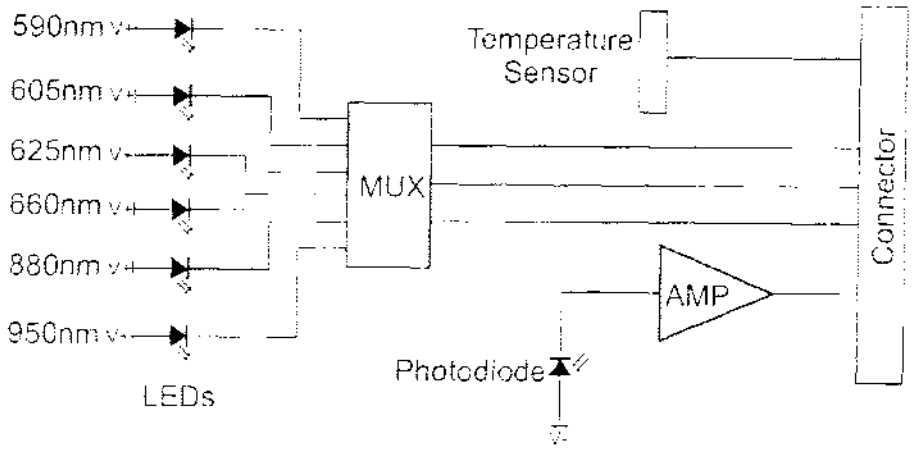


Figure 2.5: The sensor head circuit. An eight channel multiplexer was used to reduce the number of connections between the LEDs on the sensor head and the control unit. A complete circuit diagram of the sensor head is given in appendix C.2.

2.3.2 THE SYNCHRONOUS DETECTOR

One of the most useful experimental techniques for increasing the signal to noise ratio of a noisy signal is synchronous or phase-sensitive detection. The underlying principle of this technique is to shift the frequency of the signal of interest (usually near dc) into a 'quiet' band of frequencies. A high pass or band pass filter is then used to remove the noise components outside of this frequency range so that, after demodulation, the original signal is reconstructed without the original noise [17]. The explanation that follows describes the synchronous detector implemented for the device (figure 2.6) while a more general analysis of synchronous detection is given in section 2.3.3.

Consider the signal resulting from the photodiode on the sensor head if it were under constant illumination. The major factors contributing to noise and interference in this signal would be changes in background light levels, thermal drift and electrical interference (e.g. the switching of the LED address lines). Separating this noise from the true signal resulting from changes in scattered light would be a near impossible task.

Suppose now that the light source is modulated by a reference square wave. The resulting current through the photodiode detector will now also contain a square wave in phase with the light source (reference point 1 in figure 2.6). The magnitude of this square wave is proportional to the scattered light signal only and is synchronous with the reference. Since this ac signal is the only signal of interest the offset voltage (background dc interference signal) can be simply removed using a coupling capacitor such that the waveform becomes centred about zero (reference point 2 in figure 2.6). This ac signal tends to be small (of the order 10 – 100mV) so it is amplified before it is passed through the phase sensitive detector which uses the modulation wave to toggle an electrical switch between the signal and its inverse. As the modulation wave and signal are in phase the effect of this switching is to full-wave rectify the signal (reference point 3 in figure 2.6). By passing this signal through a simple low pass filter a smooth dc waveform proportional to the scattered light detected by the photodiode is obtained (reference point 4 in figure 2.6).

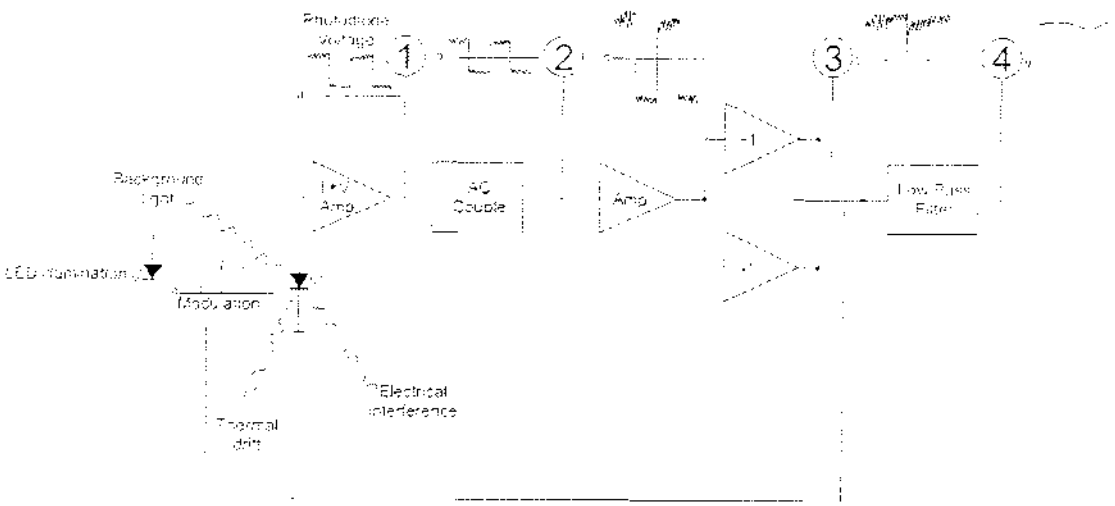


Figure 2.6: A graphical representation of the implemented synchronous detector. 1) The raw voltage proportional to the current through the photodiode with a background offset. 2) The raw signal with the background offset removed. 3) Full wave rectification of the amplified signal. 4) A waveform directly proportional to the illumination of the photodiode by the LED.

2.3.3 SYNCHRONOUS DETECTION

The fundamentals of the synchronous detector can be seen by analysing the effect of the detector on an arbitrary sine wave. In thinking of the detector as a black box, the general noise signals that are present at the input consist of the supposition of numerous sine waves each of varying amplitude, frequency and phase. Examining the resulting output of the detector to these signals explains how an improved signal to noise is achieved. Consider first of all a sinusoidal input signal (2.1) that is in phase and has the same frequency as the switching frequency ($f_0 = \omega_0/2\pi$) of the detector (figure 2.7).

$$V_{sin} = V_0 \sin(\omega_0 t) \tag{2.1}$$

The switching process of the detector has the effect of full wave rectifying (2.1) generating (2.2) as the result.

$$V'_{sin} = V_0 |\sin(\omega_0 t)| \tag{2.2}$$

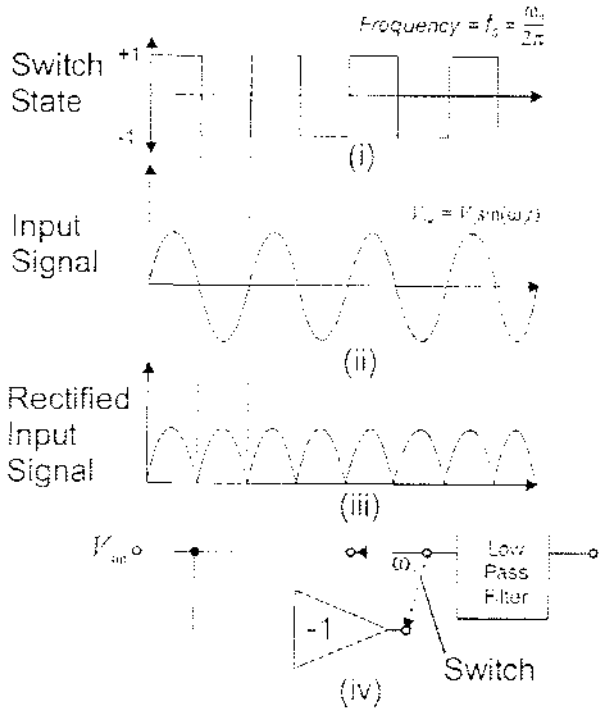


Figure 2.7: Demodulation of an in-phase, input sine wave. (i) The detector switch state; (ii) an input sine wave signal in phase and of the same frequency as the switching of the detector; (iii) the full-wave rectified sine wave; and (iv) the full-wave detector and low pass filter.

Decomposing (2.2) into a Fourier series of the form (2.3) [18] gives (2.4)

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(2\pi n f_0 t) + b_n \sin(2\pi n f_0 t)] \tag{2.3}$$

$$a_n = \frac{2}{T} \int_{-T_0/2}^{T_0/2} f(t)\cos(2\pi n f_0 t)dt, \; n = 0,1,2,\ldots, \tag{2.3a}$$

$$b_n = \frac{2}{T} \int_{-T_0/2}^{T_0/2} f(t)\sin(2\pi n f_0 t)dt, \; n = 1,2,3,\ldots, \tag{2.3b}$$

$$V'_{sin} = \frac{2V_0}{\pi} - \frac{4V_0}{3\pi}\cos(2\omega_0 t) - \frac{4V_0}{15\pi}\cos(4\omega_0 t) - \frac{4V_0}{35\pi}\cos(6\omega_0 t) - \ldots \tag{2.4}$$

If a low pass filter is designed with unity gain at dc and a corner frequency much less than $2\omega_0$ (i.e., $1/RC << 2\omega_0$) then only the dc component of (2.4) will pass and the output of the filter ($2V_0/\pi$) is proportional to the amplitude of the input signal only (figure 2.8).

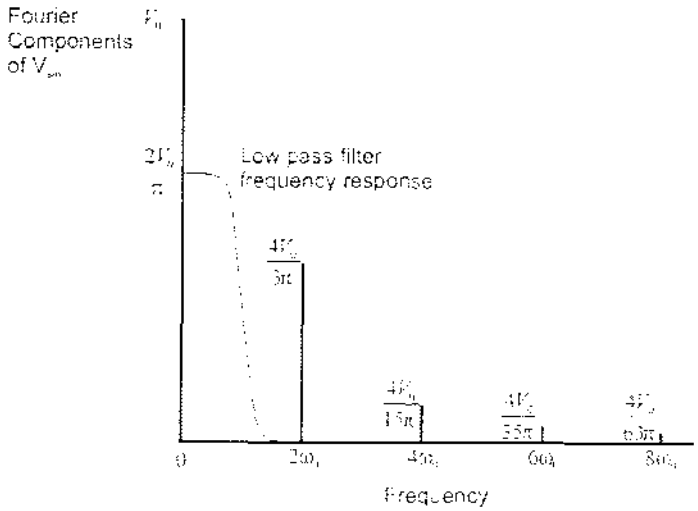


Figure 2.8: The Fourier components of the full-wave rectified sine wave. A low pass filter with unity dc gain and a corner frequency much less than $2\omega_0$ will pass only the dc component of the waveform.

Consider now a sinusoidal input signal of the same frequency as the switch frequency but out of phase by ϕ (where $-\pi < \phi < \pi$).

$$V_{out} = V_0 \sin(\omega_0 t + \phi) \tag{2.5}$$

The dc component of (2.5) at the output of the detector is now

$$\frac{a_0}{2} = \frac{4V_0}{2T_0}\int_0^{T_0/2} \sin(\omega_0 t + \phi)\, dt = \frac{2V_0}{\pi}\cos(\phi) \tag{2.6}$$

Using the same low pass filter (i.e., with $1/RC << 2\omega_0$) the output is proportional to the amplitude of the input signal and the cosine of the phase difference between the signal and the detector switch. Clearly, this dc output is less than the output signal resulting from (2.4) that has $\phi = 0$ so this result implies that the detector is phase selective.

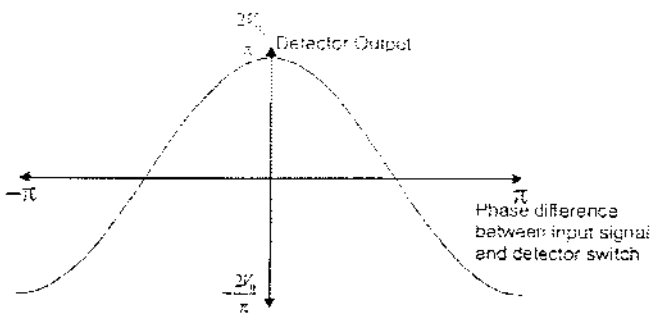


Figure 2.9: The phase selectivity of the detector. The magnitude of the input sine wave dc component varies as $(2V_0/\pi)\cos(\phi)$ for a phase difference of ϕ between the input sine wave and the detector switch.

Noise at the input of the detector consists of the supposition of sine waves of all frequencies. Therefore, the contribution to the detector output by frequencies other than the modulation frequency is of interest. Consider a sinusoidal input signal with an arbitrary frequency, f.

$$\omega = 2\pi f$$
$$V_{sin} = V_0 \sin(\omega t) \tag{2.7}$$

Taking a converse approach to that previously, the Fourier decomposition of the detector switch signal (the modulation square wave) is first calculated

$$Switch\ Signal(t) = \frac{4}{\pi}\left(\sin(\omega_0 t) + \frac{\sin(3\omega_0 t)}{3} + \frac{\sin(5\omega_0 t)}{5} + \ldots\right) \tag{2.8}$$

Multiplying (2.8) by the arbitrary sine wave, (2.7), gives the Fourier components of the signal at the detector switch output.

$$V'_{sin} = \sum_{n=1,3,5,\ldots}^{\infty} \frac{2V_0}{n\pi}\left[\cos(\omega t - n\omega_0 t) - \cos(\omega t + n\omega_0 t)\right] \tag{2.9}$$

Equation 2.9 shows that the frequencies present after the detector switch are the sum and difference of the switch frequency and the input signal frequency and in the special case of $\omega = \omega_0$, (2.9) simplifies to (2.4).

If the low pass filter stage is constructed with a corner frequency, $\omega_c\ (= 1/RC) << \omega_0$, then the only surviving frequency components after the low pass filter will be the difference terms.

$$V'_{sin} = \sum_{n=1,3,5,\ldots}^{\infty} \frac{2V_0}{n\pi}\cos(\omega t - n\omega_0 t), \tag{2.10}$$

$$with \quad -\omega_c < (\omega - n\omega_0) < \omega_c \tag{2.11}$$

The restriction of (2.11), that the difference frequency must be less than the low pass filter corner frequency, defines the bandwidth of the detector equal to $2\omega_c$. Figure 2.10 shows a plot of the analytical solution (equation 2.9, appendix B.1) and physical data collected for a detector using 1kHz modulation and a low pass filter with a corner frequency at 1Hz. Note that figure 2.10 is not a plot of the transfer function of the demodulator but a plot of the contribution that sinusoids of various frequencies make to the output. The figure shows that signals synchronous with the detector switch (or signals modulated at the detector switch frequency) have the most significant contribution to the demodulated output. Unwanted signals, with frequencies very near the modulation frequency, are still significantly large at the detector output and appear as low frequency ripple on the true signal. For this reason the modulation frequency is selected in a 'quiet' frequency range so that the effect of these unwanted signals is minimised.

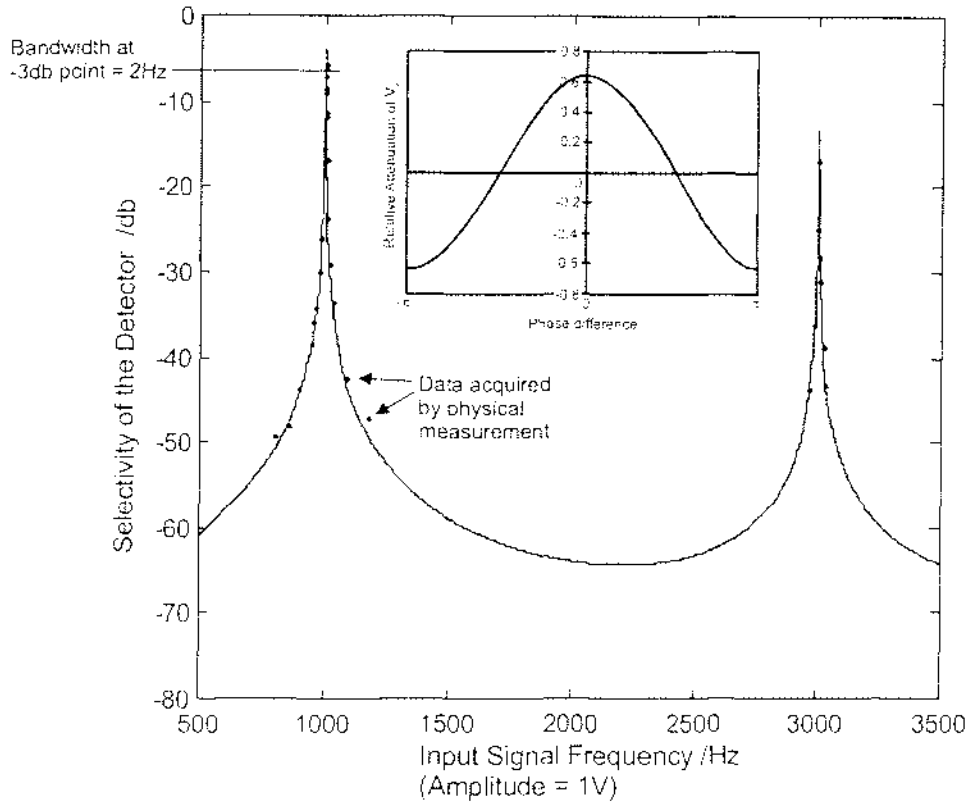


Figure 2.10: The frequency and phase response of the synchronous detector. These data were calculated for a synchronous detector using 1kHz modulation (ω_0 = 1kHz) and a low pass filter with a corner frequency at 1Hz. The bandwidth of the dominant peak is 2Hz equal to twice the low pass filter corner frequency. The peak at $3\omega_0$ is predicted in equation 2.11 when $\omega = 3\omega_0$ and $n = 3$. The C++ program written to generate the data for the synchronous detector is given in appendix B.1.

2.3.4 THE ANALOGUE STAGE

On the device, the synchronous detector was implemented using a modulation frequency of ~1kHz and a low pass filter corner frequency of 3.4Hz. The demodulator was a DG419 analogue switch connected to a passive first order low pass filter. Since the function of the synchronous detector does not depend critically on the selectivity of the low pass filter a simple filter design was chosen to minimise weight and the circuit board area required.

The remainder of the analogue circuitry (figure 2.11) is concerned with adjusting the signal levels so that they lie within the range of the analogue to digital converter (0V to 5V). The two input signals from the sensor head were converted into four outputs, signal-out, high-gain signal, pulse signal and the temperature signal. The first two of these were for LED measurements. The signal-out output was calibrated for the 950nm, 880nm, 660nm and 625nm LEDs and the high-gain output was for the shorter wavelength LEDs (605nm and 590nm) from which the signal strengths were much weaker. The pulse signal output was designed to amplify the signal oscillations to improve the computation of the pulse period (section 3.4). Most of the dc offset was removed by level shifting the signal to approximately 0.1V and then amplifying by a factor of 19. This increased the ac component of the waveform while keeping the

signal within the digitisation range. Finally, the temperature signal output was designed to increase the sensitivity of the temperature sensor by subtracting a dc offset and increasing the sensitivity from 10mV/K to 40mV/K. This meant that the temperature sensor was no longer directly calibrated in Kelvin but temperature measurements became four times more sensitive.

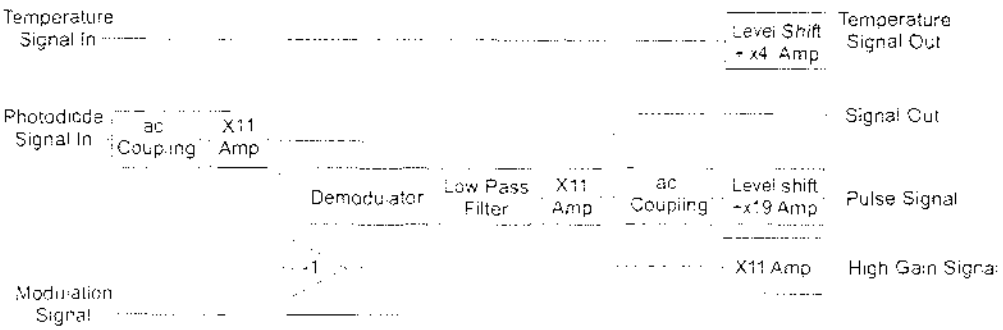


Figure 2.11: The amplification and filtering stage of the device. This analogue section of the device converts signals from the sensor head into measurable signal outputs ready for digitisation. A complete circuit diagram is given in appendix C.3.

2.4 Digital Circuitry

The control unit's digital circuitry consisted of four functional blocks: the microcontroller, analogue to digital converter (ADC), real time clock (RTC) and serial EEPROM (figure 2.12). The latter three (slave devices) were connected to the microcontroller (master device) using the Motorola Serial Peripheral Interface (SPI) [19][20] which is a three-wire system for communication between digital devices. A complete circuit diagram for the digital circuitry is given in appendix C.4.

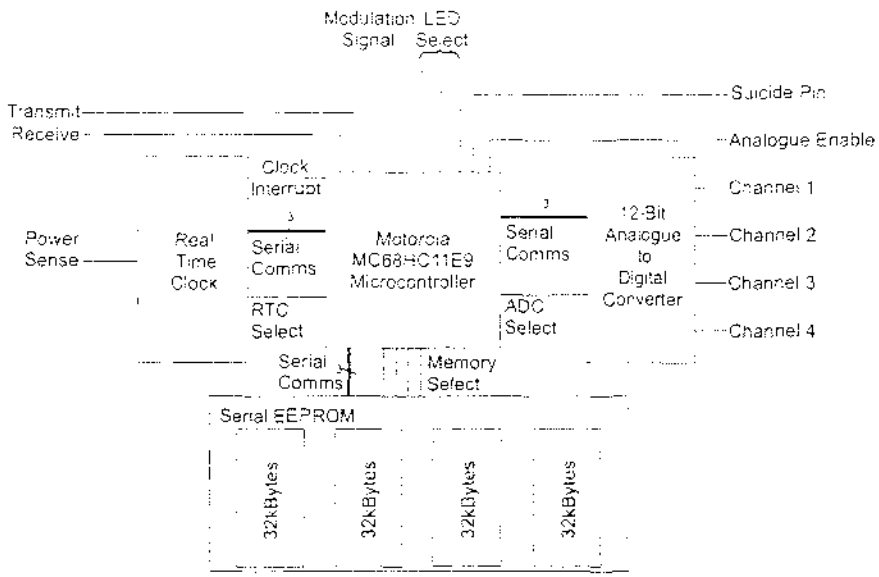


Figure 2.12: The digital circuit components. Each of the slave devices (RTC, ADC, memory) were connected to the microcontroller via the serial peripheral interface (indicated by Serial Comms).

For the microcontroller to send and receive data from any slave device it first had to deselect all other devices connected to the serial data lines to avoid data collision. Once a single slave device was selected, data were transmitted and received bit by bit, synchronised by the serial clock (figure 2.13). When 8 bits had been shifted, two complete bytes of data were been swapped between the master and slave device. In each case the received byte was transferred to a data buffer which was then internally accessible by each device. Using this form of communication the microcontroller was able to sample data using the ADC, at timed intervals controlled by the RTC, process the data and store the results into memory.

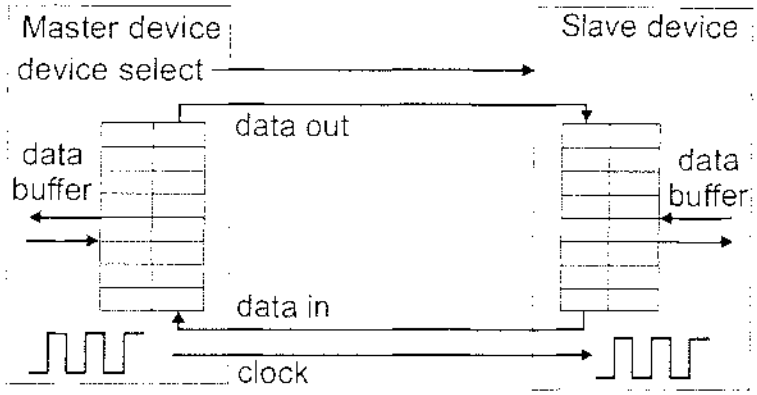


Figure 2.13: Serial communication between digital devices using the Motorola serial peripheral interface.

2.4.1 SAMPLING AND DIGITISATION

At the end of the analogue stage there were four outputs, each used depending on the measurement to be made (section 2.3.4; figure 2.11). These were connected to the four channels of the LTC1594 analogue to digital converter [21], the features of which included 12-bit resolution, low supply current and automatic shutdown. A major consideration when selecting components for the device was power consumption (section 2.4.5.3). An attractive feature of this ADC is that it draws only 320μA during conversion and then drops automatically to approximately 1nA between conversions. Therefore the ADC could remain constantly connected to the power supply and high-resolution data could be sampled at any time without concern for power consumption.

To collect a sample using the ADC, a byte (with the last 4 bits containing port number information) was sent while the chip-select pin was high (unselected). Before the next SPI clock cycle the ADC chip was selected (pin set low) and two consecutive bytes (16 bits) were read by the microcontroller. The 12-bit sample value was retrieved by concatenating the two bytes and reading bits 2 through 13. The bits outside this range could be used for error detection.

2.4.2 THE MICROCONTROLLER

The microcontroller used to control the device was the Motorola XC68HC711E9CF-S2. This device was chosen because there was local knowledge and experience with the 68HC11 and also because it was readily available. A block diagram of the controller is given in figure 2.14 showing the functional blocks and input and output connections. Some of the features that prompted its use include:

- 12k bytes of erasable programmable read-only memory (EPROM)
- 512 bytes of electrically erasable programmable read-only memory (EEPROM)
- 512 bytes of static RAM
- Serial peripheral interface (SPI)
- Serial communication interface (SCI)
- Eight channel 8-bit analogue to digital converter
- 16 bit timer system with output compare functions
- Power saving mode via the STOP instruction

In the final implementation of the device the control software (chapter 3) was stored in the 12K bytes of EPROM, the 512 bytes of EEPROM were used to store measurement sequence information and the 512 bytes of RAM were used for system variables.

2.4.2.1 Output Compare

The 'output-compare' functionality of the microcontroller was used to implement the modulation of the LEDs. Each output-compare had an associated register that was used to trigger an interrupt and optionally a pin on port A of the microcontroller. An output-compare was triggered when the value of the internal clock (represented by a 16-bit free running counter) equalled the data held in one of the output-compare registers.

LED modulation on the device was achieved using the first output-compare function (OC1). A special feature of OC1 is the ability to control any of the pins of port A. During initialisation of the controller, OC1 was configured to control pins 1 through 4, which were connected to the multiplexer on the sensor head (section 2.3.1) and to the demodulator of the synchronous detector (figure 2.11). LED modulation was initiated by setting the value of the OC1 register to a time in the future. The state for port A was assigned into the OC1 data register and the interrupt mask was removed. When the interrupt occurred, the contents of the OC1 data register were mapped to the pins of port A and program execution entered the OC1 interrupt routine.

Within the interrupt routine, the value of half the modulation period was added to the OC1 register so that the next interrupt would occur half a period later. The OC1 data register was assigned a value of either zero or an LED address, depending on its previous state.

Periodic sampling was achieved in a slightly different manner using output compare 5 (OC5). In this case OC5 was not configured to control a pin of port A but only to generate an interrupt. A sample was acquired during the interrupt routine and the time period until the next sample was added to the OC5 register.

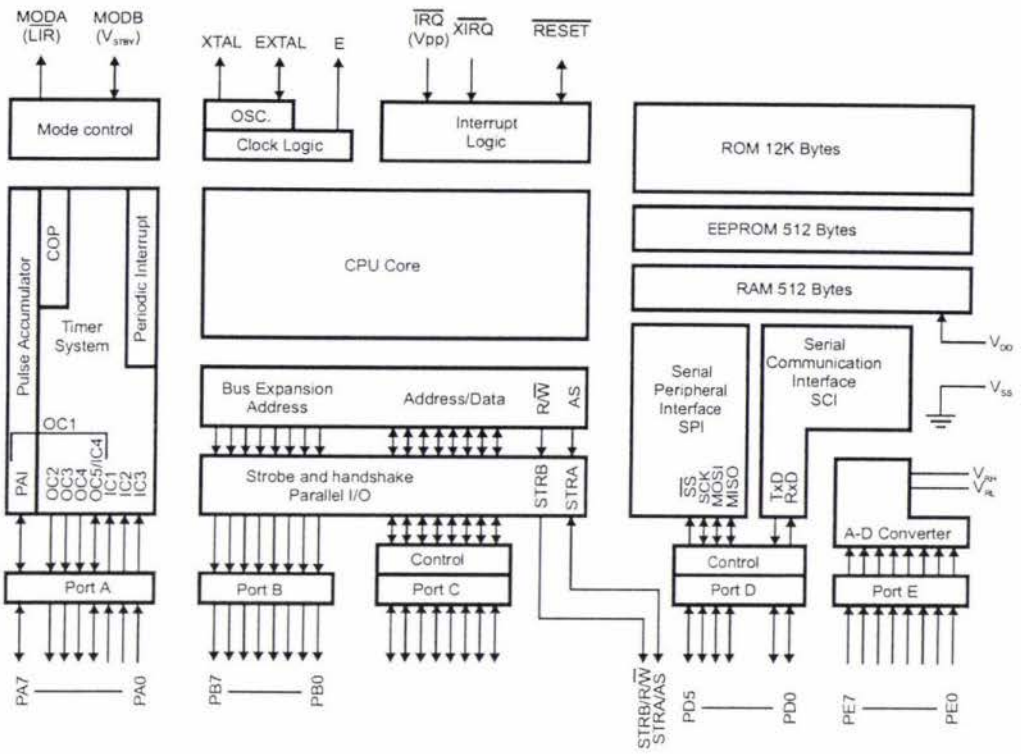


Figure 2.14: Block diagram of the 68HC11E9 microcontroller.

2.4.2.2 Interrupts

Another feature of the microcontroller was the ability to assign the highest priority to a particular user interrupt. The pulse measurement algorithm (section 3.4), used to calculate the period of a pulse waveform, relied on the sampling frequency being known. This was ensured by assigning the highest priority to the sampling output compare interrupt. Other interrupts used by the device were OC1 (for LED modulation) and the external non-maskable interrupt XIRQ (for receiving status from the real time clock).

2.4.2.3 Real Time Clock

The Motorola MC68HC68T1 real time clock (RTC) [22] was used for timing over long time periods, for power supply level sensing, and recovery from power saving mode (section 3.2.2.3). Periodic interrupts, alarm interrupts, and power sense interrupts generated by the RTC were communicated through the microcontroller's non-maskable interrupt pin. When one of these events occurred, the microcontroller received an XIRQ interrupt. The interrupt service routine read a byte from the RTC containing status information that was saved in a system variable. It was then up to the operating system software to parse the status byte to determine the reason for the interrupt and to respond accordingly.

2.4.2.4 Additional Features of the Microcontroller

Five ports were available on the microcontroller (figure 2.14) each with a specific purpose. Port A was used in conjunction with the output compare function as described earlier in section 2.4.2.1. Port B was a general purpose output port used in this application to select between the various SPI devices (ADC and memory), to enable the analogue circuitry (section 2.4.5.2) and to shut down the system after detecting a failing power supply (section 2.4.5). Port C of the microcontroller is for

general purpose input and output but was not used in this application. The pins of port D were used for serial communication for both the SPI and the SCI (section 2.4.4). Through this port instructions were received from a computer, status was read from the real time clock and data were transferred between the ADC and memory. Finally, port E was the input to an 8 channel, 8-bit, on-chip analogue to digital converter. This ADC was found to have inadequate resolution for this application and an external 12-bit converter was used in it place.

2.4.3 MEMORY

The acquired data were stored in four banks of 32Kbyte EEPROM (AT25256 [23]) totalling 128Kbytes of available storage space. Each chip was connected to the SPI data lines and selected using port B of the microcontroller. At one sample per second (each sample had 12 bits so two bytes were used to store each sample value) there was enough available storage for 17 hours of data collection. The rate of collection varied depending on the nature of the measurement sequence so by introducing a delay into the measurement sequence, extended collection periods were possible.

Data were written to the memory chips using a 64-byte page write technique. Each memory chip had 64 bytes of RAM that could be mapped to any page boundary within memory. This writing technique began by enabling the appropriate memory chip, sending a write enable command, deselecting the memory chip and then waiting until the chip entered the 'write enabled' state. Once in this state the memory chip was again enabled and a 16-bit address sent in the form of two bytes, most significant byte first. After this, the memory chip was ready to store consecutive bytes of data. When writing a data stream was completed or a page boundary was encountered the memory chip had to be deselected and then reselected in order to continue writing data. The memory chips automatically returned to the write disabled state when deselected and a write cycle time of 5 to 10ms occurred before the next stream of data could be written. The sequence of sending a write enable command first, reduced the risk of accidental erasure of existing data in the event of a software error or microcontroller malfunction. The memory chips were also designed to 'fail secure' so that if the power supply failed, data was not lost.

The process of reading from memory was much simpler as data vulnerability was much less. The memory chip was selected and three consecutive bytes were sent, the read instruction followed by two address bytes (most significant byte first). Data were then continuously read (the paged implementation did not complicate the reading process) until the memory chip was deselected. In the control software an interface to the four memory chips was created to hide the discrete nature of the memory and provide a contiguous memory model. The memory protection features, together with large capacity, small standby current (~5μA) and surface mount package design made these memory chips ideal for this application.

2.4.4 COMMUNICATION

Transfer of instructions to the device and retrieval of data by a computer was achieved using the microcontroller's SCI. This feature of the microcontroller implements the RS-232 communication protocol using 5V logic levels that were converted to the standard of 12V levels by a serial interface unit (section 2.4.5.1, circuit diagram given

in appendix C.7). A non-standard baud rate of 7812 baud was used for normal communication between the microcontroller and the computer as this rate was one of the highest achievable for the microcontroller when using a 2MHz crystal (section 2.4.5.3). In the special case of writing measurement sequence instructions to the microcontroller's internal EEPROM a communication rate of 300 baud was used to allow time for the longer write cycle of the EEPROM.

Serial port communication software (TEmu) was written that could communicate at the non-standard rate of 7812 baud as well as switch between the rates of 300 and 7812 baud. This software was designed for sending commands and receiving data, real time viewing of data and debugging purposes (section 3.1).

2.4.5 POWER CONSIDERATIONS

The device was used in two states, the configuration state (power supplied externally) and the measuring state (power supplied by on-board batteries). The power supply circuitry (figure 2.15) was designed to make a continuous transition from the external supply to the on-board supply as well as provide control over power supplied to regions of the device and feedback on the battery voltage status. These features were designed to extend battery life by minimising power consumption and provide a clean shutdown of the system in the event of a failing power supply.

2.4.5.1 Power supply sources

Initially, while the device was connected to a computer for configuration, the serial interface unit supplied power (figure 2.16). A 12V lead-acid battery supplied both the serial circuitry and the device, thus making the whole system portable. Once configured, a jumper on the device was used to connect two 3.7V lithium thionyl chloride batteries that supplied power when the device was isolated from the serial interface unit and taken into the field.

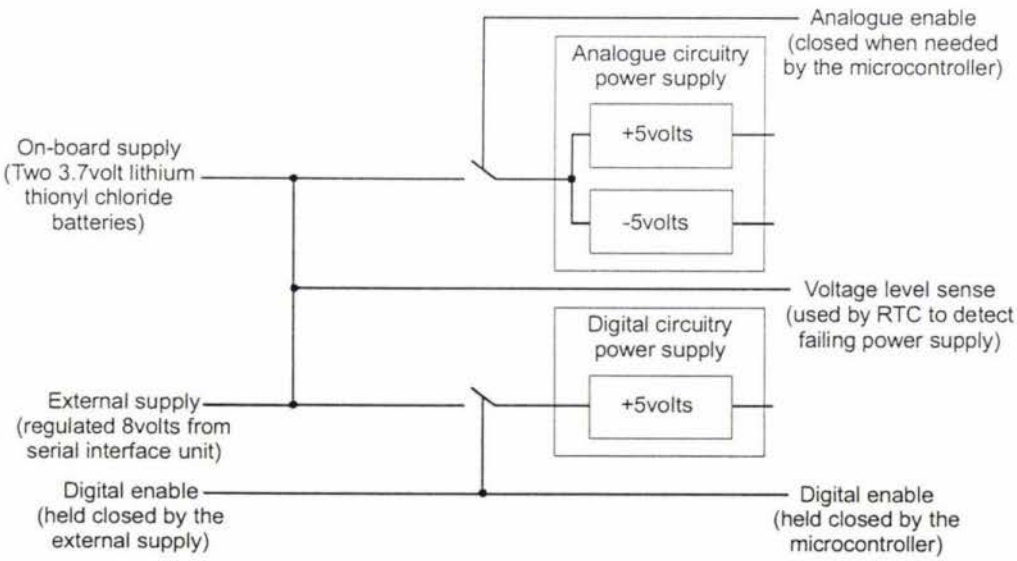


Figure 2.15: The power-supply schematic. Separating the power supplies into a digital and analogue supply had two main advantages: interference from switching digital components through the ground connection was minimised and the analogue region of the circuit could be shutdown as a means of conserving power. A complete circuit diagram of the power supply is given in appendix C.6.

2.4.5.2 Power considerations on the device

Supply lines that provided power to the digital circuits were generally noisy due to the switching of digital logic levels. For this reason the power supply was separated into two branches immediately after the battery connection. One branch provided power to the digital components using a single 5V regulator and the other provided 5V and –5V using a CMOS voltage converter (ICL7660CBA) with two 5V regulators. An added advantage of this was that the analogue supply could be controlled by the microcontroller allowing it to be shut down when the analogue circuitry was not required thereby reducing power consumption. This control switch is labelled analogue enable in figure 2.15 and was connected to port B of the microcontroller.

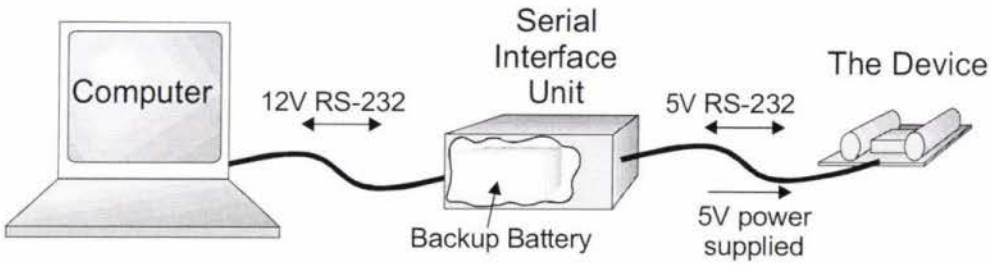


Figure 2.16: Connectivity of a computer, the serial interface unit and the device. During configuration the device was supplied power by the serial interface unit and then by its on-board batteries during operation.

2.4.5.3 Power saving

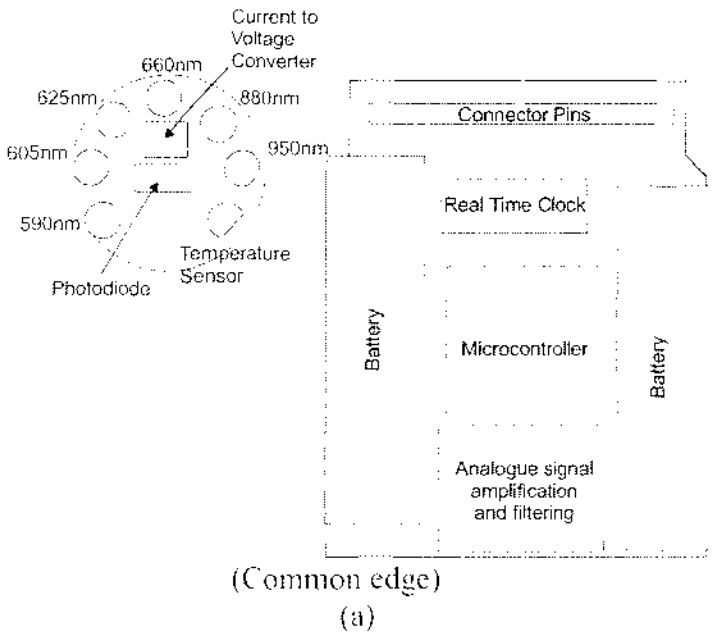
As with any battery-operated system, a goal was to maximise the useful operating time through minimising the load on the batteries. The most significant feature of the device to help with this was the microcontroller's stop instruction which, in co-ordination with the real time clock, reduced the microcontroller's load on the batteries to approximately 100μA. When executed, the stop instruction caused all of the microcontroller's internal clocks and main oscillator to freeze placing the processor in a minimum-power standby mode, a state in which it stayed until a signal on the non-maskable interrupt pin was received. While in standby mode, all I/O ports and internal registers retained their values and, as the microcontroller is a fully static device, after recovery from stop execution continued with the next CPU instruction.

The other components that were continuously connected to the supply were the ADC that had an auto shutdown feature that reduced its load to approximately 1nA, the memory chips that drew 5μA each in standby mode and the real time clock that ran continuously drawing approximately 25μA (all analogue circuitry was disconnected from the supply by the microcontroller's analogue enable pin).

Another means of reducing power consumption was by selection of the microcontroller crystal frequency. During development a crystal frequency of 8MHz was used but replaced in the final design by a 2MHz crystal thus reducing power consumption to a quarter of that used previously by the microcontroller. An effect of this frequency reduction was that the maximum sampling rate was reduced due to a limit imposed by the minimum number of CPU instructions that needed to be executed between sampling interrupts.

2.5 Construction

In accordance with the design specifications, minimisation of the size and weight of the device was achieved using double-sided printed circuit boards (PCBs) and surface mount components. Important aspects of the layout for the main control unit and sensor head were separation of noisy digital tracks from analogue signal tracks, separate ground tracks for the analogue and digital components to further reduce electrical interference and minimising necessary PCB area. Figure 2.17 is a block diagram of the functional regions in the final layout (for complete PCB layouts see appendix D) and shown in figure 2.18 are photographs of the completed device.



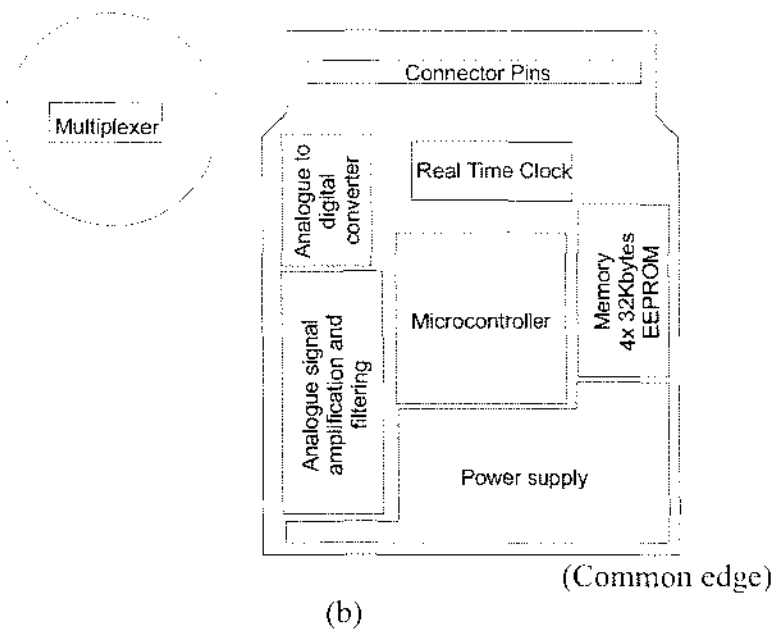
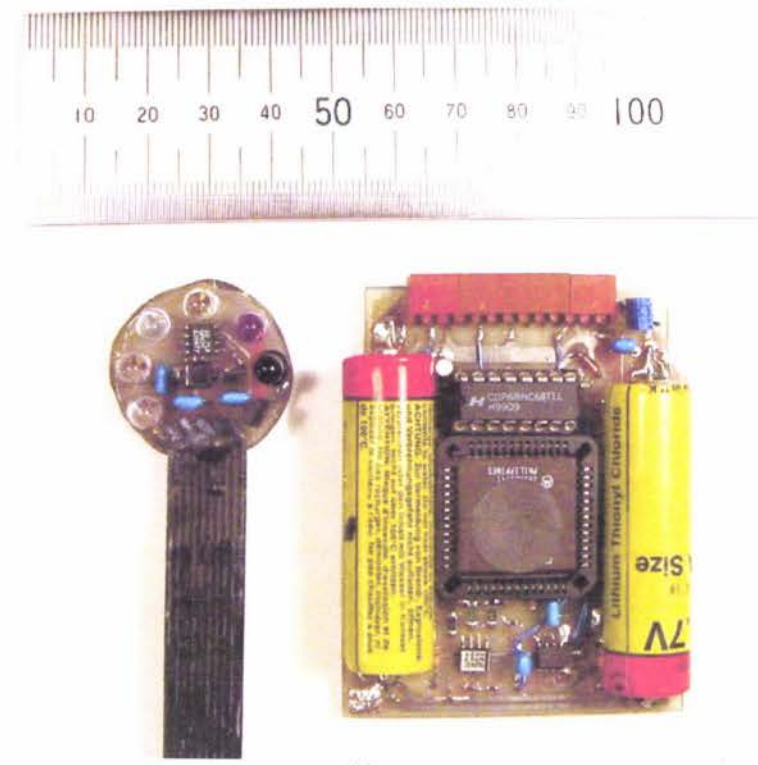
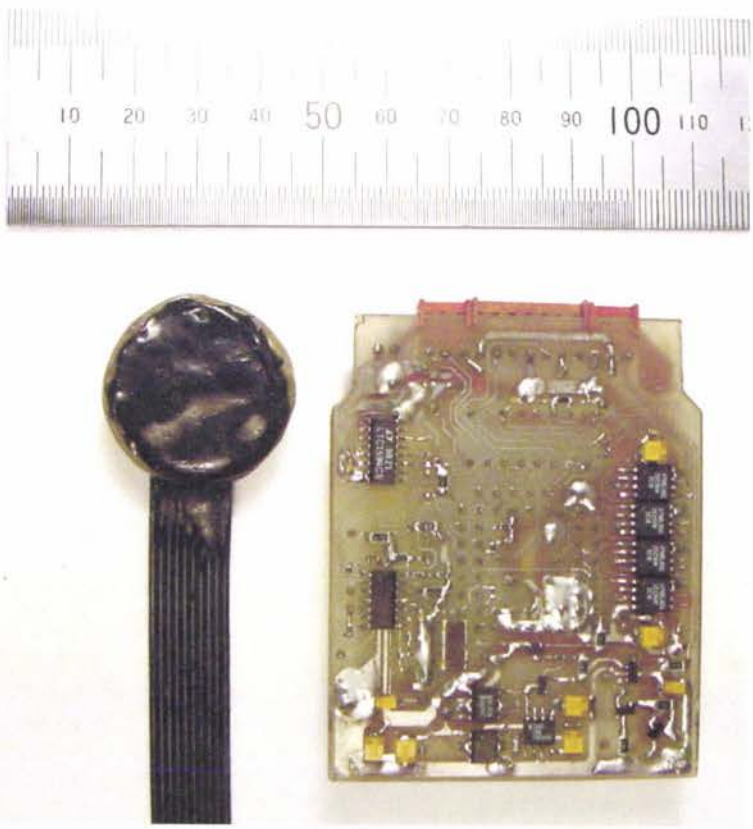


Figure 2.17: The functional regions of the PCB layout for the control unit (a) from above and (b) from below.



(a)



(b)

Figure 2.18: Photographs of the completed device from (a) above and (b) below.

Chapter 3

Control Software and Algorithms

In order to control the microcontroller, assembly language software was written that provided a user interface and a script language interpreter, controlled lower level hardware and performed data processing. The assembly language used was the Motorola M68HC11 instruction set [19][20].

The interactions between user files, software and hardware are shown in figure 3.1. Script files were processed by a conversion program turning them into valid assembly language that was then assembled and transferred to the device via the terminal emulator program. At the completion of a measurement sequence, data were retrieved from the device in hexadecimal format, and converted into a more usable format using another conversion program.

The operating system source code was assembled and then loaded into the EPROM of the microcontroller using the Motorola PCBug11 software and a development programming board. After the microcontroller was programmed it was removed from the programming board and inserted into the device.

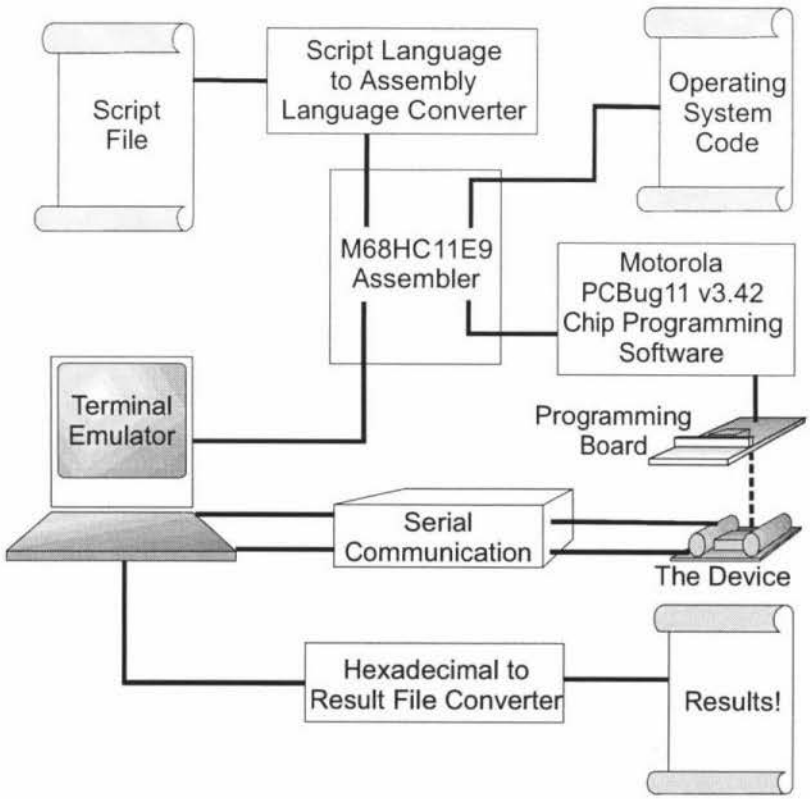


Figure 3.1: System overview. The dotted line indicates physical movement of the microcontroller between the programming board and the device.

3.1 The Logical Model of the Device

The device had two regions of memory and a process that executed commands from one region (the measurement sequence memory) to generate results that were stored in the second memory region (the result memory). A list of measurements were created (using the scripting language described in section 3.3), compiled and loaded into the measurement sequence memory. Sequence execution was initiated using the run command and, at the completion of the measurement sequence, the device was reset and the acquired data downloaded from the result memory through the serial port.

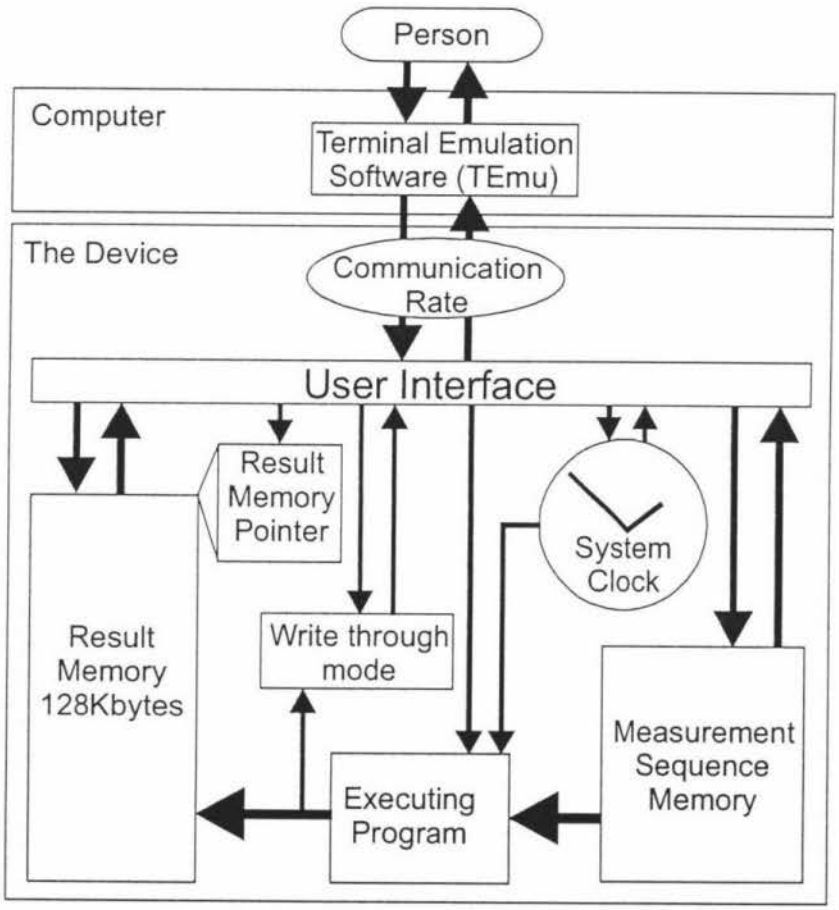


Figure 3.2: The logical model of the device. Arrow weighting indicates the relative amounts of data transfer.

The logical model of the device showing the various elements with which the user interacts is given in figure 3.2. A serial port terminal emulation program (TEmu) was written specifically for communication with the device. It used a first in, first out cyclic buffer to send and receive data to and from the computer's serial port [24]. A window within the TEmu program displayed text sent from the device and was used to input commands. Initially, a help screen and command prompt sent by the device, was displayed in the TEmu communication window. Commands were then entered that performed the instructions given in table 3.1.

The commands issued to the device using TEmu were sent, via the serial port, to the microcontroller operating system. Typically, a measurement sequence was first uploaded to the device where it could be edited or displayed using the measurement sequence memory operations (table 3.1). Device-dependent variables such as the system clock, result memory pointer and operating mode were configured and then execution of the measurement sequence was initiated. At this stage, TEmu was used to interpret and plot data sent back during real-time operation or once the measurement sequence had completed, TEmu was used to download the results and save them to file.

Table 3.1: Operating system commands. The assembly code for each instruction is listed in the appendix indicated.

Instruction Class	Instructions	Keystroke	Source Code Reference
Measurement Sequence Memory Operations	Load a measurement sequence	L	E.6.3
	Erase a measurement sequence	E	E.6.12
	Output the contents of measurement sequence memory	O	E.6.4
	Modify single consecutive bytes of measurement sequence memory	M	E.6.6
	Fill measurement sequence memory with a specific value	F	E.6.7
	Run a measurement sequence	G	E.6.1
Result Memory Operations	Output the contents of result memory	D	E.6.5
	Fill result memory with a specific value	I	E.6.8
	Set the result memory pointer	P	E.15.13
System Clock Operations	Set the system clock	S	E.6.10
	Display the current system time	T	E.6.9
Serial Communication	Toggle communication baud rate between 7812 and 300	B	E.6.11
Data Collection	Toggle write through mode	W	E.1
Debugging Operations	Manually set the system status register (debug mode)	U	E.6.14
	Run from EEPROM (debug mode)	R	E.6.2
Help	Display the help screen and system status	H	E.1

In addition to the simple use of the system described above, there were various functions that were required by the hardware or added flexibility to the device. These included functions to display data as it was collected, set and display the system clock, change the communication rate, display the help screen and provide debugging facilities. A complete source code listing of the operating system commands are given in appendix E.6

3.1.1 Setting write-through mode and the result memory pointer

As well as storing data to memory, the device could display measurement results in real time. This was called write-through mode as data to be stored in result memory was also written to the serial port. Using TEmu, these data could be plotted to generate graphs for real-time use or to help with debugging. The displayed data were still written into result memory so they could be downloaded at a later stage. Write-through mode was set or cleared using the 'toggle write-through mode' command (appendix E.1) and its status was viewed using the help command.

The result memory pointer allowed data to be written starting from any location in memory. For reasons described in section 3.2.2.2 the result memory pointer consisted of a segment value that ranged from 1 to 4 and an offset value ranging from 0000 to 7FFF. The value of the pointer was assigned using the set command (appendix E.15.13) and displayed using the help command.

3.1.2 Setting the system clock

The system clock could be set by or displayed for the user. It was also called upon by the executing program to facilitate the timing of measurements, delay between measurements and power saving features (section 2.4.5). After the device had been reset, the system clock had to be initialised to the correct time (appendix E.6.10).

3.1.3 Communication rate

Normal communication between the device and the terminal emulator program was at 7812 baud (section 2.4.4). However, as measurement sequences were loaded into the microcontroller's EEPROM, a slower rate of 300 baud was required to allow for the slower programming rate. The toggle baud rate command (appendix E.6.11) modified a specific register in the microcontroller to change the communication rate between 7812 and 300-baud.

3.1.4 Help Command

The help command was used to display the current system status and to list the user commands. System status consisted of the value of the result memory pointer and the individual bits of the system status register (table 3.2). The help command displayed the status information, waited for a key to be pressed and then displayed the user commands as in the help screen displayed on start up.

3.1.5 Debug mode

During the development of the device, it was necessary to test and debug various functions of the operating system. These debugging facilities will be useful for further development of the device and so remain a part of the available commands.

Modification of a program in EPROM involved erasing the existing program through exposure to UV light (recommended exposure time of approximately 40 minutes (Dykstra, R., personal communication) and electronically reprogramming the EPROM using the programming board (figure 3.1). To avoid this procedure the microcontroller's EEPROM was utilised which could be erased and reprogrammed using the internal hardware of the microcontroller (reprogramming time of approximately 1 minute). The functions to be tested were loaded into the EEPROM and either executed directly using the run from EEPROM command (table 3.1) or referenced from within the main program via branch vectors. The use of branch vectors, which were also used in programming the interrupt routines, allowed the test routines to be modified without affecting the absolute addresses within the operating system code. Branch vector use (figure 3.3) involved a function call that caused the program counter to jump to a fixed address in the last bytes of EEPROM where there was a jump instruction that pointed to the start of the function body. This jump instruction was called the branch vector and the value to which it pointed could be changed without having to change the jump address in the main program.

Table 3.2: The association of bits in the system status register.

Bits	Description
0	The escape character has been entered by the user to abort the current command
1	The current baud rate is 300 (set) or 7812 (clear)
2	Echo characters received through the serial port
3	Not Used
4	Write through mode on (set) or off (clear)
5	Not Used
6	Line feed character sent with carriage return
7	The crystal frequency used is 2MHz (set) or 8MHz (clear). For debugging using the Motorola programming board (8MHz).

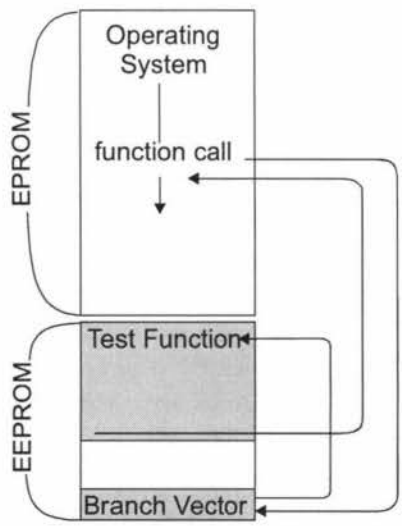


Figure 3.3: Branch vector use in operating system debugging.

3.2 The Microcontroller Operating System

The main functions of the operating system were to allow communication between the device and a computer, to interpret and execute measurements found in the measurement sequence memory and to manage low level interaction with the hardware. There were four module classes (figure 3.4): control, measurement sequence instructions, system data and hardware. Of these, the highest level were the control modules that called upon the hardware modules to perform the functions of the operating system. The sequence execution module called upon the measurement instruction modules to execute each instruction as they were read from measurement sequence memory.

3.2.1 CONTROL MODULES

3.2.1.1 Operating System

This module linked the other components into a single program. It began by initialising and configuring the microcontroller mode, operating system variables, system interrupts, serial communication and the system clock. It then entered a simple two-state loop in which commands were fetched and then executed. The commands, received from TEmu via the serial port, were interpreted within the main loop and then transferred to the appropriate function within the command module. Global data, interrupt vectors and variable definitions were linked into this module making them available to the rest of the program.

3.2.1.2 System Initialisation

This module was the first to be run after a device reset. A register was set in the microcontroller defining the RAM and register memory regions within the address space. It then determined which features of the microcontroller were active, set the priority of interrupts and the phase and polarity of the clock used for communication between the other connected devices. After the microcontroller-dependent initialisation had finished, the operating system variables were set to their defaults.

3.2.1.3 Command

Contained in this module were the functions that performed the commands provided by the operating system module (table 3.1). When a command was received, the operating system module invoked one of the command module functions to perform the desired task and return control back to the operating system module. These command routines were constructed using the functions provided by the hardware modules (grey, round ended boxes in figure 3.4).

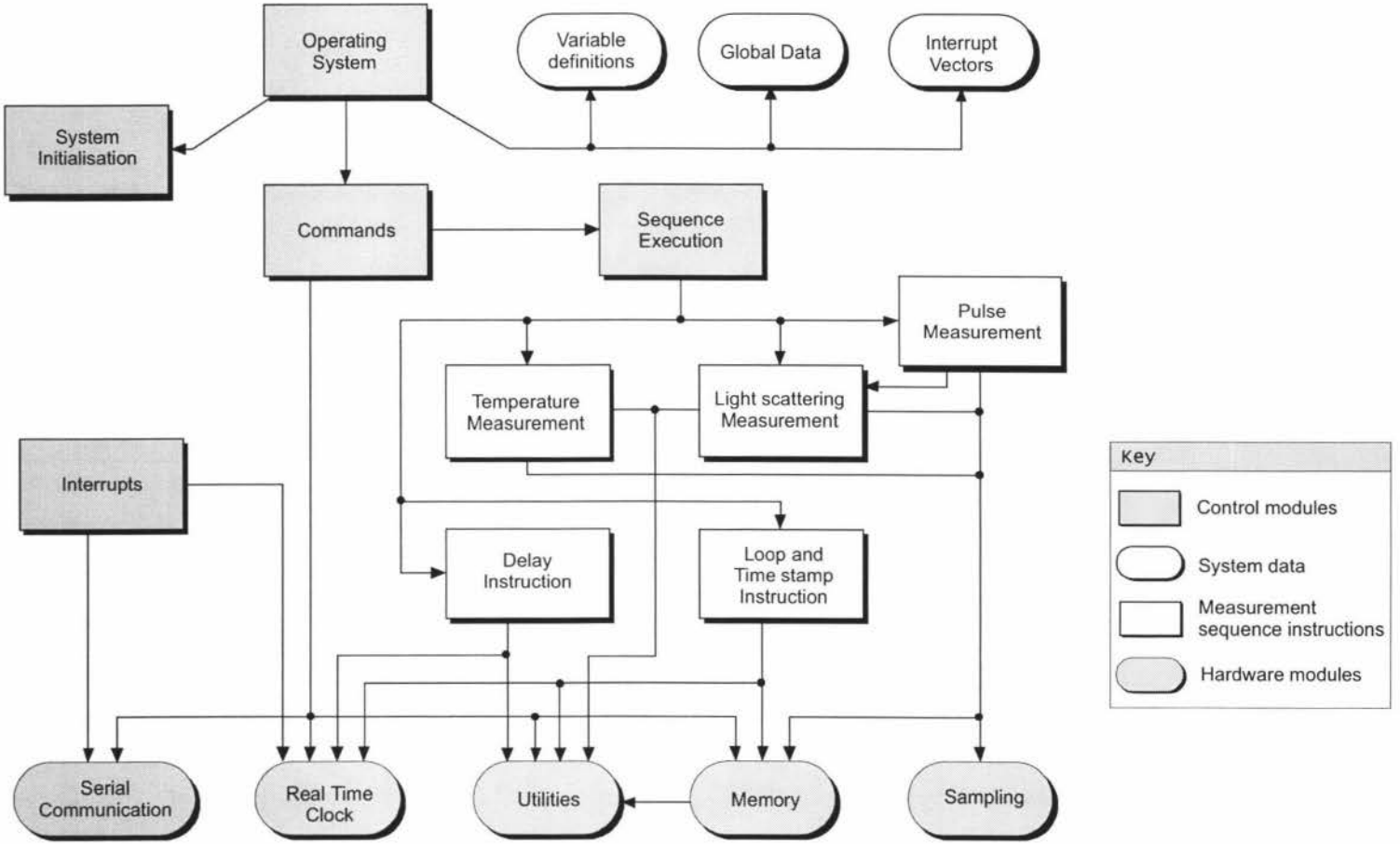


Figure 3.4: Hierarchical arrangement of modules within the microcontroller operating system. Generally, the height of each module in this structure diagram indicates the level of abstraction from the device hardware. The arrows indicate the modules called upon by the higher level modules.

3.2.1.4 Interrupts

The interrupt driven processes included the modulation of the LEDs, sampling and the interaction with the real time clock (section 2.4.2.3). The interrupt service routines associated with these three processes were located in this module.

Steady modulation of a particular LED was achieved using the output compare function of the microcontroller (section 2.4.2.1) and a global variable containing the LED to be modulated. Within the service routine associated with the output compare function, the next interrupt was set to occur half the modulation period later and the state of LED was switched based on its current state. When the next output compare interrupt occurred, this process repeated and the LED was modulated at the modulation frequency. As this process occurred independently of the main program, control of an LED for the light scattering measurements was reduced to selecting a particular LED and activating the output compare interrupt.

The sampling interrupt routine worked in a similar fashion but, rather than toggling an LED, a sample was acquired from analogue to digital converter. The sampled value was stored in a global variable along with the number of samples acquired (the sample count was incremented by one every time the sampling interrupt occurred). To the rest of the program these registers contained a sample value and a count variable that increased every time the sample value changed. A function that read these values could calculate the time elapsed between two measurements by knowing the sampling frequency and the number of samples acquired.

The third interrupt routine services the non-maskable interrupt, which was connected to the interrupt line of the real time clock (section 2.4.2.3, section 3.2.2.3). The main purpose of this interrupt was to awaken the microcontroller from power saving mode (section 2.4.5.3) at the completion of a delay period. When an interrupt from the real time clock was generated, an internal byte was set that held the status of the clock and the reason for the interrupt. The interrupt service routine read this byte and stored it in a global variable for the operating system to interpret if necessary. When the real time clock was used simply as an alarm the interrupt service routine was not called.

3.2.1.5 Sequence Execution

The sequence execution module was essentially a program that ran using the facilities provided by the operating system. When initiated with the run command, the sequence execution module read instructions located in the measurement sequence memory, interpreted any associated parameters and then called one of the measurement sequence instruction modules to perform the measurement. The program worked in a similar way to the main operating system loop except that its input commands came from the interpreted measurement instruction codes (table 3.6, section 3.3) retrieved from the measurement sequence memory.

3.2.2 HARDWARE MODULES

3.2.2.1 Serial Communication

This utility module provided the functionality (table 3.3) necessary for communication with TEmu utilising the microcontroller's on-chip, asynchronous serial communications interface (SCI). Included were functions to initialise the SCI (set the baud rate, number of data bits and necessary interrupts (section 2.4.4)), read and write in various formats (character, byte, word, binary byte and a null terminated string), and timing dependent communication functions.

Table 3.3: Functions provided for serial communication

Function name	Description	Appendix reference
Serialinit	Initialisation of the serial port	E.17.1
Inbin	Read a binary number	E.17.4
Inhex	Read a hexadecimal number	E.17.3
Inword	Read a double hexadecimal number	E.17.2
Inchar	Read a character	E.17.5
Outword	Write a double hexadecimal number	E.17.6
Outhex	Write a hexadecimal number	E.17.7
Outbin	Write a binary number	E.17.8
Outstr	Write a string	E.17.9
Outchar	Write a character	E.17.10
SCIwaitw	Wait for the transmit buffer to empty	E.17.11
SCIwaitr	Wait for the end of a stream of input	E.17.12
Return	Write a carriage return	E.17.13
Space	Write a white space	E.17.14

3.2.2.2 Memory

The routines of this module interacted with the 128 Kbytes of serial EEPROM. As described in section 2.4.3, the serial memory consisted of four separate memory chips. The purpose of the memory module was to make this appear as one contiguous memory block referenced by a segment and offset. The interface provided for memory access involved activating the memory at the location pointed to by the result memory pointer (MemWon, MemRon), reading and writing using the memory access commands (SaveB, ReadB, SaveW or ReadW (table 3.4)) and then deactivating memory (MemWoff, MemRoff).

The 64 byte paging technique used by the memory chips (section 2.4.3) was handled automatically by these save and read routines. For every byte read or written the result memory pointer was incremented by one and, as the end of a memory chip was reached, roll over to the next memory chip is handled automatically.

Table 3.4: Functions provided for the use of the result memory.

Function name	Description	Appendix reference
MemWon	Turn on memory ready for writing of data at the location pointed to by the result memory pointer.	E.15.1
SaveB	Write a byte to serial memory	E.15.2
SaveW	Write a word to serial memory	E.15.3
MemIncW	Increment the result memory pointer allowing for switching between memory chips	E.15.5
MemWoff	Turn off memory	E.15.4
MemRon	Turn on memory ready for the reading of data at the location pointed to by the result memory pointer	E.15.6
ReadB	Read a byte from serial memory	E.15.7
ReadW	Read a word from serial memory	E.15.8
MemIncR	Increment the result memory pointer allowing for switching between memory chips	E.15.10
MemRoff	Turn off memory	E.15.9
MemRdy	Wait for the completion of a memory write	E.15.11
MemStatus	Return the status register value of a memory chip	E.15.12
GetMem	Ask the user for a result memory pointer value	E.15.13
MemDisp	Display the current result memory pointer in the format memory segment : memory offset	E.15.14

3.2.2.3 Real Time Clock

The real time clock is a serial peripheral device with registers that held the current time and the alarm time. The operating system functions used to initialise the clock to the correct time modified these registers (table 3.1). The major use of the real time clock was for the delay instruction that used the alarm feature to switch the microcontroller out of power saving mode (section 2.4.5.3, section 3.2.1.4). The interrupt line of the real time clock was connected to the non-maskable interrupt line of the microcontroller and was triggered when an alarm occurred. To activate an alarm the appropriate registers were set to a future time and the alarm interrupt was unmasked.

The real time clock could also generate periodic interrupts ranging from once a day to every 0.488ms. This feature was used in a different delay method in which the clock was set to interrupt every 0.977ms and the resulting interrupts counted until the desired delay had elapsed. The other functions within this module facilitated writing to or reading from the real time clock registers and activating or deactivating the alarm or periodic interrupt.

3.2.2.4 Sampling

Interaction with the analogue to digital converter (ADC) and the process of acquiring a sample was simplified by the functions contained in this module. Similar to the interface provided for memory use, acquiring a sample involved activating the sampling process (Sampleon or ADCon), reading the sample value from the global variable (updated by the sampling output compare interrupt (section 3.2.1.4)) and then deactivating the sampling process (Sampleoff or ADCoff) (appendix E.14). As the ADC was an SPI device, it had to be deactivated before a write to memory was attempted (section 2.4). This introduced two scenarios for use of the ADC, one was the complete initialisation of the sampling process in which the sample count was reset, the analogue circuitry was enabled, serial clock phase was set, the ADC was activated and the sampling output compare interrupt was unmasked. The other was when the ADC had to be deselected in order to write to memory but the sample count value had to be retained and the analogue circuitry should remain enabled to avoid the start-up stabilisation time. The two pairs of functions designed for this, Sampleon/Sampleoff and ADCon/ADCoff, were used for each case respectively. A typical use involved calling Sampleon to begin a series of measurements. Acquired data were stored to the result memory during this period by deselected the ADC using ADCoff, writing the data to memory and reactivating the ADC using ADCon. At the end of the measurement period Sampleoff was called to disable the analogue power supply and shutdown the ADC.

3.2.2.5 Utilities

This module included routines that were not specific to any of the other module types (table 3.5). The first two of these, Anenable and Andisable, were used to control power supplied to the analogue circuitry. They were called by the sampling routines so that the analogue circuitry was only active while a sample was collected. The next group of routines, Addtime, Wordmul, Roldw and Cmptime were all functions called by the pulse-measuring module (section 3.4). They performed simple arithmetic operations on 16-bit (word) and 32-bit (double word) variables that were not included in the M68HC11 instruction set. Since sampling was done using a 12-bit ADC (section 2.4.1), all processing of the acquired data used 16-bit operations. Addtime added a word value to the accumulative double word time variable. Wordmul multiplied two words together leaving a double word. Roldw rotated the bits of a double word left by a specified number and Cmptime compared a specified number of multiples of a word value against the accumulative time variable returning a greater than, less than or equal to condition. Cyclic buffers were used extensively by the pulse-measuring module and so the Cycinc routine was written that incremented a memory pointer and automatically reset it back to zero when the maximum size of the buffer was reached.

Jumpback was a debugging function designed to be used at the end of a series of test functions in the microcontroller EEPROM. It worked by pushing values onto the system stack and calling a 'return from interrupt' instruction which, in turn, pulled these values from the stack to resume program execution. The values were chosen so that program execution would start at the beginning and so the function acted like a software reset.

The sleep function was called to switch the microcontroller into power saving mode (section 2.4.5.3) after all serial communication had finished. Once in the power saving state the microcontroller could only be reactivated by a non-maskable interrupt generated by the real time clock. This routine was called from the delay instruction (section 3.3) and was quite different from the other delay routines in this module. The first of these, Delay (appendix E.18.4), used the periodic interrupt of the real time clock to generate interrupts that were approximately one millisecond apart which it then counted until the desired delay period had passed. This function could pause for periods of 1ms to 64s. Bloop was a simple 'busy wait' routine used for relatively short pauses of 10µs to 1.8ms and the Waitkey routine waited for a character from the serial port. The last delay function in this module is the Dly10 routine that was used to put a fixed period between successive writes to EEPROM. This function was called by the memory module (appendix E.15) when saving data to result memory and by the write routine (appendix E.18.15) when saving bytes to the measurement sequence memory.

Table 3.5: General utility functions.

Function name	Description	Appendix reference
Anenable	Enable the analogue circuitry	E.18.1
Andisable	Disable the analogue circuitry	E.18.1
Addtime	Add a time interval to the global variable time	E.18.8
Cmptime	Compare multiples of a time interval with the global time variable	E.18.9
Wordmul	Multiply two words together resulting in a double word	E.18.11
Roldw	Roll a double word left	E.18.12
Cycinc	Increment a word pointer cyclically.	E.18.10
Jumpback	Software reset	E.18.2
Sleep	Put the microcontroller in power saving mode	E.18.13
Delay	Delay using the periodic real time clock interrupt	E.18.4
Bloop	Busy loop delay	E.18.5
Waitkey	Wait for a character from the serial port	E.18.17
Dly10	Delay for 10 milliseconds	E.18.6
Write	Write a byte to RAM or EEPROM	E.18.15
Qkgett	Display current time without pausing for a key stroke	E.18.3
Upcase	Convert a character to upper case	E.18.16
Error	Display an error message	E.18.18

The remaining functions of this module are Qkgett, Upcase and Error. Qkgett was called before entering a measurement sequence to display the time that the measurement sequence began. Usually, it acted to remind the user that the system clock had not been initialised. Upcase converted characters to upper case, and Error displayed a particular error message based on the error code it was passed.

3.2.3 MEASUREMENT SEQUENCE INSTRUCTIONS

The measurement instruction modules included pulse measurement, light scattering measurement, temperature measurement, delay, loop and timestamp. The first of these, pulse measurement, is described in greater detail in section 3.4. The next two, light scattering and temperature measurement, used similar algorithms which, for the light scattering case, is given in figure 3.5. The routine began by calling the Sampleon function (section 3.2.2.4) to activate the analogue circuitry and enable periodic sampling. It then immediately checked whether enough measurements had been made which allowed for the boundary case of zero measurements. For every measurement there was the option to average over a series of samples and once the average value had been found the result was saved to result memory. When enough measurements had been made the routine called the Sampleoff function that deactivated the analogue circuitry and deselected the ADC. The only difference between the light scattering measurement and the temperature measurement was that the temperature sensor was selected instead of the photodiode.

The delay instruction was responsible for shutting down the microcontroller between measurements and minimising power consumption thereby extending the overall operating time of the device. The routine began by taking the desired delay, adding it to the current time and saving the result in the alarm registers of the real time clock. The alarm interrupt mask was then removed and the sleep function called, placing the microcontroller in power saving mode (section 2.4.5.3). When the real time clock reached the alarm time it generated an interrupt that reactivated the microcontroller ready for the next instruction.

Loops came in two forms, infinite and finite. In measurement sequence memory these were stored as an instruction code, a branch offset and the number of loops remaining. If the value containing the number of remaining loops was zero it was interpreted as an infinite loop and the sequence counter was moved back by the amount stored in the branch offset. Otherwise, if the number of remaining loops was non zero the value was reduced by one and the sequence counter was again moved back by the branch offset. When there was only a single loop remaining the loop count was set to negative one (FFFF), if this value was encountered the loop was ignored and the sequence continued with the next instruction. Therefore the maximum number of finite loops was 65534 (FFFE). When the remaining loop count was changed its value was stored back into measurement sequence memory. Consequently, executing a sequence with finite loops more than once required the number of remaining loops to be modified each time either by editing the individual bytes (modify byte command; table 3.1) or through reloading the measurement sequence.

The time of each measurement occurrence could be calculated by summing the delays within a measurement sequence and then adding this to the starting time of the experiment. However, for long measurement sequences, this would become increasingly inaccurate. The timestamp function overcame this by obtaining the time from the real time clock and saving it in result memory. This function was frequently used as the first instruction of the measurement sequence loop so that the acquired data could be correlated with behavioural patterns or other observations.

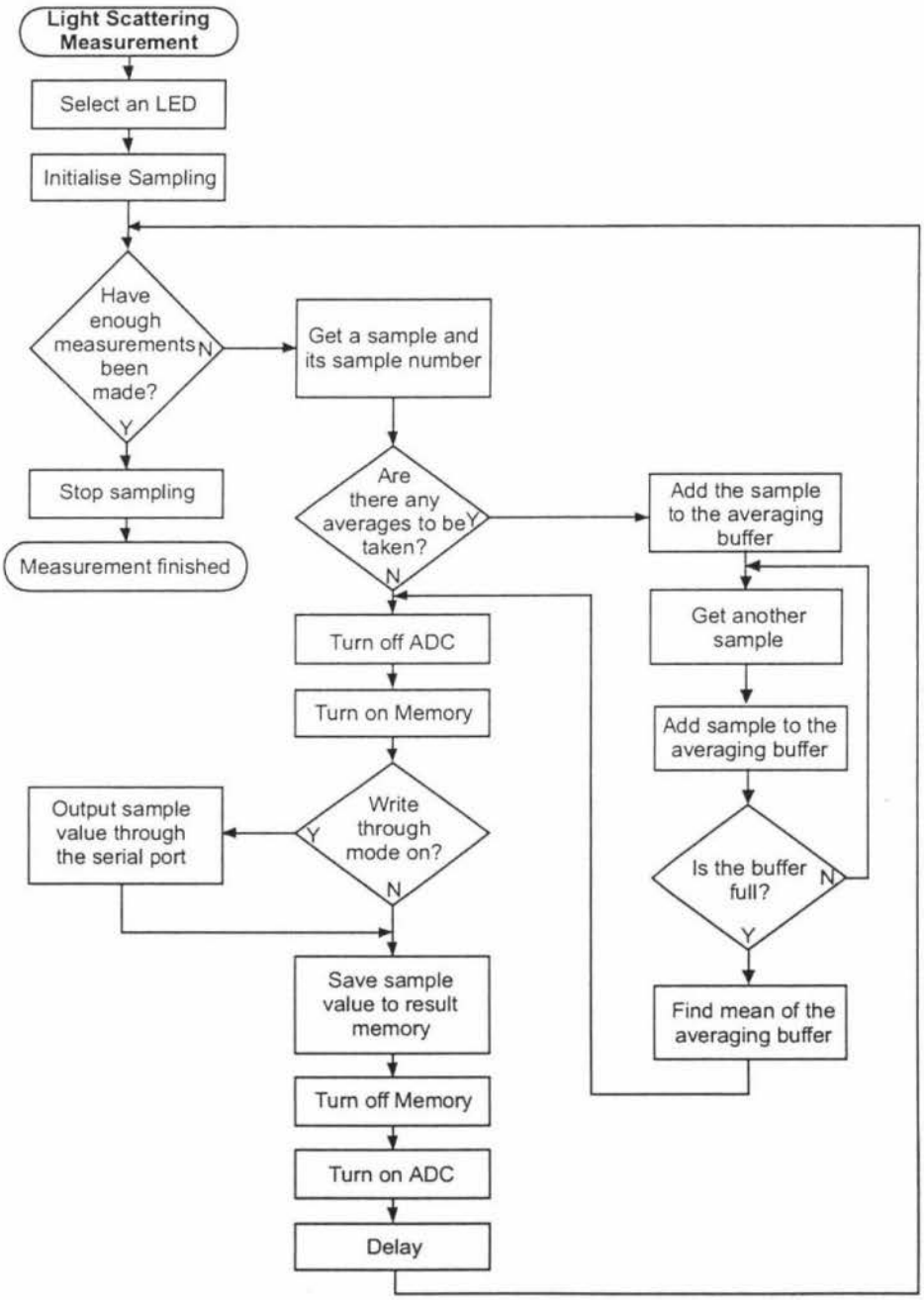


Figure 3.5: The algorithm used to make light scattering measurements.

3.3 Measurement Scripting Language

Flexibility was built into the device using a scripting language (table 3.6). A text file containing the desired sequence of measurements was created, converted into assembly language, compiled and uploaded to the measurement sequence memory of the device. In memory, each instruction was stored as an instruction code (table 3.6) followed by a series of parameters. The sequence execution module (section 3.2.1.5) interpreted the instruction code and read the number of subsequent bytes required for that particular instruction. The module then set variables inside the operating system to the interpreted values and called the appropriate measurement instruction routine.

Table 3.6: The measurement sequence script language instructions. Further details for each instruction are given in table 3.7

Instruction type	Instruction format	Instruction code	Size /bytes
Measurement instructions (write to result memory)	PULSE Mbuf Pbuf thres to	10	6
	LED1..6 hm d avr	90.. E0	6
	TEMP hm d avr	20	6
	TIME timemask	60	2
Sequence instructions	DELAY hours mins secs	30	4
	LOOP label [number]	40	5
	END	00	1
Hardware configuration instructions	MEM segment offset	50	4
	SAMFREQ frequency	70	3
	LEDFREQ frequency	80	3
	PORT port-number	F0	2
Debugging instructions	WAITKEY	01	1
	BREAKENABLE	02	1
Compiler directive	ORG address	N/A	N/A

The script language instructions were divided into four distinct groups, measurement, sequence, hardware configuration and debugging instructions (table 3.6). The first of these groups, the measurement instructions, performed actual measurements, collected data and stored results into result memory. The format and number of data stored by these instructions depended on the parameters that succeed each instruction code (table 3.7). The sequence instructions related only to the script and affected the order or timing of the other instructions. Features of the device were selected using the hardware configuration instructions and the debugging instructions allowed step by step analysis of each script language instruction during development. The final script

instruction, ORG, was used to indicate the location of this script file in measurement sequence memory and allowed for the possibility of multiple script files to be loaded.

The script language instructions allowed for a large variety of measurement sequences. Specialised measurements for real-time collection of pulse, temperature, blood or tissue oxygenation data could be created using the infinite loop instruction and reducing all delays to zero. General long-term experiments that measured these physiological variables over several days were possible by minimising power consumption using the delay feature. As an example, table 3.8 is a listing of a script file that was used during field testing of the device (section 4.2). The important aspects of this script begin with the ORG instruction that tells the compiler to load the script into measurement sequence memory at address B600 (hex). A series of hardware initialisation instructions were then executed before the sequence entered the start-up loop. This finite loop was used to indicate that the device was operating correctly after detachment from the computer by switching between two LEDs. At the completion of the start-up loop the device was put into power saving mode for twenty minutes to allow time for attachment of the device to a penguin. Once the delay period had ended the result memory pointer was reset to the first byte in memory and the main experiment loop began. Each cycle of the main loop placed a time stamp in memory followed by a pulse measurement, the six light scattering measurements and a temperature measurement. The port instruction was used to select different levels of amplification for the pulse measurement and the shorter wavelength LEDs (section 2.3.4).

Table 3.7: The script language instructions with an explanation of their parameters.

PULSE	Make a real time pulse measurement (section 3.4).
Mbuf	The median buffer size. This value controlled the length of the input waveform used to calculate the median. Range 1..64 (hex)
Pbuf	The period averaging buffer size. This value controlled the confidence in the period estimate by changing the number of periods averaged. Larger values reduce the likelihood of obtaining a measurement. Range 1,2,4,8
Thres	The maximum allowable deviation between the average period and values in the period buffer. Range 1..FF (hex)
to	Time Out. The number of sample periods between the use of the median, upper and lower quartile in calculation of the square wave. Range 1..FFFF(hex)
Example	PULSE 64 05 0A 0300
LED1..6	Make a light scattering measurement
TEMP	Make a temperature measurement
hm	'How many' samples to take. This corresponded to the number of values stored in memory. Range 1..FFFF (hex)
d	Delay, in units of 0.977 ms, between each sample. Range 0..FFFF (hex)
avr	Number of averages taken for each sample. Range 0..FF (hex)
Example	LED4 0FA5 1000 10, TEMP 0800 0010 08

Table 3.7: Continued.

TIME	Put a time stamp in result memory
Timemask	A byte (hex) containing a mask indicating which time values to store The bit ordering was 0,0,yr,mm,dd,hr,min,sec
Example	TIME 07
DELAY	Put the microcontroller in power saving mode for a specified time.
Hours	Hours of power saving mode
Minutes	Minutes of power saving mode
Seconds	Seconds of power saving mode
Example	DELAY 12 34 56
LOOP	Repeat a section of the sequence an infinite or specified number of times
label	Branch location. Label had to occur before the loop command.
Number	(Optional) Specified the number of times a loop was repeated. If omitted the loop was assumed to be infinite. Range 0..FFFF (hex)
Example	LOOP Start 0025
MEM	Move the result memory pointer
Segment	The memory bank to write to. Range 01..04
Offset	The offset into the memory bank. Range 0000..7FFF (hex)
Example	MEM 01 0000
SAMFREQ	Set the sampling frequency
freq	The sampling frequency. Range 1..100Hz
Example	SAMFREQ 100
LEDFREQ	Set the LED modulation frequency
freq	The modulation frequency. Range 1..950Hz
Example	LEDFREQ 950
PORT	Select the sampling port (section 2.3.4)
Portnum	The port number. Range 01..04. 01 = Direct, 02 = Level shifted and amplified, 03 = 10 times amplification, 04 = temperature sensor
Example	PORT 02
WAITKEY	Halt sequence execution until a key is pressed
BREAK-ENABLE	Allow serial communication to interrupt a running sequence
ORG	The location of a measurement sequence in memory.
Address	First address location. Range B600..B7FF (hex)
Example	ORG B600
END	End of the measurement sequence

Table 3.8: As an example, the script file used for field testing of the device

ORG	B600	; Set the program origin in measurement sequenc
		; memory
LEDFREQ	950	; Set the LED modulation frequency to 950Hz
SAMFREQ	100	; Set the sampling frequency to 100Hz
MEMORY	01 0000	; Set the result memory pointer to the start o
		; memory
STARTUP		;
LED3	0001 0000 10	; Flash LED3 and LED6 37 (25 hex)
LED6	0001 0000 10	; Times to signal that the device
DELAY	00 00 01	; Is functioning properly
LOOP	STARTUP 0025	;
DELAY	00 20 00	; Wait 20 mins before starting main loop
MEMORY	01 0000	; Set the result memory pointer back to start
MAIN		
TIME	07	; Put a timestamp (hr:min:sec) into result memory
PORT	02	; Select port two for a pulse measurement
PULSE	64 04 10 0400	; Make a pulse measurement
		; Median buffer length = 100
		; Period buffer length = 4
		; Error threshold = 16
		; Number of samples between using the median,
		; upper or lower quartile = 1024
PORT	01	; Select port one for regular LED measurements
LED1	0008 0000 10	; 8 measurements using the first LED
LED2	0008 0000 10	; 8 measurements using the second LED
LED3	0008 0000 10	; 8 measurements using the third LED
LED4	0008 0000 10	; 8 measurements using the fourth LED
PORT	03	; Select port three for higher signal gain
LED5	0008 0000 10	; 8 measurements using the fifth LED
LED6	0008 0000 10	; 8 measurements using the sixth LED
TEMPERATURE	0008 0000 10	; 8 temperature measurements
LOOP	MAIN	; Continue the main loop ad infinitum

3.4 Pulse Rate Calculation Algorithm

To measure a pulse rate the device was configured to drive one LED and observe the resulting waveform. If a pulse signal was observed the waveform was periodic and an algorithm was needed to estimate the period. The simplest approach to the problem would be to store the entire waveform in memory, plot it at a later time and measure the period by hand. The memory limitations of the device precluded this approach and so a routine was required that could calculate the periodicity of the input signal and store only the result in memory. The algorithm implemented looked at the input signal in real time and used a level crossing technique to determine the period.

3.4.1 THE ALGORITHM

A diagram summarising the steps of the algorithm is given in figure 3.6. Initially, all variables and buffers were set to zero and the main loop began by reading and entering data into the median buffer. Data were added cyclically to this buffer so that a history of the waveform as long as the length of the buffer was maintained. At the same time, the number of samples taken so far (the sample-count value) was recorded for use later in the algorithm to determine the time elapsed during a complete period. A rough measure of the algorithms running time was maintained by summing the differences between these sample-count values. The median buffer was then scanned to calculate the maximum, minimum, median, upper quartile and lower quartile. The length of the buffer had to be greater than a single period of the waveform so that the true minimum and maximum were calculated (figure 3.7).

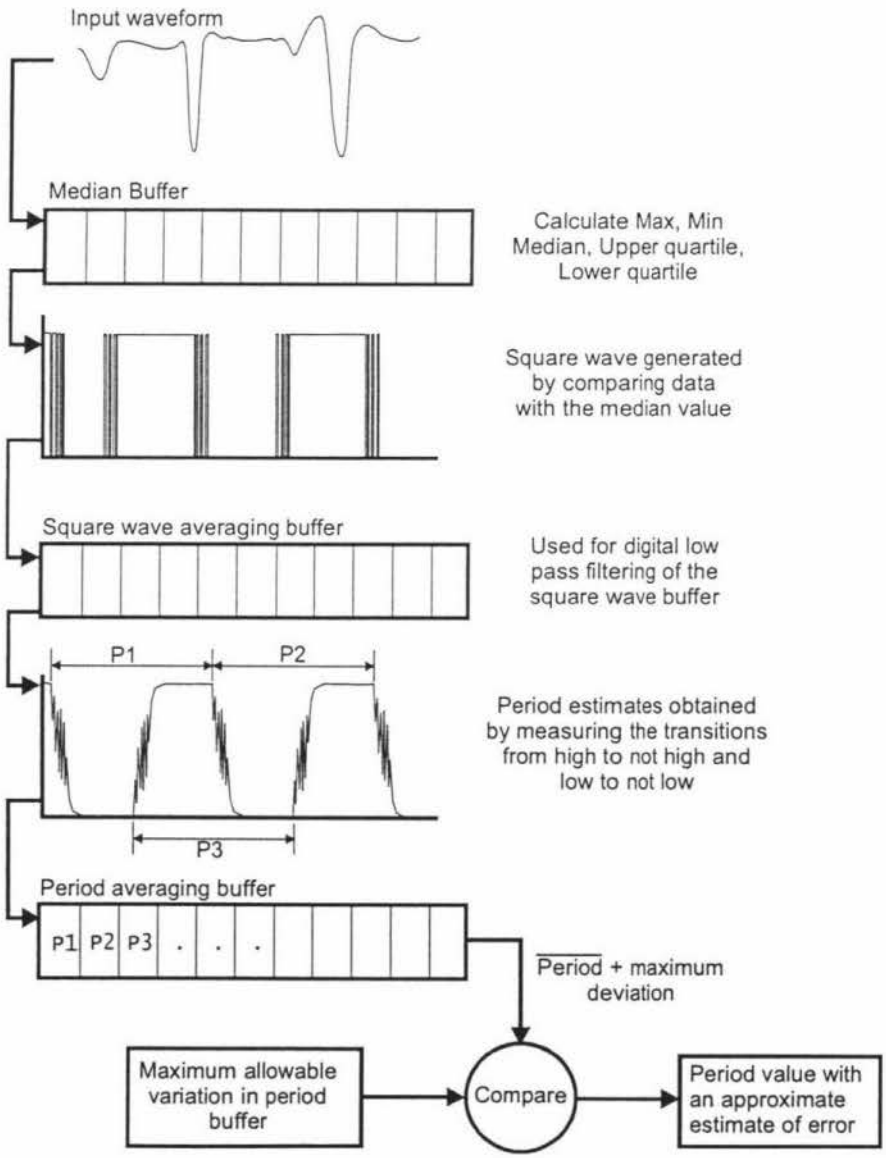


Figure 3.6: A graphical representation of the pulse rate calculation process.

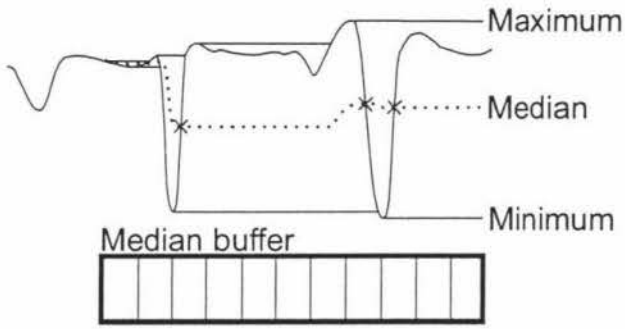


Figure 3.7: The beginning of calculation of the maximum, minimum and median values from the waveform stored in the median buffer. The median value was calculated from the maximum and minimum values, for clarity, the upper and lower quartiles are left out of this diagram. Markers show the points where data crosses the median line.

Having determined these values, the next point in the square-wave averaging buffer was a one if the current datum was greater than the median or zero if it was less. The algorithm, at this stage, was sensitive to waveforms that crossed the median more frequently than the true period (figure 3.8), a problem which was combated by comparing the datum with the upper or lower quartile.

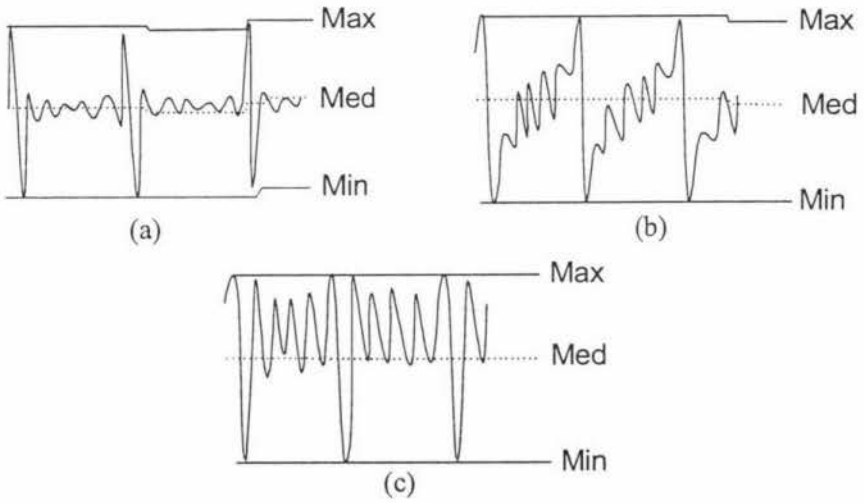


Figure 3.8: Example waveforms that posed a problem to pulse algorithm without the use of the upper and lower quartile in generating the square wave. Waveform (a) was observed when the sensor-head was attached too tightly and is suspected as the result of a combination of a pressure signal (upward spike) and a change in oxygen saturation (downward spike). During the development of the pulse algorithm and the device a triangular waveform was observed and waveform (b) is a noisy example. Waveform (c) is a noisy example of a typical waveform observed during testing of the device on humans.

The square-wave averaging buffer can be thought of as a digital low-pass filter used to generate the points of the smoothed square wave (figure 3.9). The algorithm searches for transitions in the waveform that occurred when neighbouring points of the smoothed square wave went from low to not low or high to not high indicating that the input waveform had crossed the median value. The levels for not low and not high were set at 10% and 90% of full scale (6554 and 58981 of 65535) (figure 3.9).

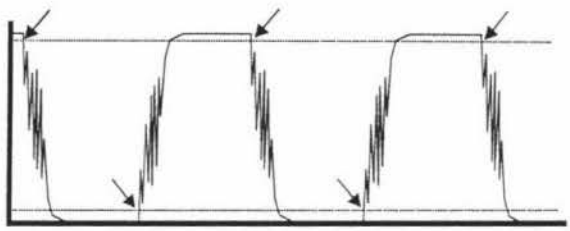


Figure 3.9: Transition points used to obtain period estimates.

Period information was obtained from the transition points by recording the number of samples that occurred between each similar transition. This number was entered into the period buffer that was then scanned to find an average value and the maximum deviation from the average. If the maximum deviation was below a predetermined threshold then the number of samples between similar transitions was accepted as a valid period estimate. The most accurate period (i.e. the one with the smallest deviation from the mean) was found over the course of the pulse measurement. After a fixed number of repetitions, the input data were compared with the upper quartile and then with the lower quartile during the square wave generating phase in an attempt to improve the pulse rate estimate. Once the main loop had completed, the values saved were the number of samples that had occurred during one period of the input waveform and the number of samples by which this value may deviate. Since the sampling frequency was known, the period estimate could be converted into a true period (i.e. in units of time instead of sample multiples) by multiplying the number of samples by the sample period.

3.4.2 ANALYSIS OF THE PULSE MEASUREMENT ALGORITHM

The pulse measurement algorithm was recreated and tested using Matlab to determine the limits within which a pulse measurement could be made. The three aspects investigated where the effect of drift in the input signal, the characteristics of the filter used to generate the smoothed square wave and the algorithms limiting susceptibility to noise.

As indicated in figure 3.7, the algorithm was dependent on the size of the median buffer. When the buffer was too small incorrect values for the signal maximum and minimum were obtained and small deviations in the input signal resulted in erroneous level changes in the square wave buffer. Conversely, if the median buffer was too long the input signal could drift away from the calculated median such that the signal no longer crossed the median value and no level changes were recorded in the square wave buffer. These two extremes imply that there was an ideal median buffer length

for which the effects of signal drift and level changes due to noise were minimised. This was investigated by using the level crossing technique described in the algorithm to generate a square wave from a simple sine wave with linear drift. Equation (3.1) describes the input signal used where A is the signal amplitude, B is the drift gradient.

$$Input\ Signal(x) = A\sin(x) + Bx \tag{3.1}$$

The effective length of the median buffer in relation to the input signal is a combination of the frequency with which the input signal is sampled and the absolute number of samples stored. A more logical description of the median buffer length is the fraction of one period stored. If, for example, the median buffer holds one complete period of the input signal then its effective length is the same if both the sample frequency and number of samples stored are doubled.

For input signals with various amplitudes, a graph of the limiting drift gradient is given in figure 3.10. This graph represents the boundaries at which the input signal ceases to cross the calculated median value and therefore ceases to generate the square wave. The greatest impunity to drift and erroneous level crossings is achieved using a median buffer that contains between 1 and 1.7 input signal periods. The Matlab program written to find these limits is given appendix B.3.

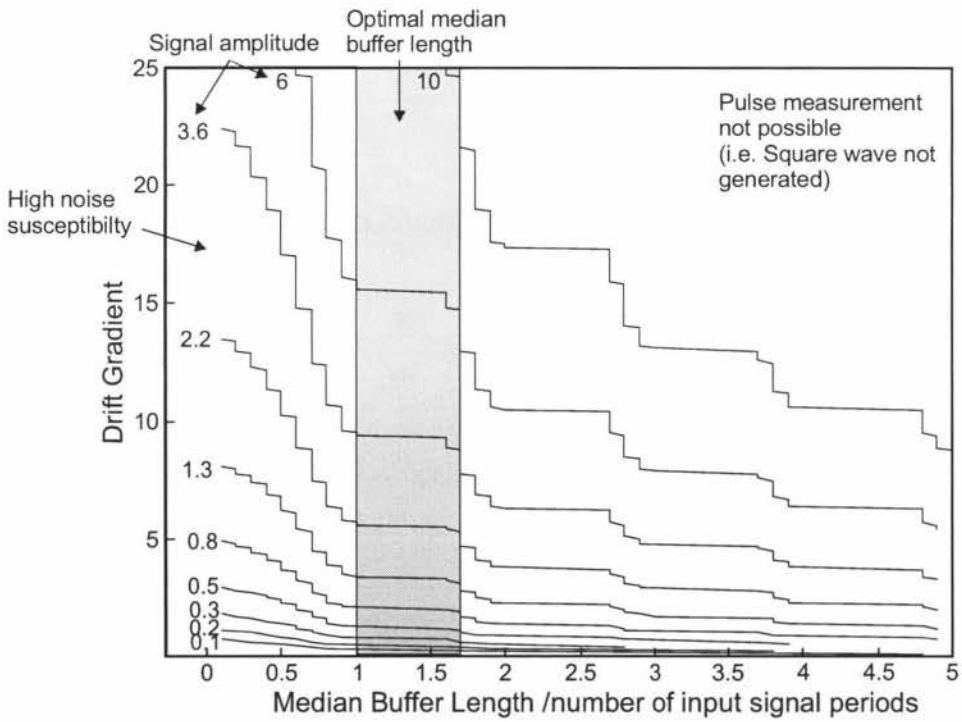


Figure 3.10: The boundaries for which each input signal ceases to cross the calculated median value and therefore ceases to generate the square wave. As the median buffer length decreases below 1 the susceptibility to noise when generating the square wave increases. Independence of drift as well as noise rejection is best achieved using a median buffer length of 1 to 1.7 input signal periods.

Another major component of the pulse algorithm was the digital low-pass filter used to generate the smoothed square wave (figure 3.6). The frequency characteristics of the digital filter place a limit on the maximum measurable period so to determine the filter response, discrete signal analysis was used [25].

Equation (3.2) is a function that describes the output of the digital filter (a finite impulse response filter) where $y(n)$ represents the points of the smoothed square wave,

$$y(n)=\frac{1}{8}[x(n)+x(n-1)+x(n-2)+...+x(n-7)] \tag{3.2}$$

The steady-state transfer function corresponding to (3.2) is

$$\mathrm{H}\left(e^{j\omega T}\right)=\frac{1+e^{-j\omega T}+e^{-2j\omega T}+...+e^{-7j\omega T}}{8} \tag{3.3}$$

where $\frac{1}{T}$ is the sampling frequency. Equation (3.3) is a finite geometric series that can be written as

$$\mathrm{H}\left(e^{j\omega T}\right)=\frac{1-e^{-8j\omega T}}{8\left(1-e^{-j\omega T}\right)} \tag{3.4}$$

After some manipulation, equation (3.4) becomes

$$\mathrm{H}\left(e^{j\omega T}\right)=\frac{\left(e^{4j\omega T}-e^{-4j\omega T}\right)}{8\left(e^{\frac{1}{2}j\omega T}-e^{-\frac{1}{2}j\omega T}\right)}e^{\frac{-7}{2}j\omega T} \tag{3.5}$$

This result may then be expressed as amplitude (3.6) and phase (3.7) functions

$$\mathrm{A}(\omega)=\frac{\sin(4\omega T)}{8\sin\left(\frac{\omega T}{2}\right)} \tag{3.6}$$

$$\beta(\omega)=\frac{-7\omega T}{2} \tag{3.7}$$

The frequency response of the digital filter is given in figure 3.11 for a sampling frequency of 100 Hz, the same sampling frequency used in the example script file given in table 3.8

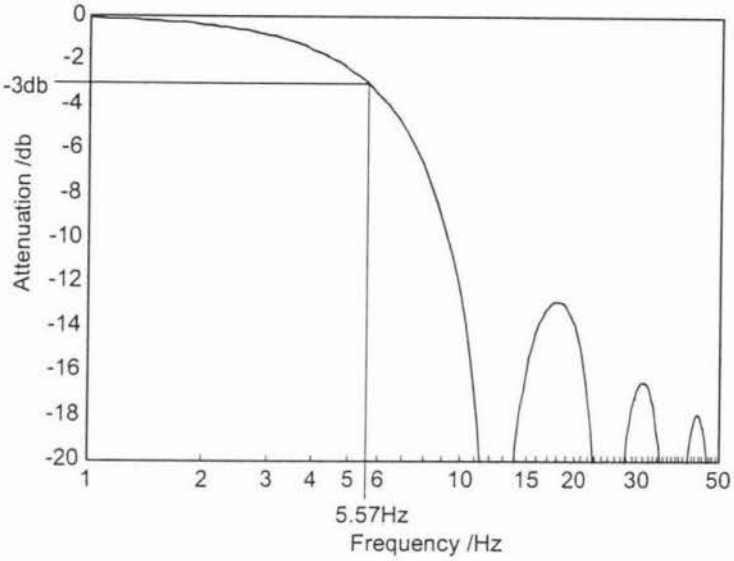


Figure 3.11: Transfer function of the digital low-pass filter used in the pulse measurement algorithm.

One of the effects of the digital filter was to place a limit on the maximum pulse frequency that the algorithm could measure. Figure 3.12 shows how the algorithm begins to fail as the pulse frequency increased beyond the corner frequency of the digital filter. The corner frequency at 5.6Hz corresponds to a pulse rate of 336 beats per minute, a normal human pulse rate lies in the range of 50 to 180 beats per minute which corresponds to a frequency range of 0.83 to 3 Hz. Clearly, attenuating signals above the corner frequency does not influence the measurement of realistic human pulse rates. It was expected that the maximum pulse rate of the Adélie penguin is also less than this corner frequency.

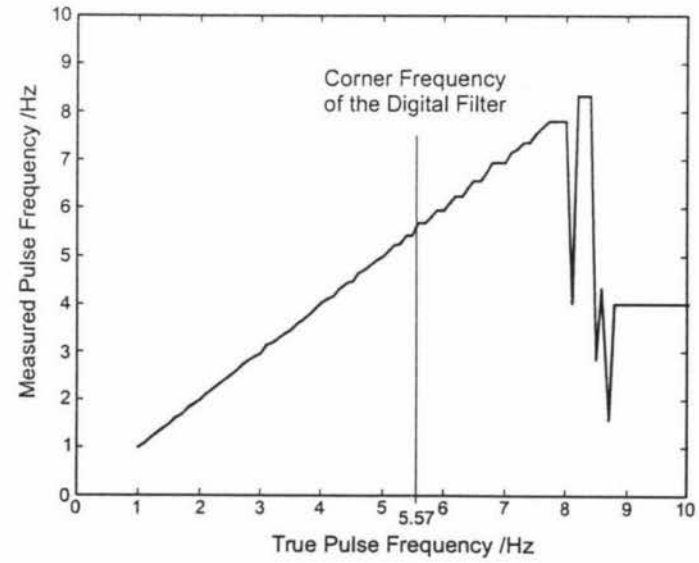


Figure 3.12: The performance of the pulse measurement algorithm as the pulse frequency increased beyond the corner frequency of the digital low-pass filter (based on the Matlab simulation given in appendix B.4).

Another important measure of the pulse algorithm was its performance with input signals of varying levels of signal to noise. This was investigated using sine and asymmetric square waves with added computer generated white noise as test signals. The asymmetric square wave approximates real signals observed during tests on human subjects. Figures 3.13 and 3.14 show the limiting signal to noise ratio at which the algorithm was unable to measure a pulse for the sinusoidal and square waveforms respectively. The chosen waveforms had a true period of 0.7s, similar to an average human pulse rate. The Matlab programs used for this simulation are given in appendix B.5.

For the sinusoidal test signal (figure 3.13) the limiting signal to noise ratio was approximately 2 and in the asymmetric square wave case (figure 3.14), approximately 2.8. In general, the input waveform should have a signal to noise ratio of greater than 3 in order to expect a meaningful result from the algorithm.

Finally, an example of the pulse algorithm applied to real data is given in figure 3.15. The input signal was collected during testing of a prototype version of the device and is typical of real signals encountered by the pulse algorithm. In this case the subject was recovering from physical exertion and had a heart rate of approximately 102 beats per minute.

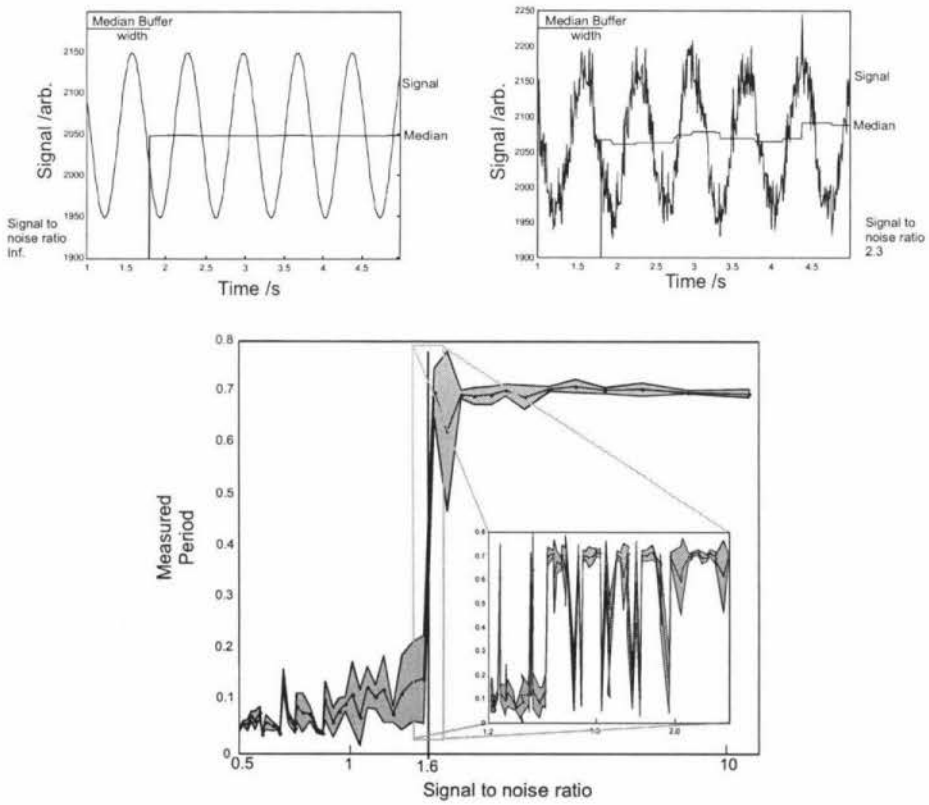


Figure 3.13: The limiting signal to noise ratio for a sinusoidal input signal. Grey regions represent error in the measured period. It can be seen from this plot that a signal to noise ratio of greater than 2 is required in order to expect accurate period measurements.

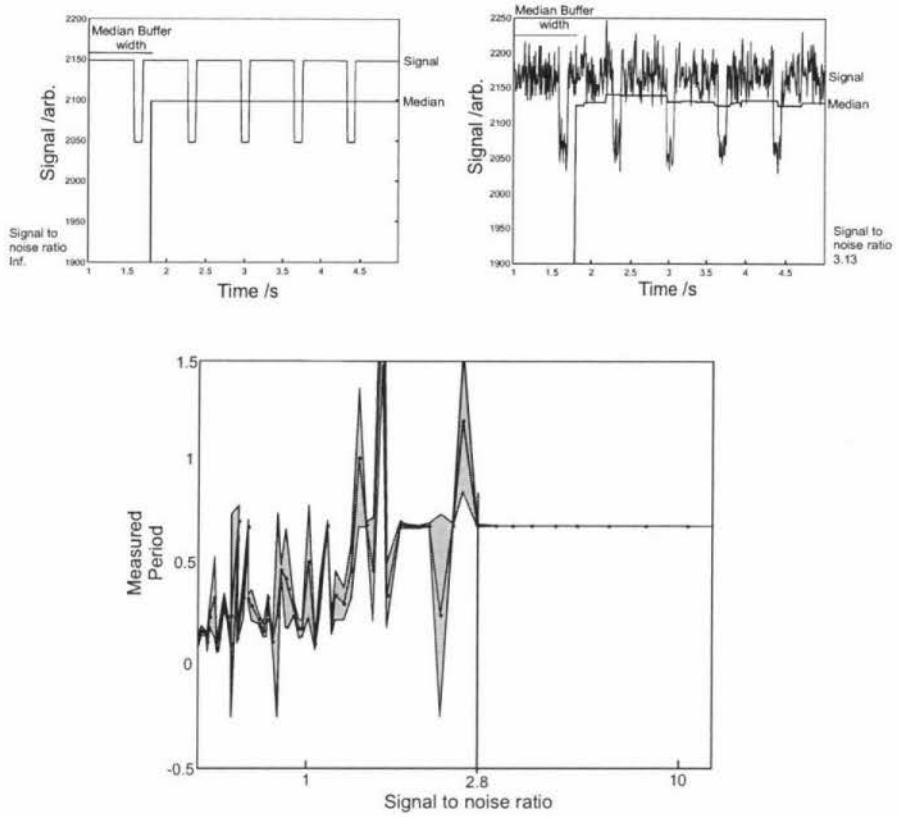


Figure 3.14: The limiting signal to noise ratio for an asymmetric square wave input signal of similar shape to acquired human data. In this case a signal to noise ratio of at least 2.8 was required for accurate period measurement. In the general case a signal to noise ratio of at least 3 – 4 should be used.

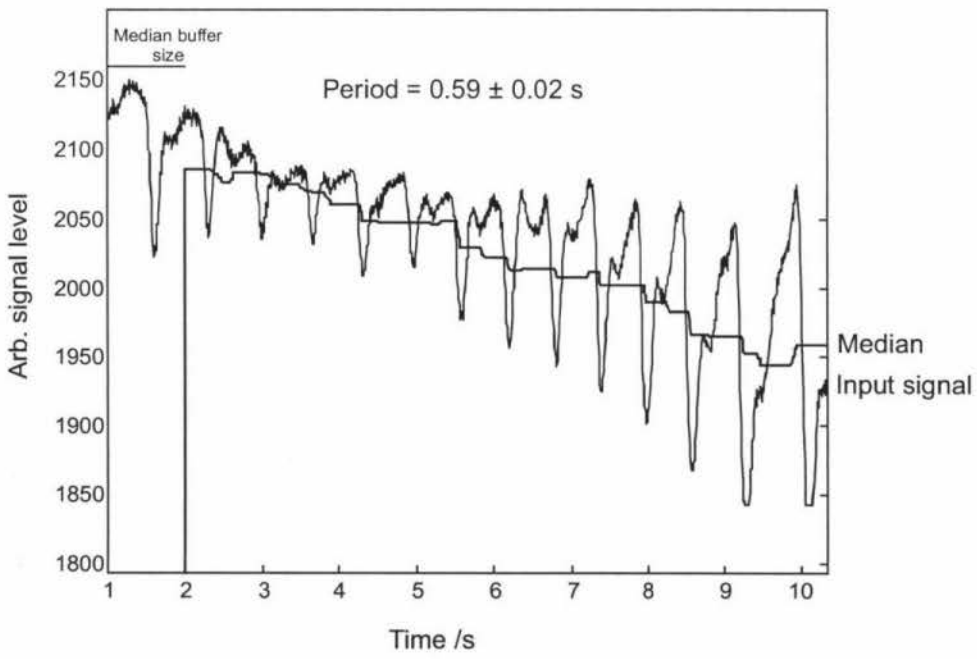


Figure 3.15: A pulse measurement example using real data collected during testing on a human subject.

Chapter 4

Prototyping and Application

4.1 Developmental Testing

The device was developed in stages. Four prototype versions were built starting with a very basic emitter and detector system. Each prototype allowed specific aspects of the design to be tested. The final device incorporated the results obtained from the prototypes and was field-tested in Antarctica.

4.1.1 EARLY PROTOTYPES

The earliest prototype consisted of a 660nm LED, photodiode detector and amplifier connected to an oscilloscope. The source and detector were arranged in transmission mode and the signal was obtained through a human finger. The primary use of this system was to confirm that a signal, related to a pulse, could be obtained using an LED light source. Results from this prototype indicated that the detected signal strength was of the order of millivolts and noise susceptibility was strongly affected by changes in background light, skin pigmentation, electrical noise and skin temperature. Increased signal strength was achieved using a large area photodiode and two LEDs but noise susceptibility and a need for further improvement in signal strength led to the use of synchronous detection.

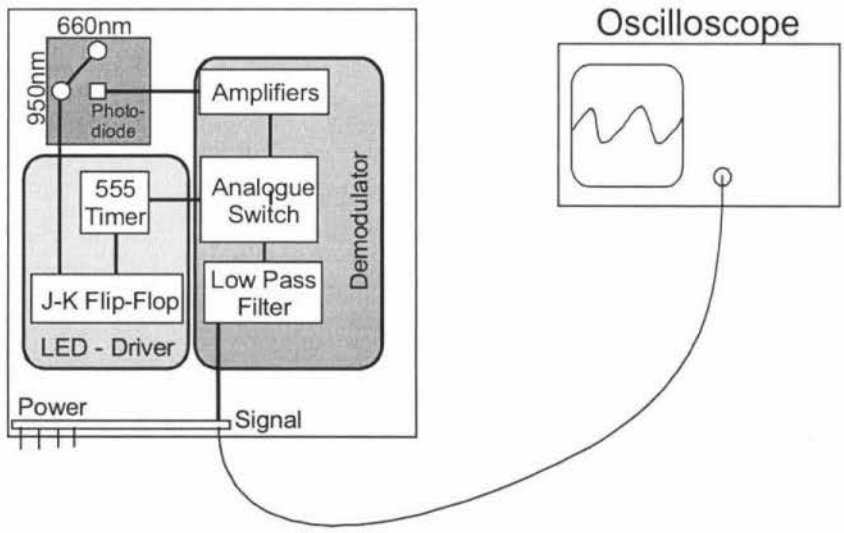


Figure 4.1: The second prototype. With this system, sufficient signal strength was obtained using a single LED light source and the synchronous detection technique. This prototype was also the first to use reflectance mode detection.

The second prototype (figure 4.1) used reflectance mode detection and consisted of two pairs of 660nm and 950nm LEDs arranged about the photodiode. The LEDs were modulated at 5kHz by a driver circuit constructed from a 555 timer and a J-K flip-flop. The resulting optical signal was coupled to a demodulator circuit consisting of an analogue switch (controlled by the clock output of the 555 timer), a low-pass filter and

a second amplifier. Again, the demodulated signal was displayed using an oscilloscope. The signal to noise ratio for the optical signal was improved significantly using synchronous detection which allowed the signal to be amplified to an acceptable level for digitisation and acquisition. Also, the second prototype indicated that sufficient signal strength could be obtained from a single LED. Therefore only single light sources were used for each wavelength in the subsequent prototypes.

4.1.2 SIGNAL VERIFICATION USING AN ELECTROCARDIOGRAM (ECG)

To confirm the biological value of the observed signals it was necessary to demonstrate a correlation between signals from the prototype and a recognised medical instrument. Data were collected simultaneously from both an ECG (electrodes attached to the right arm, chest and left leg) and the prototype device (optical signals obtained from the index finger or thumb). The QRS complex in the ECG waveform, corresponding to depolarisation of the ventricles and hence blood flow into the aorta [26], coincides with the rapid drop in the absorbance signal due to an increase in oxygenated blood at the finger (point (i) in figure 4.2). The time difference between these two events, of approximately 0.2s, was due to the time required for the wave of blood to reach the finger.

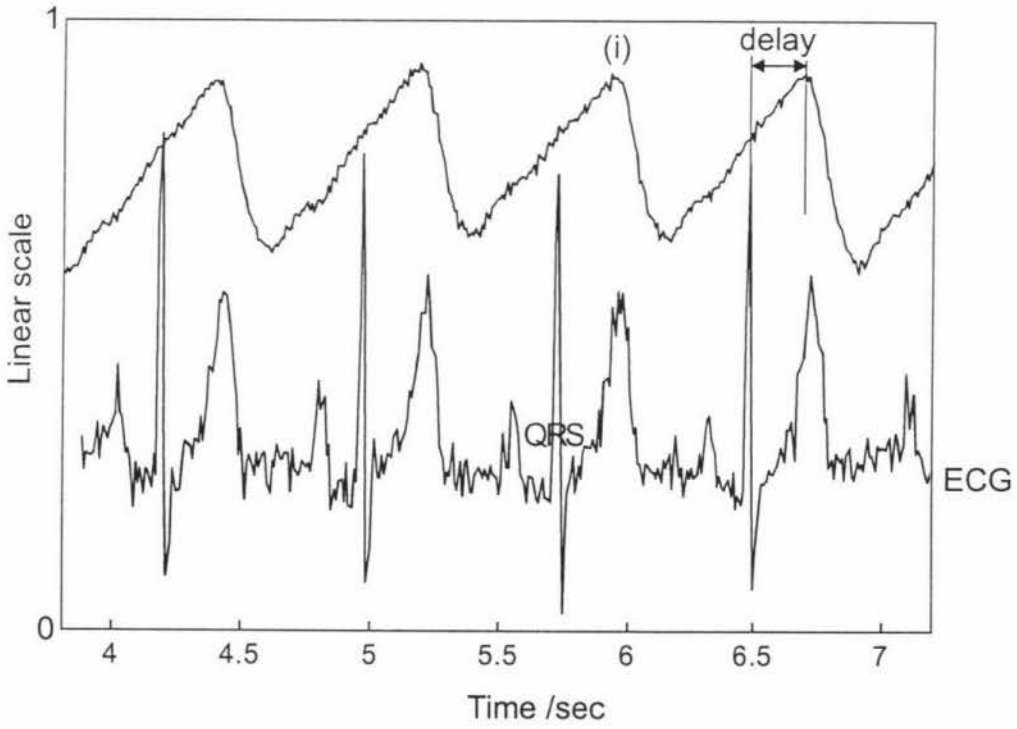


Figure 4.2: The correlation between ECG data and the signal obtained from the second prototype. Measurements were taken simultaneously from the arm, chest and leg for the ECG and from the index finger or thumb for the second prototype. The QRS complex is related to blood flow and is correlated with the fall in the absorbance signal at point (i) (Delayed by approximately 0.2s).

A second example of the ECG and second prototype comparison (figure 4.3) illustrates the signal obtained for a subject with an increased heart rate (~93 bpm). More significant differences between figures 4.2 and 4.3 are the increased amplitude of the optical signal in figure 4.3 and the change of the waveform shape. Greater signal amplitude is likely to be the result of increased blood volume and the rise in absorbance (after point (i), figure 4.3) is consistent with an increased rate of oxygen consumption within the blood as a result of exercise.

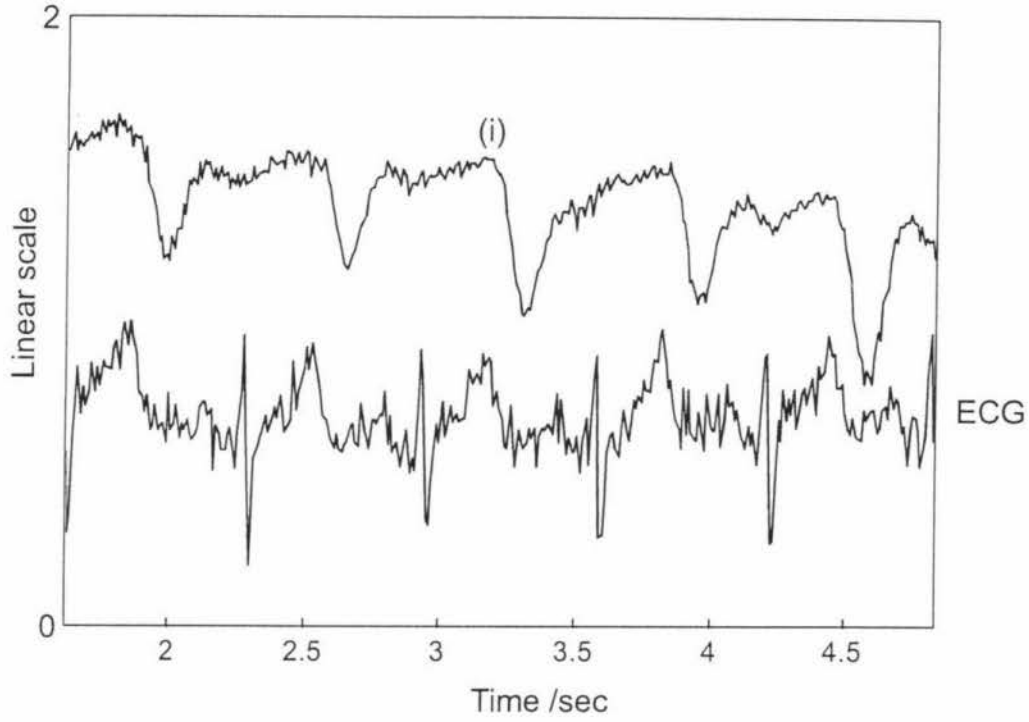


Figure 4.3: ECG absorbance signal comparison for a subject with increased heart rate. Important differences from figure 4.2 in this signal are increased signal amplitude as a likely result of greater blood volume and the rapid rise in absorbance after point (i) indicating an increased rate of oxygen consumption.

4.1.3 SOFTWARE DEVELOPMENT SYSTEM

With the signal level established the focus of development moved to the control aspects of the device. The third prototype system was constructed on printed circuit board and designed to attach to the Motorola 68HC11 development board. This prototype had a sensor head similar to that of the final design and the LED modulation circuit was driven by interrupts from the microprocessor. The major advancement of this prototype, however, was signal acquisition using the on-chip, 8-bit, analogue to digital converter. This allowed testing of various software components including data sampling, signal analysis (in the case of pulse measurements) and communication between the prototype and a computer. By relaying acquired data to the computer, basic blood oxygenation and pulse measurement experiments were run that imitated the experiments to be run by the final device. One result gained from these experiments was the inadequacy of the 8-bit analogue to digital converter and the need for higher resolution sampling. As shown in figures 4.4, 4.5 and 4.6, the acquired data are severely resolution limited which prompted the use of the 12-bit analogue to digital converter in the final device.

4.1.4 BLOOD OXYGEN SATURATION

By configuring the third prototype to continuously sample and relay the acquired data through the serial port, the terminal software (TEmu) could then perform the blood oxygenation calculations (equation 1.8) and graph the results. The sensor head was fitted to the wrist of the subject who, after a minute of signal stabilisation time, commenced physical activity by cycling an exer-cycle. After approximately two minutes the subject finished the exercise and waited for a resting breathing rate to return (figure 4.4). These events are reflected by changes in the oxyhaemoglobin concentration.

The cellular demand for oxygen varies in proportion to the level of physical activity. This is indicated by the fall in oxyhaemoglobin concentration when exercise begins, (start of physical exercise, figure 4.4). This trend approaches the steady state condition where the cellular demand for oxygen is balanced with the oxygen delivery by haemoglobin. Physical activity ceased after approximately two minutes and hence the cellular oxygen demand decreases. This corresponds to the rapid increase in the oxyhaemoglobin concentration, (end of physical exercise, figure 4.4). After another period of approximately a minute the subject's heart rate and breathing rate return to normal and the saturation curve stabilises.

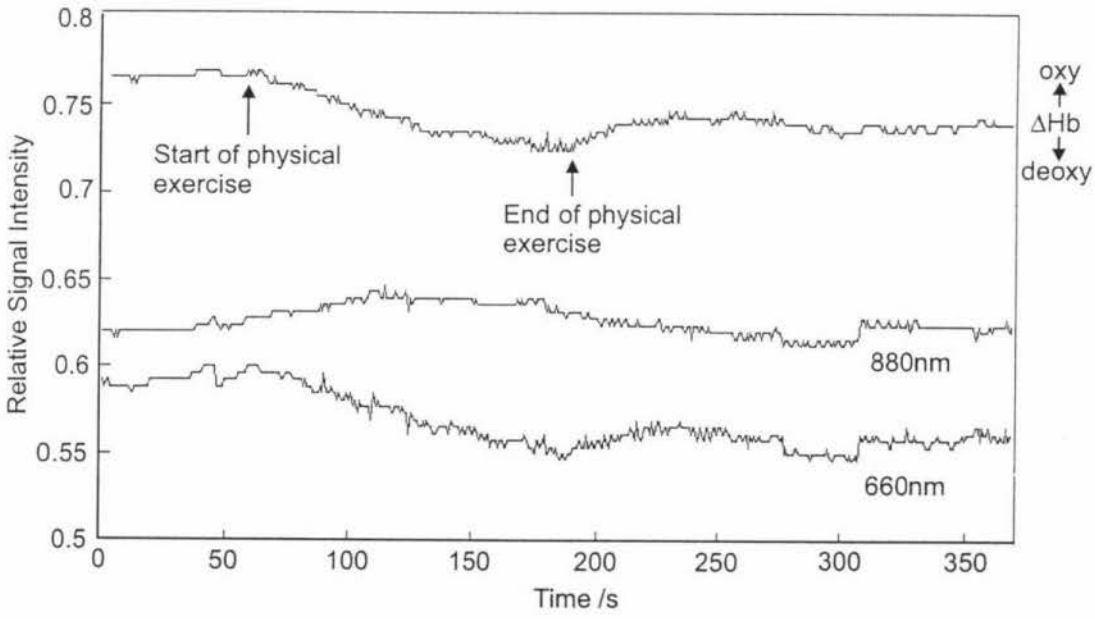


Figure 4.4: Blood oxygen saturation measurement for a subject undergoing approximately two minutes of physical exertion on an exer-cycle. As physical activity proceeds, the demand for oxygen increases and hence the blood oxygen saturation decreases. At the end of the exercise period the elevated breathing rate and reduced cellular demand for oxygen cause the blood oxygenation of the subject to rapidly increase corresponding in the rise at ~190s. After 60s of rest the blood oxygenation levels stabilise. These blood oxygen data were calculated using the double wavelength equation, (1.8).

4.1.5 PULSE RATE MEASUREMENT

The pulse measurement algorithm was first designed and simulated using Matlab and the signals acquired from the ECG experiments. The major limitations taken into account when developing the algorithm for the microprocessor were integer computation and limited algorithm complexity due to speed and space restrictions. The algorithm was based on a digital filtering and level crossing technique that was sufficiently robust to accommodate input signals with noise. The algorithm was then implemented in the TEmu program and used with real time data from the third prototype to perform preliminary pulse rate measurements. Further refinements and optimisations were made based on these preliminary pulse measurements (such as level crossings based on the upper and lower quartile (section 3.4)), before the algorithm was implemented in assembly language for the final device.

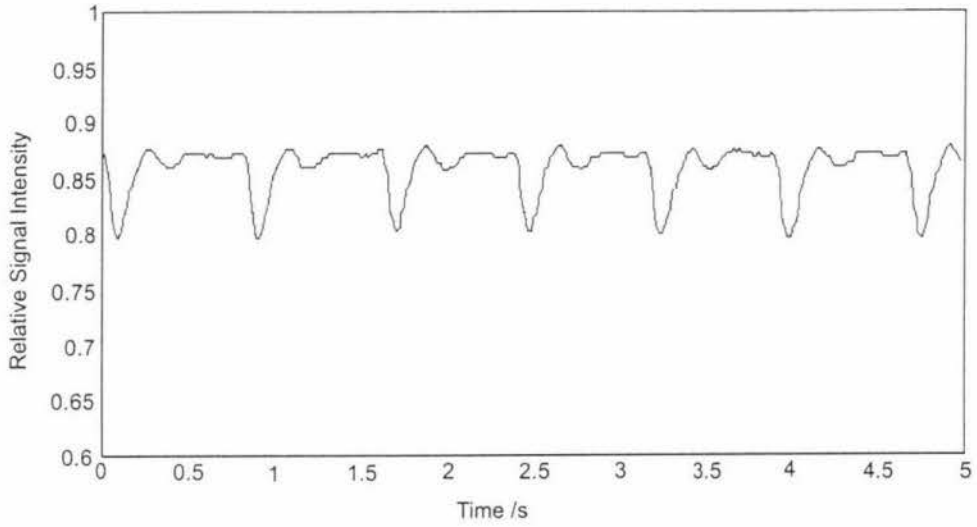


Figure 4.5: Resting pulse signal. A result obtained from the pulse measurement algorithm implemented within the TEmu software that gave a heart rate of 77 (±2) bpm for this signal.

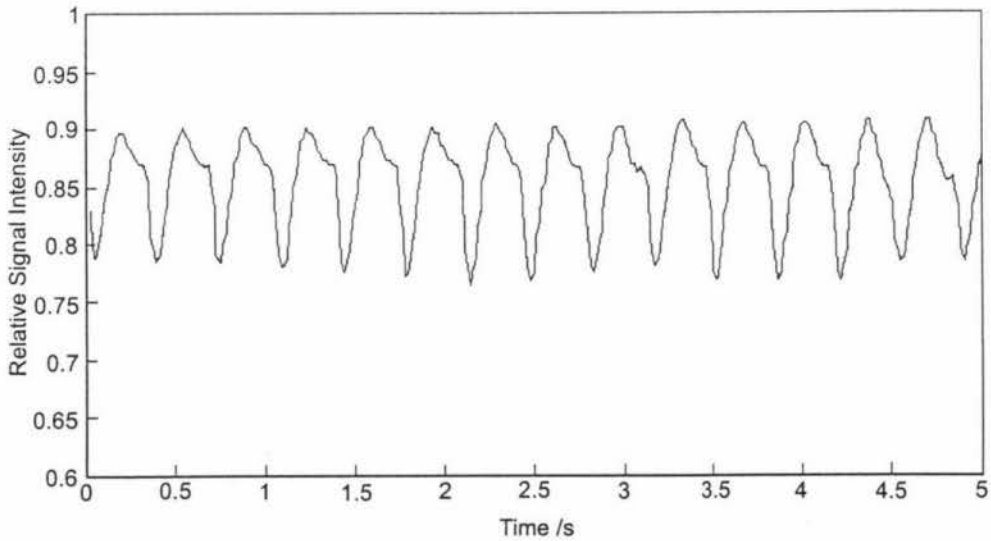


Figure 4.6: Pulse signal after exercise. In this example the subject's heart rate was increased by physical exercise and measured by the algorithm to be 170 (±5) bpm.

4.1.6 THE STAND-ALONE PROTOTYPE

With many of the control and hardware modules individually tested (section 3.2.1) a system was needed to amalgamate the various software components and allow final software development for the memory chips and 12-bit analogue to digital converter. This prototype was intended to be as close to the final device as possible therefore design aspects such as minimising physical size, connectivity between the device and a computer, and package construction were incorporated. The device was built using double-sided printed circuit board and the same electronic components that would be used in the final design. Since power to the device was provided by a battery supply, the power management circuitry was also developed.

Significant development of the control software took place with this prototype as it was the first system to have enough programmable memory space to include the entire operating system. Using the branch vector technique (section 3.1.5), the memory and sampling modules were developed and software bugs that were not apparent during module testing were found and resolved. Once development was complete, six final version devices were built based on this prototype.

4.1.7 TESTING OF THE FINAL DEVICE

Blood oxygenation, tissue oxygenation and pulse-rate measurements were made using one of the six devices. A measurement sequence program that made a pulse measurement, six light scattering measurements using each LED and a temperature measurement repeatedly without delay was uploaded to the device. After the measurement sequence had begun and the sensor head was attached to the thumb with tape, approximately 5 to 10 minutes were spent resting before physical activity started. Increased heart rate and oxygen consumption were achieved by running up and down stairs for approximately 2 minutes, during which, breathing rate was controlled as much as possible. Initially, breathing rate was suppressed so that oxygen demand would outweigh the supply resulting in decreased tissue and blood oxygenation. When breathing rate suppression became too difficult, a normal breathing rate was used until the end of the exercise period. During the recovery period, an increased breathing rate was maintained so that oxygen supply to the tissue was now greater than the demand, resulting in a rise of the oxygen concentrations. Figures 4.7 to 4.10 show the results collected during one of these tests.

The raw data collected by each of the six LEDs are shown in figure 4.7. While each wavelength exhibits the same basic trends, there were noticeable differences between each signal during the exercise period. The relative changes in detected intensity for the 880nm and 950nm signals were greater than the change in the 660nm signal and similarly for the 605nm signal that varied more than the 625nm and 590nm signals. These relative differences were expected as the result of the changing oxygen demands during exercise that give rise to the changes in blood oxygenation and tissue oxygenation.

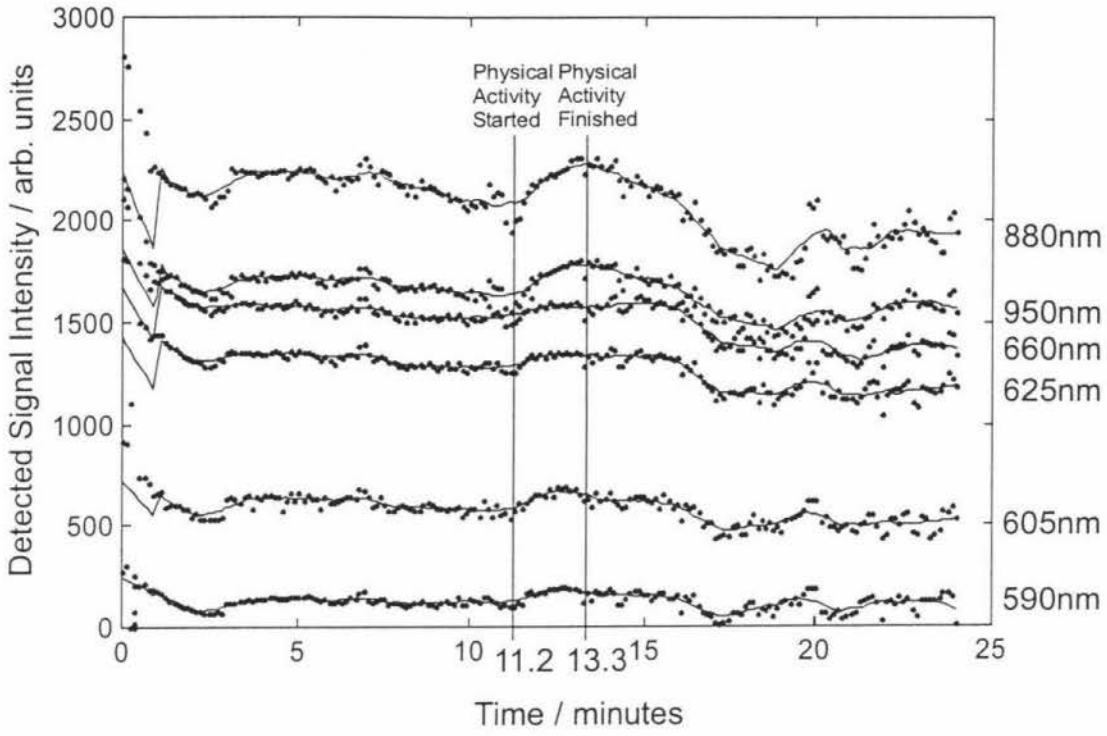


Figure 4.7: Intensity data acquired from the device. These raw signals show the relative signal strengths for each LED. While the same general trends are apparent in each signal, physical activity resulted in noticeable differences between each wavelength.

The change in relative blood oxygen saturation, as calculated using the double and triple wavelength equations (1.8 and 1.14), is shown in figure 4.8. The wavelengths used were 880nm and 660nm for the double wavelength calculation and 880nm, 660nm and 625nm for the triple wavelength calculation. In each case there was a clear decline in haemoglobin oxygenation as a result of physical exercise that returned to normal once the exercise had finished. Although the change in blood oxygen saturation is more pronounced in the double wavelength measurement the signal also suffers significant baseline drift that is not present in the triple wavelength measurement.

The affect of physical exercise on the reduction state of cytochrome oxidase was considerably less than its affect on blood oxygenation due to the greater opacity of tissue at the shorter wavelengths (590nm to 625nm). Consequently the signal had a lower signal to noise ratio and the oxygenation changes were less obvious (figure 4.9). For the double wavelength calculation the wavelengths, 605nm and 625nm, were used and a reduction in the oxygenation signal can be seen during the exercise period although the correlation is not strong. For the triple wavelength measurement, the additional wavelength, 590nm, was used. As the magnitude of the changes in blood-oxygenation were reduced in the triple wavelength measurement it is likely that changes in tissue oxygenation, calculated using the triple wavelength equation, were overwhelmed by noise. In both signals, however, the upward trend after the exercise period may be due to the increased supply of oxygen and reduced demand during the resting period.

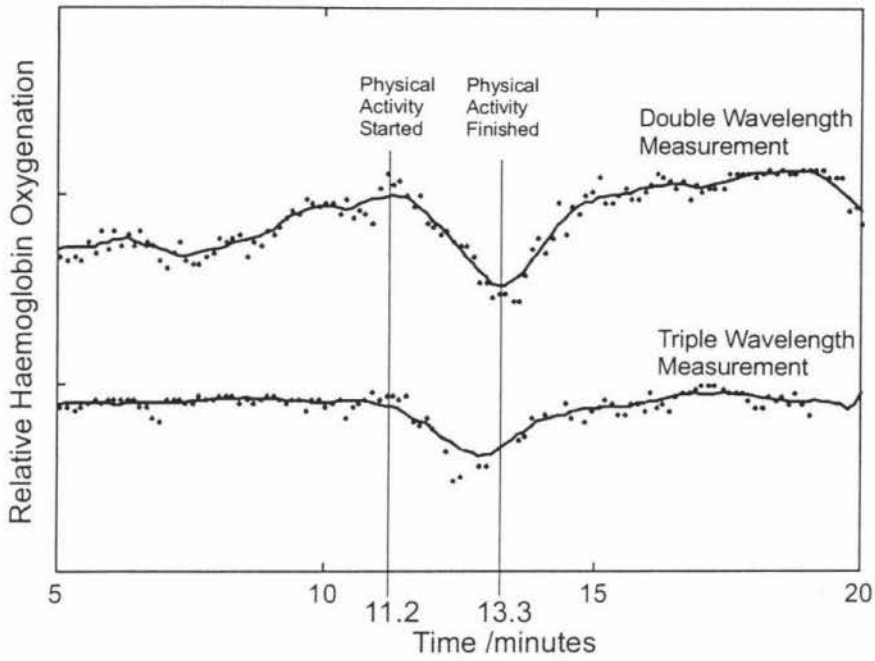


Figure 4.8: Relative blood-oxygen saturation. Physical activity resulted in a clear decline in haemoglobin oxygenation that increased again once activity had finished. Noticeable differences between the calculation techniques are that the double wavelength measurement appeared to drift while the base line for the triple wavelength measurement was approximately constant. The difference in haemoglobin oxygenation, however, was more pronounced in the double wavelength measurement.

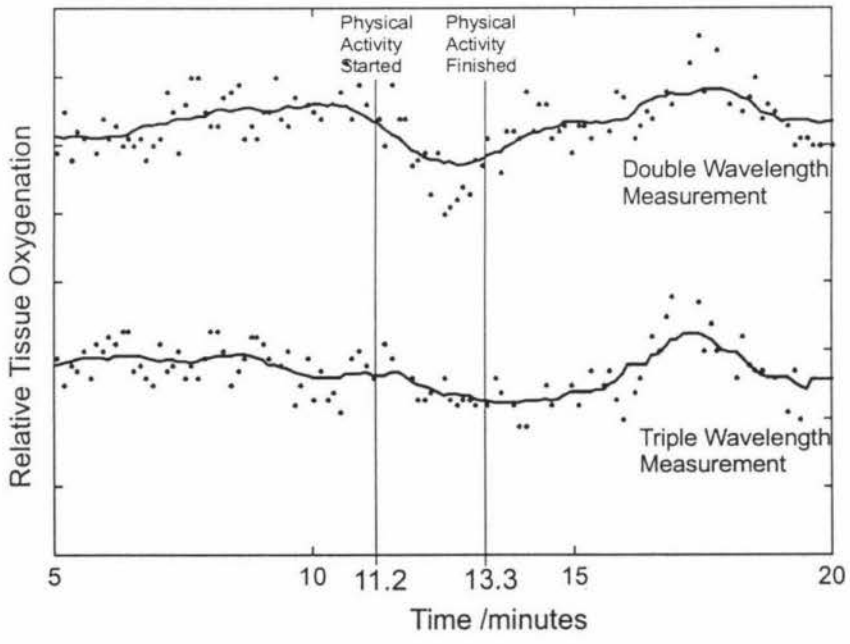


Figure 4.9: Relative cytochrome oxidase oxygenation. These data were calculated using wavelengths for which tissue opacity was greater and consequently the signal to noise ratio was lower. A downward trend in the double wavelength measurement is visible during the physical activity period but not obvious in the triple wavelength measurement for which the effects of noise appear to overwhelm the signal. After the exercise period, the upward trend could be due to the increased supply of oxygen and reduced demand by the cells.

The relative change in blood volume as a result of physical exercise is shown in figure 4.10. The body's reaction to physical activity is to increase the supply of oxygen to the muscles. Increased blood flow is another means of achieving this so a rise in blood volume during the exercise period was an expected result. Even though there was obvious baseline drift in the signal, a marked increase in blood volume occurred while exercising that subsided to the normal level approximately 5 to 7 minutes after the activity finished.

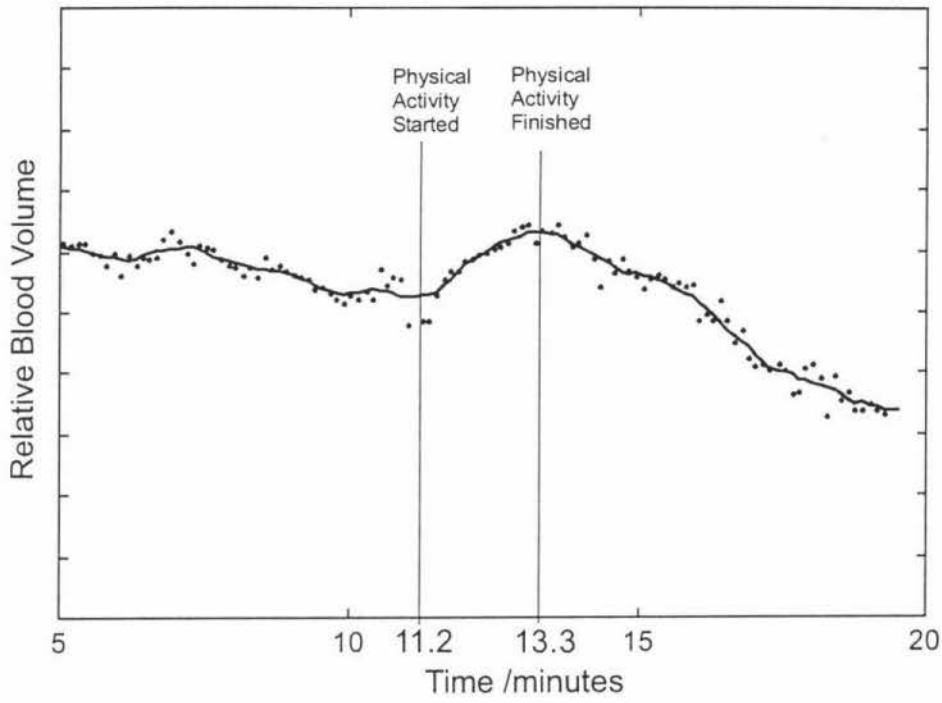


Figure 4.10: Relative blood volume. Apart from the drift in the signal, the blood volume clearly increased during the physical activity period and slowly returned to normal once the activity had finished.

Pulse rate data from this experiment was inadequate due to noise introduced by movement of the sensor head during the exercise. In other experiments, where this movement was reduced (muscle tension exercises), more consistent pulse rate data were obtained (figures 4.11 and 4.12).

From the waveforms used to calculate the pulse rate (example given in figure 4.11), variations are clearly visible at points (i) and (ii). These variations are possibly due to a reflected pulse that occurs as a result of the elasticity of blood vessels (point (ii), figure 4.11) [Machon, R., Veterinary science lecturer, personal communication]. The higher sampling resolution of the final device also shows a third reflected pulse at point (iv).

Data obtained during a muscle tension exercise are given in figure 4.12. Initially the pulse rate increased rapidly to approximately 110 beats per minute. Once the exercise had finished, a normal pulse rate returned within about three minutes. As a consequence of sensor head movement, larger error estimates and fewer points were recorded during the exercise period.

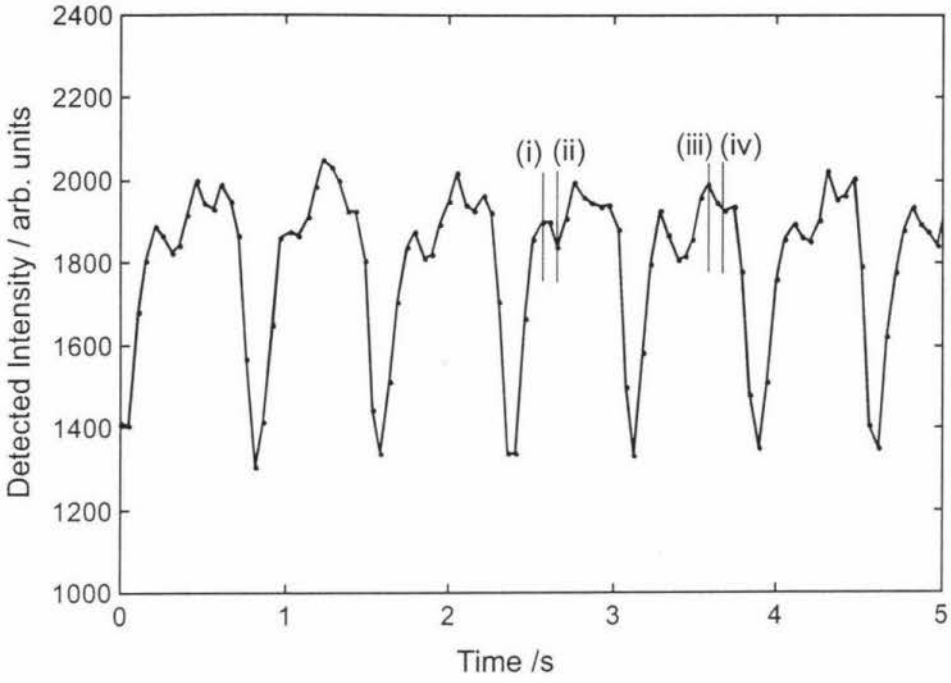


Figure 4.11: A pulse waveform used to calculate pulse rate. The points of this waveform are received in real time by the pulse algorithm that attempts to find the signal periodicity. Points (ii) and (iv) indicate fluctuations in the waveform that may be due to second and third reflections of the initial pulse arising from the elasticity of the blood vessels.

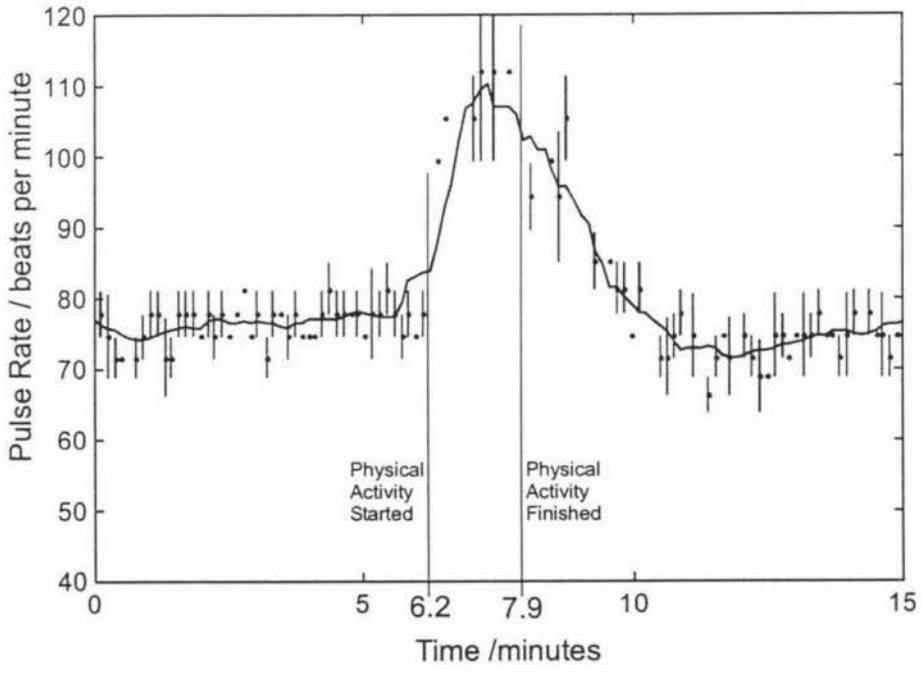


Figure 4.12: Pulse rate measurement. From signals similar to that of figure 4.11, pulse rate information was calculated. In this experiment, the pulse rate rapidly increased as a result of physical activity and then gradually returned to normal once the activity had finished. Movement of the sensor head during a pulse rate calculation can disturb the pulse waveform and prevent the pulse algorithm from finding a result, hence the scarcity of points during the exercise period. The error bars in this graph indicated the pulse rate error estimates (section 3.4).

Finally, the response of the temperature sensor was investigated by the application of a heat source to the sensor head resulting in the temperature curve given in figure 4.13. The initial rise in temperature within the first minute is most likely related to the heating effect of current flowing through the device. When placed in contact with the heat source the time constant for the device was 18.6s. Once the heat source was removed, approximately 15 – 20 minutes were needed for the sensor to return to 18° by dissipating its heat into the surrounding air. Temperature measurements made during exercise experiments or on nesting Adélie penguins were made on a larger time scale than 18.6s so the time constant was presumed acceptable.

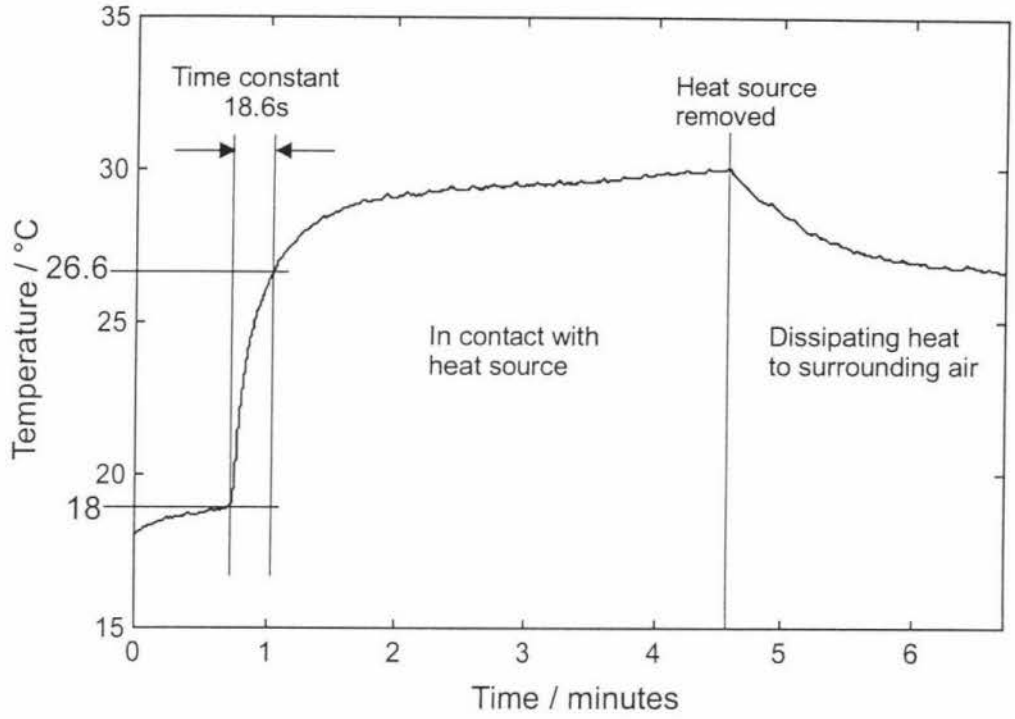


Figure 4.13: Temperature measurement. Measurements were made over larger time scales than the 18.6s time constant and therefore should not significantly affect the temperature measurements. Other important considerations indicated by this graph were the need for good thermal contact and the stabilisation time needed to avoid the internal heating effects of the device.

4.2 Field Testing

Field testing, on the Adélie penguins in Antarctica, began on the 22nd of November, 1999 using six of the final version devices. The initial intention was to use all six devices to perform the same experiment on six birds simultaneously so that typical biological responses could be compared with the behavioural activities observed. This soon proved to be impractical however, due to the difficulties in tracking the instrumented birds as they moved within the large colonies. Once an instrumented bird was out of sight, the black coloured packaging of the device made it exceptionally difficult to relocate. Therefore, a single device was attached to a penguin and constant observation of its activities were maintained from a distance using binoculars. After a data collection period of 1 to 5 hours the bird was recaptured and the device retrieved.

Due to the nature of this experimental technique, conclusions drawn from these experiments suffer the limitation of a small sample size. However, consistency between behavioural activity and biological responses would indicate that meaningful data were obtained from the device and that prediction of the penguins biological mechanisms in response to cold and stress could be made.

Aside from the biological data to be acquired by the device, physical data such as weight, flipper and beak dimensions were also measured. These morphometric parameters were used with a discriminant function (4.1) to determine the sex of each bird [27]. In some cases, however, the sex of the bird was apparent due to observations of its behaviour (e.g. mating). The gender information was obtained in case differences were observed in the biological responses of male and female penguins.

$$D = 0.582B_L + 1.118B_D + 0.219F_W \qquad (4.1)$$

where B_L = bill length, B_D = bill depth, F_W = flipper width and D = the discriminant value that was compared to the mean discriminant score of 55.39. This technique is claimed to predict the gender of Adélie penguins correctly in 89% of cases. However, a lower accuracy than this was expected due to interpretation of the measurement technique and location of the colony.

4.2.1 CAPTURE TECHNIQUE AND ATTACHMENT

As non-invasive measurement by the device was an important consideration, so too was the entire measurement process. Care was taken to minimise stress on the birds during capture, attachment and removal of the device and all manipulations of the penguins were approved by the Massey University Animal Ethics Committee.

Capture of a penguin involved quietly moving toward the bird with a wide area, shallow net held horizontally and low to the ground. When within approximately 2m to 3m the net was used to capture the bird and then both net and penguin were carried away from any nearby birds. Once removed from the net the penguin's hind legs were held in the handler's right hand, while the bird's head was tucked under the handler's left arm so that the penguin's eyes were covered. The penguin was held in this position, facing downward, so that the device could be attached to the bird's lower back and once in this position, the penguin became noticeably calmer.

The attachment technique used was similar to that described by Wilson *et al.* [28] whereby 'Tesa' tape was used to attach packages of varying weight to African and Adélie penguin. Tesa tape was described as a cloth backed tape that is light, strong, inexpensive, waterproof, non-restrictive and simple to apply. Unfortunately, this German tape was unavailable so a tape with the same specifications as Tesa 4651 was selected. The 'Scotch' brand tape chosen was a strong, light, cloth backed tape that, most importantly, was available in black (section 2.1).

Attachment of the device was achieved in three steps (figure 4.14). To begin with, a patch of feathers on the lower back of the bird was raised and a strip of tape, approximately 20cm x 4cm and adhesive side facing outward, was placed under these feathers. The device was then located over the centre of the tape and the left and right

edges of the tape were wrapped around the device. Finally, the ulnar artery under the penguin flipper was located by touch and the sensor head detector was centred over the artery. A smaller piece of tape, approximately 10cm x 1.5cm, was used to fix the sensor head in this position. If necessary, another small piece of tape was used to fix any excess interconnecting cable to the back of the penguin. A small fold was placed on the outermost flap of each piece of tape so that removal of the device was easier and faster.

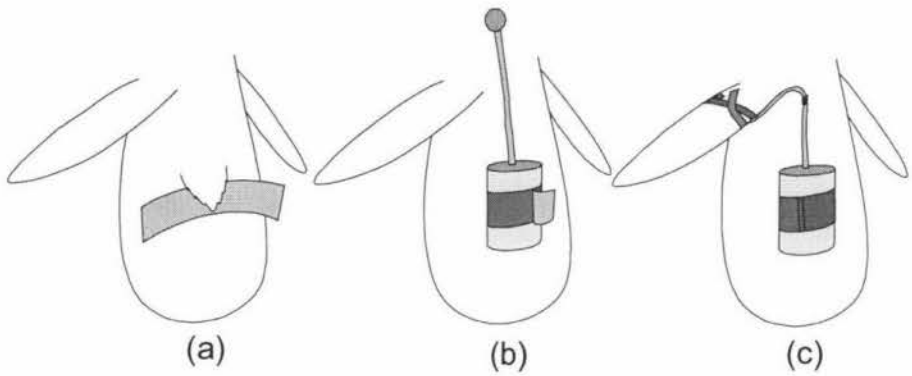


Figure 4.14: Attachment of the device using tape. (a) A patch of feathers were lifted from the bird's back and tape was applied to the underside, adhesive side facing outward. (b) The device was centred over the patch of feathers and held in place by the tape. (c) The sensor head was located over the ulnar artery under the flipper and also held in place with tape.

4.2.2 PHYSICAL RESULTS AND OBSERVATIONS

Apart from the biological data obtained, important physical observations were also made. Most importantly, the instrumented penguins indicated no obvious signs of stress as a result of wearing the device. Usual behaviour such as displaying, eating snow and stone collecting all continued without impediment and in one particular case, a female penguin was seen to mate while fitted with the device.

Observation of the penguin after release from both attachment and removal of the device showed only short-term unsettled behaviour. Within 5 to 10 minutes the activities of the equipped or recently released birds were indistinguishable from those of other birds.

Incidents that occurred in two experiments confirmed the importance of the colour of the device and the forgiving nature of the attachment technique. In the first instance the black paint on the interconnecting cable had been partially removed by tape and regions of the grey cable were visible. The penguin, fitted with this device, pecked and tugged at the cable whereas, in previous experiments, the cable was untouched. This experiment was ended early to avoid damage to the device and unnecessary discomfort for the penguin. In a second experiment, the penguin displayed discomfort that was due, perhaps, to the weight of the device being loaded unevenly on the group of feathers. This resulted in the bird actually removing the device before it could be recaptured indicating that, if necessary, the penguin was able to free itself from the device.

4.2.3 BIOLOGICAL RESPONSES

The majority of experiments performed were designed to capture physiological data as the penguins experienced changes in environmental temperature. This goal was extended to include the biological responses to stress as no significant temperature changes occurred during the experimentation period (the still air temperature range was approximately –7°C to 5°C).

Initial experiments indicated that the light scattering signal strength was much weaker for penguins than it had been in human tests. This was expected due to the feathers, thicker skin and thicker subcutaneous fat layer that penguins have. To compensate, the gain of the amplifier after the synchronous detector was increased but in general, the data collected suffered greater noise than that experienced during developmental testing. A major factor contributing to noise was movement of the sensor head against the flipper. Placement of the sensor head was also important and in cases where the sensor head shifted or was placed incorrectly, useful data were not obtained. Other factors that contributed to poor data were the security with which the sensor head was attached to the flipper and temperature. On the coldest days, the data showed little sign of change resulting from behavioural activities, which could be due to vasoconstriction whereby blood flow to the flipper was restricted but further experiments are required confirm this.

An example of data collected from the device is given in figure 4.15 showing two events that caused observable unsettled behaviour by the penguin. At 122 minutes the penguin was approached causing the bird to stop preening and to become agitated by our proximity. For approximately five minutes an effort was made to have the penguin constantly aware of our presence but not to cause the bird to run. Signs of stress in Adélie penguins are a 'glaring' look, where the whites of the eyes are visible, and a rolling head movement. This behaviour was displayed throughout the five minute period. At approximately 177 minutes a helicopter flew over the colony and signs of stress were displayed by many of the penguins.

Figure 4.15 shows the measured intensities for each wavelength over the period of the two stressful events. Signal level changes occurred at the two events but were related only in part to the induced stress. Movements of the bird caused the sensor head to move that resulted in intensity level changes in the data. Figures 4.16, 4.17 and 4.18 are the blood oxygenation, cytochrome oxidase saturation and blood volume data calculated from this data set respectively.

These figures and figure 4.19 contain a signal labelled "Behavioural activity" that was obtained by estimation of the penguins observed energy expenditure. Values were assigned to the bird's various behaviours such that high-energy activities corresponded to high signal level. Typical values were 1 for standing, 0.8 for lying down, 1.2 for stone collecting and 1.5 for Skua threat. A moving average of these data was taken to give the behavioural activity signals shown.

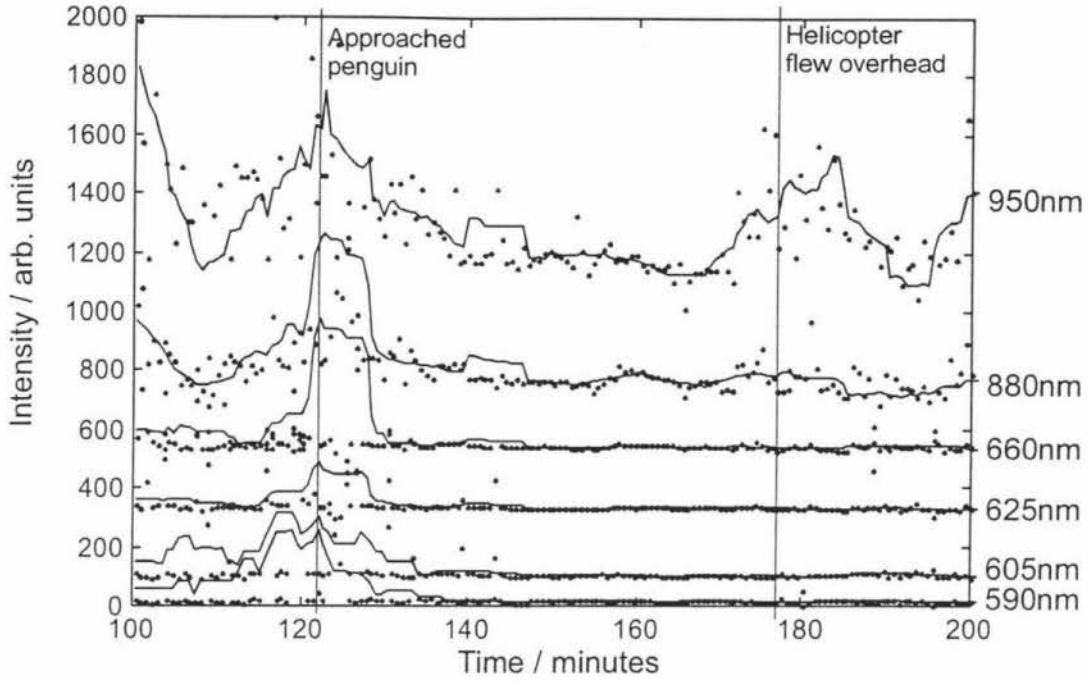


Figure 4.15: Data collected during two potentially stressful events for an Adélie penguin. At 122 minutes the penguin was approached causing the bird to display the typical signs of stress. This was maintained for approximately 5 minutes. Peaks in the data at this time may be related to these stressful events but may also be the result of sensor head movement.

The double wavelength measurement in figure 4.16 shows decreased blood oxygenation at 122 minutes that coincides with the first stressful event. A lower blood oxygenation implies that the demand for oxygen had increased while the supply had not, that is, increased metabolic rate was a response to stress. At 122 minutes a small change in the triple wavelength measurement was also seen, however after approximately 135 minutes the signal became erratic due, most likely, to movement of the sensor head.

Changes in the cytochrome oxidase signals for both the double and triple wavelength measurements (figure 4.17) did not show responses obviously related to the two stressful events. After the suspected sensor head movement at 135 minutes, both signals became approximately constant. Given the sensitivity of the cytochrome oxidase measurements made during developmental testing, it is most likely that fluctuations in these signals were due to movement of the sensor head.

The blood volume data (figure 4.18) show a clear drop in blood volume during the first stressful event that returns to the original level approximately 5 minutes later. The location of the sensor head meant that the blood flowing in to and out of the flipper was measured. As a decrease in blood volume was measured, this response indicates that stress caused less blood to flow to the extremities.

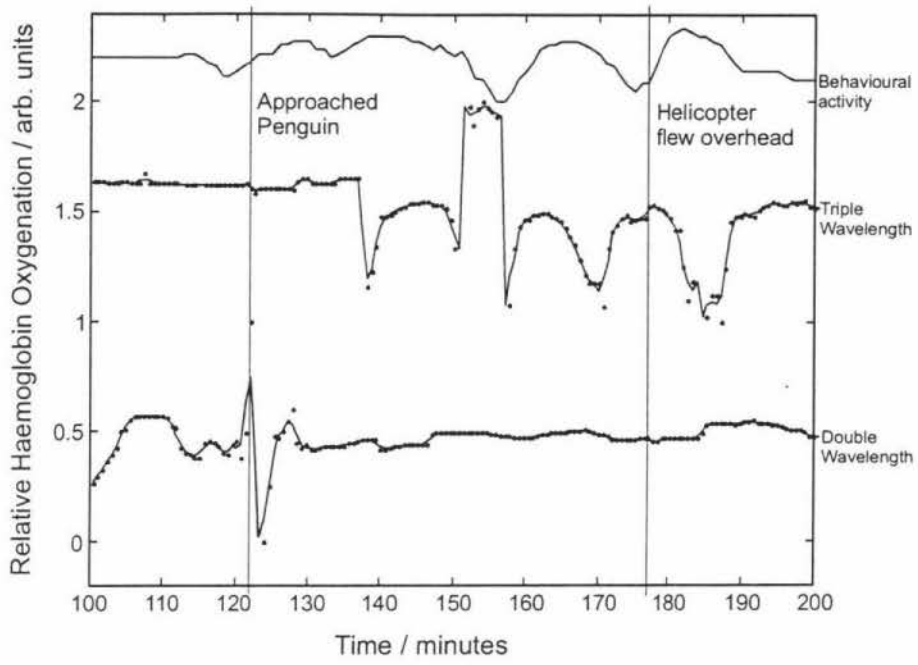


Figure 4.16: Relative blood oxygen saturation calculated from the data in figure 4.15. At 122 minutes, the decline in the double wavelength measurement and the subtle drop in the triple wavelength measurement imply a rise in metabolic rate not matched by an increase in oxygen supply as a result of stress. A shift of the sensor head is the probable reason for the erratic activity of the triple wavelength measurement after approximately 135 minutes. The data labelled 'behavioural activity' are an estimate of the energy expenditure during the various activities of the penguin made by observation.

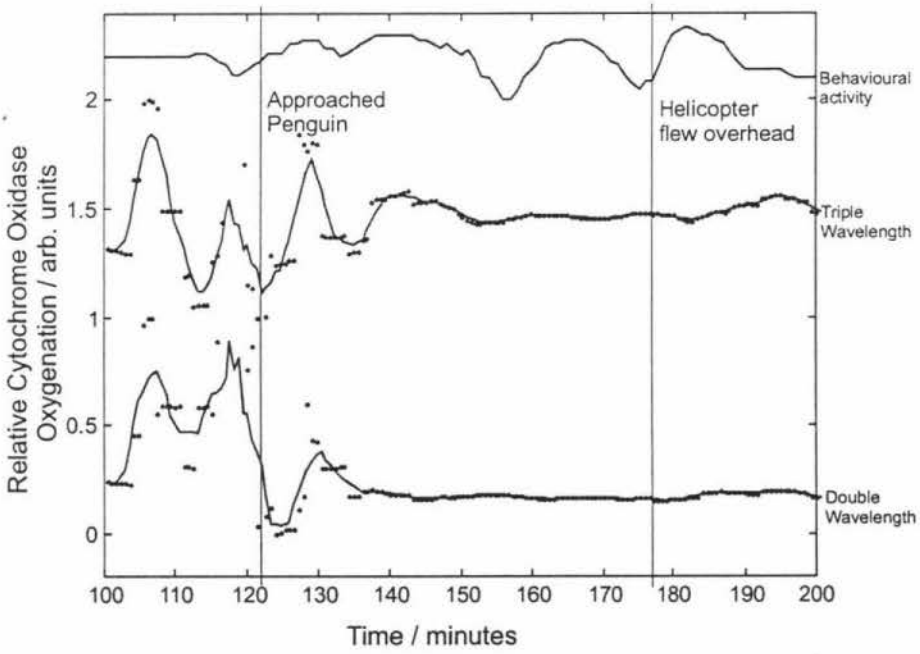


Figure 4.17: Relative oxygenation state of cytochrome oxidase calculated from the data in figure 4.15. While there is agreement between the double and triple wavelength measurements there is no obvious change as a result of stress. Given the sensitivity of the cytochrome oxidase measurements made during developmental testing these results at approximately 122 minutes are most likely due to movement of the sensor head.

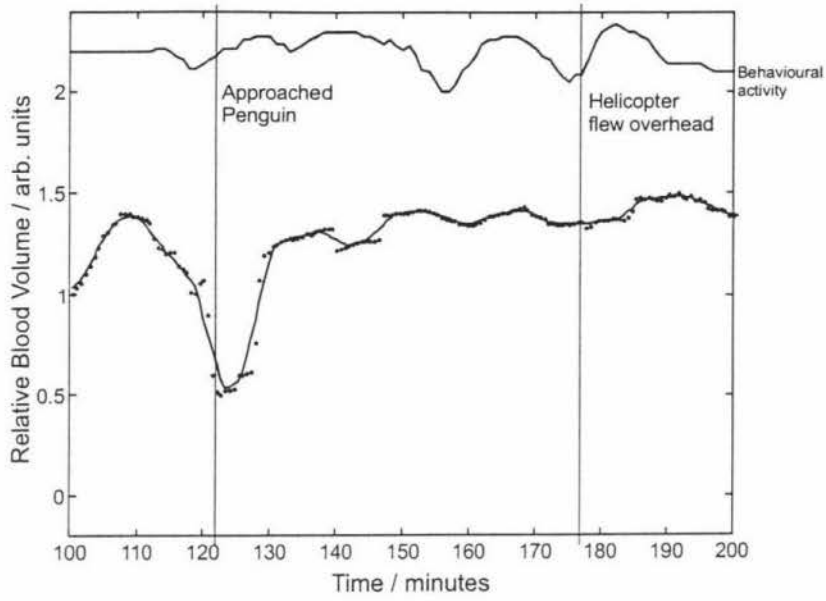


Figure 4.18: Relative blood volume calculated from the data in figure 4.15. The fall in blood volume at 122 minutes implies that the penguin's response to stress was reduced blood flow to the extremities that returned to normal approximately 5 minutes later.

Sensor head movement and noise made measurement of the pulse rate nearly impossible. All of the pulse data acquired had large errors and did not display obvious trends corresponding to the changes in behaviour. An experiment to collect only the pulse waveform by running one LED continually was made and the results indicated that, for the majority of the experiment, the periodicity of the waveform was hidden in noise. In attempts to improve the pulse data, the LED wavelengths 880, 660 and 625 were tried but none performed noticeably better. Figure 4.19 is the pulse rate data collected during the same stress experiment as above.

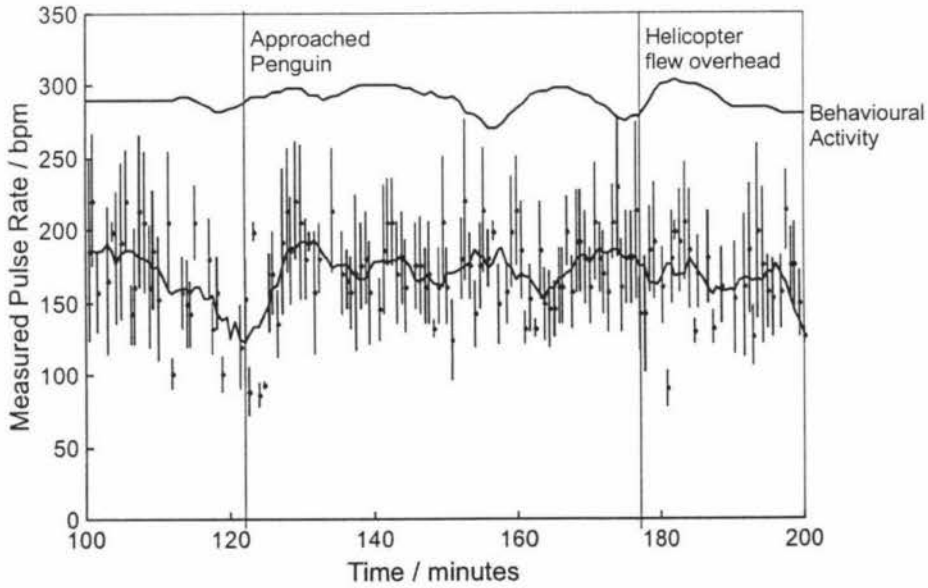


Figure 4.19: Pulse rate data acquired during the two potentially stressful events for an Adélie penguin. A rise in the average pulse rate occurs at approximately 122 minutes but is unlikely to have been caused by the first stressful event.

Chapter 5

Conclusion

Developing an instrument of biological and medical potential, and to demonstrate that potential by gaining physiological results obtained in an experimentally unique manner: through non-invasive measurements made on the Adélie penguin in its natural habitat, were the objectives of this thesis. Achieving this goal drew on a broad range of disciplines from biology, software engineering, signal processing and circuit design through to Antarctic field training. As a result, a system was developed with the ability to measure haemoglobin oxygenation, the reduction state of cytochrome oxidase, blood volume and pulse rate. The device had the control mechanisms necessary for a varying range of experiments and met the weight and size limitations necessary for working with the Adélie penguin.

The original question of how an Adélie penguin copes with the fluctuating Antarctic weather conditions was extended to ask, "what are the metabolic responses related to stress". Weather conditions that remained mild during the experimentation period and physical factors such as movement of the sensor head and weak signal strength prevented conclusive answers to these questions. However, signs of metabolic changes were present in the data allowing tentative predictions of the metabolic state to be made. An observed response to stress was the reduction in haemoglobin oxygenation implying an increased metabolic rate and a decrease in blood volume suggesting reduced blood flow to the flipper. Other observations included declining blood oxygenation as the bird recovered from capture and low, constant signal levels on cold days possibly due to vasoconstriction.

Although no statements could be made about the metabolic responses of the Adélie penguin the success of the device during developmental testing confirmed its potential. Under controlled testing conditions, the device was able to display changes in heart rate, blood oxygenation and blood volume as a result of physical activity and features of the pulse waveform may even have application in indicating heart conditions. Since obtaining metabolic data was possible with this first version of the device it is very likely that with modification and improvement, reliable data could be collected in the field.

5.1 Evaluation

As with any prototype system, experimentation usually reveals a number of potential improvements and modifications that could be made to improve the system's performance. In this case, a major limitation of the device was the crystal frequency. As a way of reducing power consumption and therefore extending the operating time of the device, a 2MHz crystal was used which meant that clock cycles within the Motorola chip occurred every 2μs. Modulating the LEDs at 1kHz required an interrupt generated every 500μs or, equivalently, every 250 clock cycles. The interrupt service routine used approximately 100 clock cycles to switch the state of the LEDs. Therefore, execution of the main program took place in the remaining 150

clock cycles. This meant that 40% of the available processing time was required for LED modulation. With hindsight, this problem could have been significantly reduced by selecting a lower modulation frequency. With more clock cycles available to the main program and sampling interrupt, greater time resolution could have been achieved over the measurement interval. Reducing the clock frequency did conserve power, however, fieldwork showed that long term measurements were impractical and hence power consumption, in practice, was not a limiting factor.

The most obvious limiting factor for signal acquisition is the efficiency of the sensor head and substantial improvement would come, most simply, from improvements made to this part of the device. Mendelson [7] suggests a number of techniques to improve the signal to noise ratio of acquired signals. These include, increasing the photodiode/LED separation and LED current, heating the local skin temperature to between 34°C and 45°C, and using an inverse sensor head design whereby several photodiodes surround the LED sources thus increasing the active area of the detector.

Of the techniques suggested by Mendelson, increasing the LED current and inverting the arrangement of the source and detector are probably the most suitable. At further cost to the operating time of the device, increasing the LED current would strengthen the acquired signal. However, the main benefit of increasing the LED current is obtained when the source/detector spacing is also increased and since the sensor head size is limited by the size of the penguin flipper, only minor signal improvement would be achieved in this way. Local heating of the skin to between 34°C and 45°C by the device is not a viable option. Even moderate heating would dramatically reduce the operating time of the device and would almost certainly increase the penguin's awareness of the device resulting in unrealistic stress data.

Significant improvement might be achieved by inverting the arrangement of the source and detectors on the sensor head. Since light diffuses radially from the source, surrounding the LEDs with photodiode detectors will collect more of the incident light (figure 5.1) and therefore increase the signal strength.

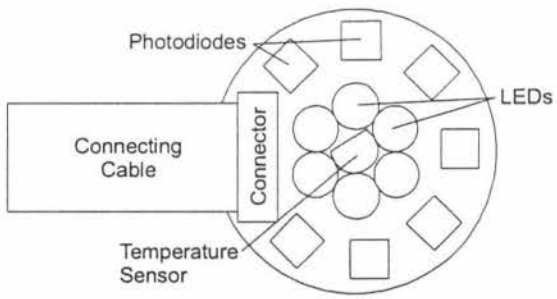


Figure 5.1: An alternative layout for the sensor head that may give improved signal strength. Advantages of this design are that the effective active area of the detector is increased and, since light propagates radially outward from the source, collection of the source light is more efficient.

From a software usability perspective, the interaction between conversion programs, terminal software and the displaying of results is complicated. Script files must be converted by two separate programs and then uploaded to the device using TEmu. When results are available, another program is used to convert the hexadecimal output into a format that may be interpreted by Microsoft Excel. Integration of these software components into one package that allowed the operator to create and upload script files then download and view the results would substantially improve usability of the system.

5.2 Future Development

Aside from these suggested enhancements the basic success of the device indicates the value of further development of this system. The first step toward future study of the Adélie penguin is to broaden the scope of possible experimentation. Currently package design, attachment technique and waterproofing limit experiments to short term investigations made on nesting penguins. Improvement in these areas could allow investigation into the metabolic response of penguins during the transition between land and sea, could determine the effects of moulting on the penguin's metabolism and show the metabolic trends over the fasting period. Each of these situations require a system capable of long term study so a review of power management along with the physical aspects of the device is necessary.

An enhancement with obvious potential is to incorporate telemetry into the operation of the device. Real time communication would allow detailed metabolic data to be transmitted during periods of unique behaviour and operating time could be extended by incorporating standby mode functionality into the communication.

The greatest potential of this system is that the non-invasive monitoring of metabolic activity can be extended to many other species, including humans. Interesting and useful biological information could be obtained in scenarios as variant as training athletes, race horses or predatory animals. While experiments of this nature are somewhat distant, the results of this work demonstrate the value of portable NIR spectroscopic devices beyond the medical profession and that continued research into such systems is both valuable and worthwhile.

Appendix A - Derivations

A.1 - Oxygen saturation derived from double wavelength measurements

The Beer-Lambert law is the linear relationship between absorbance and the concentration of an absorbing species [9]

$$A = \sum_i \varepsilon_\lambda^{X_i} [X_i] L \tag{A1.1}$$

where A is the measured absorbance, $\varepsilon_\lambda^{X_i}$ are the wavelength dependent absorption coefficients for the absorbers X_i, $[X_i]$ are the concentrations of the absorbers and L is the optical path length. The measured light scattering signal, R, is defined as,

$$R = \frac{I}{I_0} \tag{A1.2}$$

where I and I_0 are the measured and incident light intensities respectively. The relationship between R and A is

$$A = -\log R = -\log(I/I_0) = \log(I_0/I) \tag{A1.3}$$

For a particular chromophore, X, two wavelengths are selected; one for which the absorption due to oxidised and reduced forms of the chromophore are different and the other where they are approximately equal. If the contribution to the absorbance at these wavelengths by other absorbers is approximately constant then the Beer-Lambert law at these wavelengths is,

$$A_\lambda \propto \left(\varepsilon_\lambda^X [X] + \varepsilon_\lambda^{XO} [XO]\right) L \tag{A1.4}$$

For a medium that is semi-transparent in an optical range (e.g. red to near infrared), the concentration of scatterers is much greater than the concentration of absorbers. These media have the property that the mean optical path length is approximately constant for wavelengths within this range. Therefore, the optical path length term may be eliminated by taking the ratio of absorbances [8],

$$\frac{A_{\lambda_1}}{A_{\lambda_2}} = \frac{\varepsilon_{\lambda_1}^X [X] + \varepsilon_{\lambda_1}^{XO} [XO]}{\varepsilon_{\lambda_2}^X [X] + \varepsilon_{\lambda_2}^{XO} [XO]} \tag{A1.5}$$

In a closed system, the total concentration $[X_{total}]$ is equal to the sum of $[X]$ and $[XO]$. This allows (A1.5) to be rewritten as

$$\frac{A_{\lambda_1}}{A_{\lambda_2}} = \frac{\varepsilon_{\lambda_1}^{X}([X_{total}]-[XO])+\varepsilon_{\lambda_1}^{XO}[XO]}{\varepsilon_{\lambda_2}^{X}([X_{total}]-[XO])+\varepsilon_{\lambda_2}^{XO}[XO]} \tag{A1.6}$$

$$\frac{A_{\lambda_1}}{A_{\lambda_2}} = \frac{[X_{total}]\varepsilon_{\lambda_1}^{X}+[XO](\varepsilon_{\lambda_1}^{XO}-\varepsilon_{\lambda_1}^{X})}{[X_{total}]\varepsilon_{\lambda_2}^{X}+[XO](\varepsilon_{\lambda_2}^{XO}-\varepsilon_{\lambda_2}^{X})}$$

$$A_{\lambda_1}([X_{total}]\varepsilon_{\lambda_2}^{X}+[XO](\varepsilon_{\lambda_2}^{XO}-\varepsilon_{\lambda_2}^{X}))=A_{\lambda_2}([X_{total}]\varepsilon_{\lambda_1}^{X}+[XO](\varepsilon_{\lambda_1}^{XO}-\varepsilon_{\lambda_1}^{X}))$$

$$[X_{total}](A_{\lambda_1}\varepsilon_{\lambda_2}^{X}-A_{\lambda_2}\varepsilon_{\lambda_1}^{X})=[XO](A_{\lambda_2}(\varepsilon_{\lambda_1}^{XO}-\varepsilon_{\lambda_1}^{X})-A_{\lambda_1}(\varepsilon_{\lambda_2}^{XO}-\varepsilon_{\lambda_2}^{X}))$$

$$\frac{[XO]}{[X_{total}]} = \frac{A_{\lambda_1}\varepsilon_{\lambda_2}^{X}-A_{\lambda_2}\varepsilon_{\lambda_1}^{X}}{A_{\lambda_2}(\varepsilon_{\lambda_1}^{XO}-\varepsilon_{\lambda_1}^{X})-A_{\lambda_1}(\varepsilon_{\lambda_2}^{XO}-\varepsilon_{\lambda_2}^{X})}$$

$$\frac{[XO]}{[X_{total}]} = \frac{(A_{\lambda_1}/A_{\lambda_2})\varepsilon_{\lambda_2}^{X}-\varepsilon_{\lambda_1}^{X}}{(A_{\lambda_1}/A_{\lambda_2})(\varepsilon_{\lambda_2}^{X}-\varepsilon_{\lambda_2}^{XO})+(\varepsilon_{\lambda_1}^{XO}-\varepsilon_{\lambda_1}^{X})}$$

Using equations (A1.3) and (A1.4), the relationship between measured intensity the oxygen saturation is found,

$$\frac{[XO]}{[X_{total}]} = \frac{\varepsilon_{\lambda_2}^{X}\dfrac{\log(I_{0,\lambda_1}/I_{\lambda_1})}{\log(I_{0,\lambda_2}/I_{\lambda_2})}-\varepsilon_{\lambda_1}^{X}}{(\varepsilon_{\lambda_2}^{X}-\varepsilon_{\lambda_2}^{XO})\dfrac{\log(I_{0,\lambda_1}/I_{\lambda_1})}{\log(I_{0,\lambda_2}/I_{\lambda_2})}+\varepsilon_{\lambda_1}^{XO}-\varepsilon_{\lambda_1}^{X}} \tag{A1.7}$$

A.2 - Oxygen saturation derived from triple wavelength measurements

The purpose of calculating oxygen saturation using three wavelengths is to remove baseline drift. An estimate of background absorbance is found by linearly interpolating between the absorbances at two other wavelengths (λ_1 and λ_3, figure A2.1). Each wavelength is selected such that the absorbances due to the oxidised and reduced forms of the chromophore are approximately equal. By interpolating between λ_1 and λ_3 a background absorbance, A_2^* is found that is compared with the measured absorbance, A_2. The interpolation estimates the drift occurring at λ_2 and by comparing A_2 to A_2^* the drift is removed.

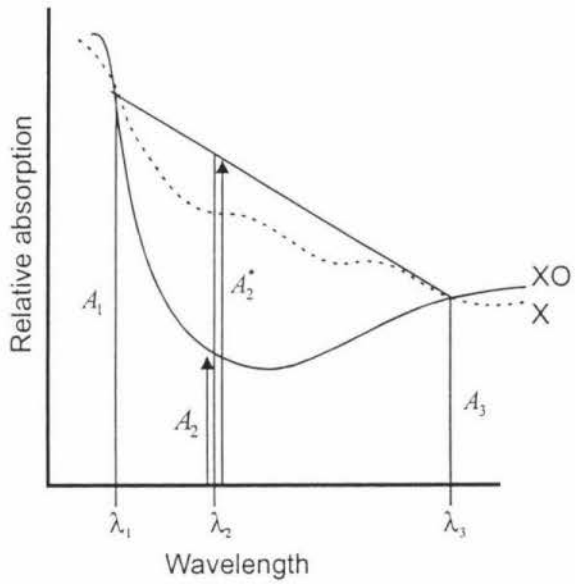


Figure A2.1: The use of three wavelengths to remove baseline drift in oxygen saturation measurements. By interpolating between λ_1 and λ_3 a background absorbance, A_2^* is found for comparison with A_2.

The value A_2^* at λ_2 is given by

$$A_2^* = \frac{A_3 - A_1}{\Lambda} + A_1 \quad \text{where} \quad \Lambda = \frac{\lambda_3 - \lambda_1}{\lambda_2 - \lambda_1} \tag{A2.1}$$

By analogy with equation (A1.5), β is defined as

$$\begin{aligned} \beta = \frac{A_2}{A_2^*} &= \frac{\Lambda A_2}{A_3 + A_1(\Lambda - 1)} \\ &= \frac{\Lambda\left(\varepsilon_{\lambda_2}^{X}[X] + \varepsilon_{\lambda_2}^{XO}[XO]\right)}{\left(\varepsilon_{\lambda_3}^{X}[X] + \varepsilon_{\lambda_3}^{XO}[XO]\right) + \left(\varepsilon_{\lambda_1}^{X}[X] + \varepsilon_{\lambda_1}^{XO}[XO]\right)(\Lambda - 1)} \end{aligned} \tag{A2.2}$$

Again, for a closed system, the total concentration $[X_{total}]$ is equal to the sum of $[X]$ and $[XO]$, therefore (A2.2) may be rewritten as

$$\beta = \frac{\Lambda\left(\varepsilon_{\lambda_2}^{X}\left([X_{total}]-[XO]\right)+\varepsilon_{\lambda_2}^{XO}[XO]\right)}{\left(\varepsilon_{\lambda_3}^{X}\left([X_{total}]-[XO]\right)+\varepsilon_{\lambda_3}^{XO}[XO]\right)+\left(\varepsilon_{\lambda_1}^{X}\left([X_{total}]-[XO]\right)+\varepsilon_{\lambda_1}^{XO}[XO]\right)(\Lambda-1)}$$

$$\beta = \frac{\Lambda\left([X_{total}]\varepsilon_{\lambda_2}^{X}+[XO]\left(\varepsilon_{\lambda_2}^{XO}-\varepsilon_{\lambda_2}^{X}\right)\right)}{\left([X_{total}]\varepsilon_{\lambda_3}^{X}+[XO]\left(\varepsilon_{\lambda_3}^{XO}-\varepsilon_{\lambda_3}^{X}\right)\right)+\left([X_{total}]\varepsilon_{\lambda_1}^{X}+[XO]\left(\varepsilon_{\lambda_1}^{XO}-\varepsilon_{\lambda_1}^{X}\right)\right)(\Lambda-1)}$$

$$\begin{aligned}&[X_{total}]\left(\beta\left(\varepsilon_{\lambda_3}^{X}+(\Lambda-1)\varepsilon_{\lambda_1}^{X}\right)-\Lambda\varepsilon_{\lambda_2}^{X}\right)\\&=[XO]\left(\Lambda\left(\varepsilon_{\lambda_2}^{XO}-\varepsilon_{\lambda_2}^{X}\right)+\beta\left((\Lambda-1)\left(\varepsilon_{\lambda_1}^{X}-\varepsilon_{\lambda_1}^{XO}\right)+\varepsilon_{\lambda_3}^{X}-\varepsilon_{\lambda_3}^{XO}\right)\right)\end{aligned}$$

$$\frac{[XO]}{[X_{total}]} = \frac{\beta\left(\varepsilon_{\lambda_3}^{X}+(\Lambda-1)\varepsilon_{\lambda_1}^{X}\right)-\Lambda\varepsilon_{\lambda_2}^{X}}{\Lambda\left(\varepsilon_{\lambda_2}^{XO}-\varepsilon_{\lambda_2}^{X}\right)+\beta\left((\Lambda-1)\left(\varepsilon_{\lambda_1}^{X}-\varepsilon_{\lambda_1}^{XO}\right)+\varepsilon_{\lambda_3}^{X}-\varepsilon_{\lambda_3}^{XO}\right)} \qquad \text{(A2.3)}$$

The direct relationship between measured intensity and oxygen saturation is found by substituting β into (A2.3). Using equations (A1.3), (A1.4) and (A2.2), β is

$$\frac{\Lambda\log\left(I_{0,\lambda_2}/I_{\lambda_2}\right)}{\log\left(I_{0,\lambda_3}/I_{\lambda_3}\right)+\log\left(I_{0,\lambda_1}/I_{\lambda_1}\right)(\Lambda-1)} \qquad \text{(A2.4)}$$

A.3 – Relative blood volume derived from double wavelength measurements

A relationship describing relative blood volume is derived from the absorbance equation (A1.4) by considering the absorbance at two different wavelengths

$$A_{\lambda_1} \propto \left(\varepsilon_{\lambda_1}^{X}[X]+\varepsilon_{\lambda_1}^{XO}[XO]\right)L \qquad \text{(A3.1)}$$

$$A_{\lambda_2} \propto \left(\varepsilon_{\lambda_2}^{X}[X]+\varepsilon_{\lambda_2}^{XO}[XO]\right)L \qquad \text{(A3.2)}$$

By rearranging (A3.1) and substituting into (A3.2) two equations describing $[X]$ and $[XO]$ in terms of the absorbance are found. Rearranging (A3.1) gives

$$[X] \propto \frac{A_{\lambda_1}-\varepsilon_{\lambda_1}^{XO}[XO]L}{\varepsilon_{\lambda_1}^{X}L} \text{ and } [XO] \propto \frac{A_{\lambda_1}-\varepsilon_{\lambda_1}^{X}[X]L}{\varepsilon_{\lambda_1}^{XO}L}$$

Substituting into (A3.2) gives

$$A_{\lambda_2} \propto \frac{A_{\lambda_1}\varepsilon^{XO}_{\lambda_2} + \varepsilon^{X}_{\lambda_2}\varepsilon^{XO}_{\lambda_1}[X]L - \varepsilon^{X}_{\lambda_1}\varepsilon^{XO}_{\lambda_2}[X]L}{\varepsilon^{XO}_{\lambda_1}}$$

and

$$A_{\lambda_2} \propto \frac{A_{\lambda_1}\varepsilon^{X}_{\lambda_2} + \varepsilon^{XO}_{\lambda_2}\varepsilon^{X}_{\lambda_1}[XO]L - \varepsilon^{XO}_{\lambda_1}\varepsilon^{X}_{\lambda_2}[XO]L}{\varepsilon^{X}_{\lambda_1}}$$

$$[X] \propto \frac{1}{L}\left(\frac{A_{\lambda_2}\varepsilon^{XO}_{\lambda_1} - A_{\lambda_1}\varepsilon^{XO}_{\lambda_2}}{\varepsilon^{X}_{\lambda_2}\varepsilon^{XO}_{\lambda_1} - \varepsilon^{X}_{\lambda_1}\varepsilon^{XO}_{\lambda_2}}\right) \tag{A3.3}$$

and

$$[XO] \propto \frac{1}{L}\left(\frac{A_{\lambda_2}\varepsilon^{X}_{\lambda_1} - A_{\lambda_1}\varepsilon^{X}_{\lambda_2}}{\varepsilon^{XO}_{\lambda_2}\varepsilon^{X}_{\lambda_1} - \varepsilon^{XO}_{\lambda_1}\varepsilon^{X}_{\lambda_2}}\right) \tag{A3.4}$$

Once again, since the total concentration $[X_{total}]$ is equal to the sum of $[X]$ and $[XO]$, combining (A3.3) and (A3.4) gives the blood volume relationship, (A3.5).

$$[X_{total}] = [X] + [XO] \propto \frac{1}{L}\left(\frac{A_{\lambda_2}\varepsilon^{XO}_{\lambda_1} - A_{\lambda_1}\varepsilon^{XO}_{\lambda_2}}{\varepsilon^{X}_{\lambda_2}\varepsilon^{XO}_{\lambda_1} - \varepsilon^{X}_{\lambda_1}\varepsilon^{XO}_{\lambda_2}} + \frac{A_{\lambda_2}\varepsilon^{X}_{\lambda_1} - A_{\lambda_1}\varepsilon^{X}_{\lambda_2}}{\varepsilon^{XO}_{\lambda_2}\varepsilon^{X}_{\lambda_1} - \varepsilon^{XO}_{\lambda_1}\varepsilon^{X}_{\lambda_2}}\right)$$

$$[X_{total}] \propto \frac{1}{L}\left(\frac{A_{\lambda_1}\left(\varepsilon^{XO}_{\lambda_2} - \varepsilon^{X}_{\lambda_2}\right) + A_{\lambda_2}\left(\varepsilon^{X}_{\lambda_1} - \varepsilon^{XO}_{\lambda_1}\right)}{\varepsilon^{X}_{\lambda_1}\varepsilon^{XO}_{\lambda_2} - \varepsilon^{X}_{\lambda_2}\varepsilon^{XO}_{\lambda_1}}\right) \tag{A3.5}$$

Appendix B – Simulation Programs

B.1 – Synchronous detector numerical solution

For the synchronous detector frequency response given in figure 2.10, a modulation frequency of 1kHz and a low-pass filter corner frequency of 1Hz was used. Over a frequency range from 500Hz to 4kHz, 3500 points were calculated. At each frequency, a complete period of the detector output signal (10000 point resolution) including the effect of the low-pass filter was evaluated to n=51 iterations of equation 2.9.

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cmath>

using namespace std;

// Globals
double const   PI = 3.1415926535897932;
int const      frequency_buffer_size        = 3500;     // 3500 point resolution for final data
int const      detector_output_buffer_size  = 10000;    // 10000 point resolution for each detector output
double         dc_input_voltage             = 1.0;      // V
double         modulation_frequency         = 1000.0;   // Hz
double         start_frequency              = 500.0;    // Hz
double         finish_frequency             = 4000.0;   // Hz
double         lpf_corner_frequency         = 1.0;      // Hz
double         frequency_stepsize           = (finish_frequency - start_frequency) / frequency_buffer_size;
int            fourier_iterations           = 51;

// Globals calculated from the globals above
double const   TwoPI                        = 2*PI;
double         modulation_freq              = TwoPI*modulation_frequency;     // rad/s
double         modulation_period            = 1 / modulation_frequency;       // seconds
double         start_freq                   = TwoPI*start_frequency;          // rad/s
double         finish_freq                  = TwoPI*finish_frequency;         // rad/s
double         lpf_corner_freq              = TwoPI*lpf_corner_frequency;     // rad/s

// Allocate space for the frequency data
double         output[frequency_buffer_size][2];
double         detector_output[detector_output_buffer_size];

// Function Prototypes
double         LowPassFilter( double Vin, double frequency );
void           DetectorSignalOut( double frequency );
double         rms( double* buffer, int const buffer_length );

int main(void)
{
        // The output file containing data
        ofstream outfile("SyncDetData.txt");

        // Initialise the buffers to zero
        memset( output, 0, sizeof(output) );
        memset( detector_output, 0, sizeof(detector_output) );

        // Main calculation loop
        unsigned int index;
        double frequency = start_frequency;
        for( index = 0; index < frequency_buffer_size; ++index, frequency += frequency_stepsize )
        {
                output[index][0] = frequency;
                DetectorSignalOut( frequency );
                output[index][1] = rms( detector_output, detector_output_buffer_size );

                outfile << fixed << setprecision(7) << output[index][0] << "\t" << output[index][1] << endl;
                cout << fixed << setprecision(7) << output[index][0] << "\t" << output[index][1] << endl;
        }
```

```
        return 0;
}

void DetectorSignalOut( double frequency )
{
        double w = TwoPI*frequency;

        // Calculate the time required for a complete period of the product of
        // 'frequency' and the modulation frequency. If 'frequency' is really
        // close to the modulation frequency assume they are the same
        double finish_time    = modulation_period;
        double beat_freq      = abs(modulation_freq - w);
        if( beat_freq > 1 )
                finish_time = TwoPI / beat_freq;

        double time_step = finish_time / detector_output_buffer_size;

        // Calculate the output from the detector
        int n, index;
        double output, upper_freq, lower_freq, upper_component, lower_component;
        double time = 0;
        for( index = 0; index < detector_output_buffer_size; ++index, time += time_step )
        {
                output = 0;
                for( n = 1; n <= fourier_iterations; n += 2 )
                {
                        lower_freq = ( w - n*modulation_freq ) / TwoPI;
                        upper_freq = ( w + n*modulation_freq ) / TwoPI;
                        lower_component = LowPassFilter( (2 * dc_input_voltage) / (n * PI), lower_freq );
                        upper_component = LowPassFilter( (2 * dc_input_voltage) / (n * PI), upper_freq );
                        output += (lower_component - upper_component);
                }
                detector_output[index] = output;
        }
}

//*****
// Finds Vout if Vin is passed to a simple low pass filter
// with the corner frequency specified
double LowPassFilter( double Vin, double frequency )
{
        double w = TwoPI*frequency;                     // Omega for the frequency of interest
        return Vin / sqrt( 1 + (w / lpf_corner_freq) * (w / lpf_corner_freq) );
}

//*****
// Finds the rms value of a signal within a buffer
double rms( double* buffer, int const buffer_length )
{
        double sum_of_squares = 0;
        for( int i = 0; i < buffer_length; ++i )
                sum_of_squares += (buffer[i] * buffer[i]);
        return sqrt( sum_of_squares / buffer_length );
}
```

B.2 – Pulse algorithm simulation

The pulse algorithm, written in assembly code, was recreated and simulated using Matlab where it was then used to investigate noise and drift limitations.

```
function [pulse,error] = Pulse(GPMedBufsize, GPPBufsize, GPThres, GPTO);
%
%       Simulates the pulse rate measurement algorithm and returns the period
%       and error estimate of a signal stored in a text file.
%       The file format is {time, signal, pure signal, noise signal)
%                          {time, signal, pure signal, noise signal)
%                          {time, signal, pure signal, noise signal)
%       Syntax:
%               [pulse error] = Pulse(medbuf,Pbuf,thres,TO)
%       Parameters:
%               medbuf = size of the median buffer to use
%               Pbuf   = size of the period buffer to use
```

```
%              thres  = limiting error threshold
%              TO     = timeout period before changing from median to upper
%                       quartile to lower quartile.
%       Example:
%          [Pulse,Error] = pulse(100,4,10,400);

% Initialise variables
data=load('C:\masters\sig.txt');      % File containing the waveform
datsize=length(data);                 % Waveform buffer length
Tempw=0;                              % Variables defined in pulse assembly code
Time=0;
GPTAmp=0;
GPAmp=0;
GPQuart=0;
GPToggle=0;
GPTime=0;
GPSWAvr=0;
GPSWOld=0;
GPSWBot=0;
GPSWTop=0;
GPPAvr=0;
GPErr=0;
GPMedBuf=zeros(1,GPMedBufsize);
GPMedBufptr=1;
GPSWBufsize=8;
GPSWBuf=zeros(1,GPSWBufsize);
GPSWBufptr=1;
GPPBuf=zeros(1,GPPBufsize);
GPPBufptr=1;
GPPeriod=0;
GPError=10000;

% Initialise intermediate graph buffers
Med=zeros(2,datsize);
Tog=zeros(2,datsize);
SW =zeros(2,datsize);
P=zeros(2,datsize);
E=zeros(2,datsize);

%Find the period
for i=1:datsize   %Main pulse measurement loop

   %Get a new Sample
   Tempw = GPTime;
   GPData=data(i,2);
   GPTime=data(i,1);
   Time = Time + (GPTime-Tempw);

   %Add data to the Median buffer
   GPMedBuf(GPMedBufptr)=GPData;
   GPMedBufptr=GPMedBufptr+1;
   if(GPMedBufptr==(GPMedBufsize+1))
      GPMedBufptr=1;
   end

   %Find the median, max and min using the correct quartile
   GPMax=max(GPMedBuf);
   GPMin=min(GPMedBuf);
   GPTAmp=GPMax-GPMin;

   %Decide which quartile to use
   switch GPQuart
   case 0,     %Median
      GPMed = (GPMax+GPMin)/2;
   case 1,     %Upper
      GPMed = (GPMax+GPMin)/2 + (GPMax-GPMin)/4;
   case 2,     %Lower
      GPMed = (GPMax+GPMin)/2 - (GPMax-GPMin)/4;
   end
Med(1,i)=Time;          % Collect data for intermediate graphs
Med(2,i)=GPMed;

   %Set the toggle value
   if ((GPToggle==0) & (GPData > GPMed))
      GPToggle=65535;
   elseif ((GPToggle==65535) & (GPData < GPMed))
```

```
      GPToggle=0;
   end

   %Generate the square wave using GPToggle
   GPSWBuf(GPSWBufptr)=GPToggle;
   GPSWBufptr=GPSWBufptr+1;
   if(GPSWBufptr==(GPSWBufsize+1))
      GPSWBufptr=1;
   end
Tog(1,i)=Time;          % Collect data for intermediate graphs
Tog(2,i)=GPToggle;

   %Apply smoothing to square wave to find SW averaged value
   GPSWOld=GPSWAvr;
   GPSWAvr=0;
   for j=1:GPSWBufsize
      GPSWAvr = GPSWAvr + GPSWBuf(j)/8;
   end
SW(1,i)=Time;           % Collect data for intermediate graphs
SW(2,i)=GPSWAvr;

   %Look for potential periods and store them in Pbuf
   if((GPSWOld < 0.1)&(GPSWAvr > 0.1))
      GPPBuf(GPPBufptr) = (GPTime-GPSWBot);
      GPPBufptr=GPPBufptr+1;
      if(GPPBufptr==(GPPBufsize+1))
         GPPBufptr=1;
      end
      GPSWBot = GPTime;
   end
   if((GPSWOld > 0.9)&(GPSWAvr < 0.9))
      GPPBuf(GPPBufptr) = (GPTime-GPSWTop);
      GPPBufptr=GPPBufptr+1;
      if(GPPBufptr==(GPPBufsize+1))
         GPPBufptr=1;
      end
      GPSWTop = GPTime;
    end

   %Check for constistant periods
   Buf=zeros(1,10);                    %create a pseudo memory buffer
   for(k=1:GPPBufsize)
      Buf(k)=GPPBuf(k);                %copy the period buffer to psuedo mem
   end
   outbuf=5;                           %pseudo temp buffer address
   inbuf=1;
   bufsize=GPPBufsize;
   while(bufsize~=1)
      k=0;j=0;
      while(k<bufsize)
         tempw = Buf(inbuf+k)/2;
         k=k+1;
         tempw = tempw + Buf(inbuf+k)/2;
         k=k+1;
         Buf(outbuf+j)=tempw;
         j=j+1;
      end
      bufsize=bufsize/2;
      inbuf=outbuf;
   end
   GPPAvr=Buf(outbuf);
P(1,i)=Time;            % Collect data for intermediate graphs
P(2,i)=GPPAvr;

   GPMax = max(GPPBuf);
   GPMin = min(GPPBuf);
   GPErr = (GPMax-GPMin)/2;
E(1,i)=Time;            % Collect data for intermediate graphs
E(2,i)=GPErr;

   if (GPErr < GPThres)

      %find the most accurate period
      if (GPErr < GPError)
         GPPeriod = GPPAvr;
         GPError = GPErr;
```

```
      end

   end

   %Change between median, upper and lower quartile
   GPQuart = fix(Time/GPTO);
   if(GPQuart == 3)
      i=datsize;
   end
end
pulse = GPPeriod;       % Return the period in the time units passed
error = GPError;        % in with data

% Display signal to noise ratio for this waveform
%       noiseRMS = rms(data(:,4))
%       SignalRMS= rms(data(:,3))
%       S_N=SignalRMS/noiseRMS
%
% Determine graph ranges
%       xmin=min(data(:,1));
%       xmax=max(data(:,1));
%       ymax=max(data(:,2));
%       ymin=min(data(:,2));
%       stepsize=(xmax-xmin)/datsize;
%
% Plot the input waveform with a line indicating the relative
% median buffer length
%       figure(1);
%       plot(data(:,1),data(:,2),'k');hold on;
%       axis([xmin,xmax,1900,2250]);
%       plot(Med(1,:),Med(2,:),'r');
%       line([xmin xmin+stepsize*GPMedBufsize],[2225 2225]);
%       hold off
%       zoom on;
%
% Plot the square wave, smoothed square wave, period and error
% estimates
%       figure(2);
%       plot(Tog(1,:),Tog(2,:)/20000,'g');hold on;
%       plot(SW(1,:),SW(2,:)/20000,'b');
%       plot(P(1,:),P(2,:),'r');
%       plot(E(1,:),E(2,:),'k');
%       axis([xmin,xmax,0,20]);
%       hold off;
%       zoom on;
```

B.3 – Input-signal drift limitations for the pulse algorithm

Input signal drift places a limit on the maximum length of the median buffer. If the input signal drifts above or below the median value then level crossings no longer occur and the period of the waveform can not be found. For this reason it is useful to know the ideal median buffer length that will accommodate the maximum amount of drift while still maintaining acceptable immunity to noise.

Computation time: 3 hours 21 minutes on a P233MHz computer.

```
function drift_test();
%
%       Investigates the effects of drift in the input signal
%       Results displayed using Drift_test_Showdata
%

% Initialise input signal
Period = 1;
Driftoffset = 0;
WaveBufferSize = 1000;
```

```
Xmin = 0;
Xmax = 10;

%Save results to a text file
outf=fopen('c:\masters\Drift.txt','w');
for( Amp = logspace(-1,1,10) )
   Amp
   % Investigate a range of drifts from 0 to 25
   for( Driftslope = 0:0.1:25 )
      Driftslope
      possible = 1;
      makewaves( Amp, Period, 0, Driftslope, Driftoffset, 0, 0, WaveBufferSize, Xmin,
Xmax );
      wave = load('C:\masters\sig.txt');            % File containing the waveform

      % Calculate actual size of the median buffer and create it
      Median_Buffer_Size = 0.1;       % Number of periods contained in the period
buffer
      GPMedBufsize = ceil(Median_Buffer_Size * WaveBufferSize * Period /(Xmax-Xmin));
      GPMedBuf = zeros(GPMedBufsize,1);

      % Search for the length of median buffer that makes period
      % measurement impossible
      while( possible & (Median_Buffer_Size <= 5))
         possible = PulsePossible(wave, WaveBufferSize, Xmax, Xmin, GPMedBuf,
GPMedBufsize);
         if( possible )
            Median_Buffer_Size = Median_Buffer_Size + 0.1;
            GPMedBufsize = ceil(Median_Buffer_Size * WaveBufferSize * Period /(Xmax-
Xmin));
            GPMedBuf = zeros(GPMedBufsize,1);
         end
      end

      % if we find the maximum length of the median buffer then record it
      if( ~possible )
         fprintf(outf,'%f %f %f\n',Amp,Median_Buffer_Size,Driftslope);
         if( Median_Buffer_Size == 0.1 )      % if failed for the first one then
            break;                            % all slopes greater than this will fail
         end                                  % as well, might as well quit now
      end
   end
end
fclose(outf);

function possible = PulsePossible(wave, WaveBufferSize, Xmax, Xmin, GPMedBuf,
GPMedBufsize);
%
% Use code from pulse.m to simulate the level crossing section of the
% pulse algorithm
% Local function for drift_test
%

% Number of consecutive crosses of the median
% Once greater than Number_of_crosses_needed assume period
% able to be calculated
crosses = 0;
Number_of_crosses_needed = 6;

GPMedBufptr = 1;
GPQuart = 0;
GPToggle = 0;
Med = 0;
for i=1:WaveBufferSize % Main pulse measurement loop

   %Get a new Sample
   GPData = wave(i,2);

   %Add data to the Median buffer
   GPMedBuf(GPMedBufptr) = GPData;
   GPMedBufptr = GPMedBufptr + 1;
   if( GPMedBufptr == (GPMedBufsize+1) )
      GPMedBufptr=1;
   end

   %Find the median, max and min using the correct quartile
```

```
   GPMax=max(GPMedBuf);
   GPMin=min(GPMedBuf);

   %Decide which quartile to use
   switch GPQuart
   case 0,      %Median
      GPMed = (GPMax+GPMin)/2;
   case 1,      %Upper
      GPMed = (GPMax+GPMin)/2 + (GPMax-GPMin)/4;
   case 2,      %Lower
      GPMed = (GPMax+GPMin)/2 - (GPMax-GPMin)/4;
   end
Med(i,1) = i*(Xmax-Xmin)/WaveBufferSize;
Med(i,2) = GPMed;              % Collect data for intermediate graphs

   %Set the toggle value
   if ((GPToggle==0) & (GPData > GPMed))
      GPToggle=65535;
      crosses = crosses + 1;
      if( crosses > Number_of_crosses_needed )
         break;
      end
   elseif ((GPToggle==65535) & (GPData < GPMed))
      GPToggle=0;
      crosses = crosses + 1;
      if( crosses > Number_of_crosses_needed )
         break;
      end
   end

   %Change between median, upper and lower quartile - try all three
   if( GPQuart ~= floor(3*i/WaveBufferSize) )
      GPQuart = floor(3*i/WaveBufferSize);
      crosses = 0;
   end
end
if( crosses > Number_of_crosses_needed )
   possible = 1;
   %display('Pulse possible');
   %plot(Med(:,1),Med(:,2));
else
   possible = 0;
   %display('Pulse not possible');
   %plot(Med(:,1),Med(:,2));
end

function Drift_test_Showdata();
%
%       Display the results of the Drift test for the pulse rate algorithm
%

% Load and initialise the data buffers
data=load('C:\masters\Drift_amp01_10.txt');
amp = data(:,1);

%Chop up data into separate buffers and display results;
figure(1);
clf;hold on;zoom on;
Amp = amp(1);
len = length(amp)
for( i = 1:len )
   med = 0;
   drift = 0;
   j = 1;
   while( (i <= len) & (amp(i) == Amp))
      med(j) = data(i,2);
      drift(j) = data(i,3);
      i = i + 1;
      j = j + 1;
   end
   if( i <= len )
      Amp = amp(i);
   end
   plot( med, drift, 'k');
end
```

B.4 – Period measurement limitation due to the digital filter

```
function Compare_test();
%
%       Performs a comparison between the ideal pulse and the pulse
%       measured by the simulated pulse algorithm for various input
%       signal frequencies
%       Results displayed using Compare_test_Showdata
%

% Initialise output buffers
bsize           = 100;                  % Range of frequencies to try
truepulses      = zeros(bsize,1);       % Buffer containing the ideal pulse values
measpulses      = zeros(bsize,1);       % Buffer containing the measured pulse values
errors          = zeros(bsize,1);       % Buffer containing the errors in the measured
values
freq            = 1;                    % Initial pulse rate

% Create sine waves and measure their periods
for(i=1:bsize)
   Makewaves(100,1/freq,0,0,0,0,2048,500,1,5);
   [measpulses(i),errors(i)] = pulse(100,4,10,400);
   truepulses(i) = 1 / freq;
   freq = freq + 0.1                    % Show where we're up to
end

% Save the data for plotting
outf=fopen('c:\masters\per.txt','w');
for(i=1:bsize)
   fprintf(outf,'%f %f %f\n',truepulses(i),measpulses(i),errors(i));
end
fclose(outf);

function Compare_test_Showdata();
%
%       Display the comparison between ideal and measured pulse
%       rate results for the pulse algorithm
%

% Load and initialise the data buffers
data            = load('C:\masters\per.txt');
truepulses      = data(:,1);
measpulses      = data(:,2);
errors          = data(:,3);

% Turn periods into frequencies
figure(2);
for(i=1:100)    % 100 is the buffer size in Compare_test
   truefreq(i)=1/truepulses(i);
   measfreq(i)=1/measpulses(i);
end

% Display the results
plot(truefreq,measfreq,'k');
axis([0 10 0 10]);
line([5.566 5.566],[0 5.566]);
```

B.5 – Input-signal noise limitations for the pulse algorithm

Noise susceptability is another important limitation of the algorithm. Quantifing the required level of signal to noise in the input signal describes when meaningful results can be expected from the algorithm. By measuring the period of sine and square waves with increasing levels of white noise, estimates of the required signal to noise ratio where obtained.

```
function SN_test_sine();
%
%       Determine the limiting signal to noise ratio for the
%       pulse measurement algorithm using sine wave input signals
%       Results displayed using SN_test_Showdata

% Initialise
num     = 80;                           % Number of steps to take
SN      = zeros(num,1);                 % Signal to noise ratio (x axis)
pulses  = zeros(num,1);                 % Measured pulse rates
errors  = zeros(num,1);                 % Errors in the measured pulse rates
j       = 10;                           % Step size

% Calculate the period of sine waves with increasing noise
for(i=1:num)
   SN(i) = Makewaves(100,0.7,0,0,0,j,2048,1000,1,5);
   [pulses(i),errors(i)] = pulse(100,4,10,400);
   j = j + 5                            % Display where we're up to
end

% Save the data for displaying later
outf=fopen('c:\masters\SN.txt','w');
for(i=1:num)
   fprintf(outf,'%f %f %f\n',SN(i),pulses(i),errors(i));
end
fclose(outf);

function SN_Test_Sqr();
%
%       Determine the limiting signal to noise ratio for the
%       pulse measurement algorithm using square wave input signals
%       Results displayed using SN_test_Showdata

% Initialise
num     = 80;                           % Number of steps to take
SN      = zeros(num,1);                 % Signal to noise ratio (x axis)
pulses  = zeros(num,1);                 % Measured pulse rates
errors  = zeros(num,1);                 % Errors in the measured pulse rates
j       = 10;                           % Step size

%Calculate the period of square waves with increasing noise
for(i=1:num)
   SN(i) = Makesqrwaves(100,0.7,0.85,0,0,0,j,2048,500,1,5);
   [pulses(i),errors(i)]=pulse(100,4,10,400);
   j = j + 5                            %Display where we're up to
end

%Save the data for displaying later
outf=fopen('c:\masters\SNsqr.txt','w');
for(i=1:num)
   fprintf(outf,'%f %f %f\n',SN(i),pulses(i),errors(i));
end
fclose(outf);


function SN_test_Showdata();
%
%       Display the measured periods for input signals
%       with increasing signal to noise ratios
%

% Flag to decide whether to include error ranges in the plot
plot_errors = 1;

% Load data and initialise buffers
data=load('C:\masters\SNb.txt');
SN=data(:,1);
pulses=data(:,2);
errors=data(:,3);

% Create the graph
if( plot_errors == 0 )
   plot(SN,pulses);
else
   %Create an x axis vector that goes from min to max and back to min
```

```
   X=[SN
      flip(SN)];
   %Create a y axis vector with + and - the error values
   a=pulses+errors;
   b=pulses-errors;
   Y=[a
      flip(b)];
   % Fill the error region
   fill(X,Y,'g');hold on;
   % Plot the pulse values
   semilogx(SN,pulses,'k.-');hold off;
end
zoom on;
```

B.6 – Utility routines

These routines were used in a number of the simulations to perform functions that are not part of the inbuilt Matlab function set.

```
function rms = rms(vec);
%
%       Return the root mean square of a vector
%       Syntax:
%               [rms_value] = rms(vector)
%       Parameters:
%               vector = a column vector containing the input signal
%       Example:
%               rms( sin(0:2*pi/1000:2*pi) )
rms = sqrt(mean(vec.*vec));

function noise = whitenoise(length);
%
%       White noise generator. Creates a random buffer in the frequency
%       domain and uses an inverse Fourier Transform to convert it into
%       the time domain. The rms value of the random buffer is 1
%       Syntax:
%               noise = whitenoise(length);
%       Parameters:
%               length = the length of the whitenoise buffer wanted
%       Example:
%               noise = whitenoise(bufsize)
%
freqvec         = rand( length*2, 1 );
temp            = abs( ifft(freqvec) );
noise           = zeros( length, 1 );
for( i = 2:length+1 )
   noise(i-1) = temp(i);
end
mx = rms(noise);
noise = noise/mx;

function vec2 = flip(vec);
%
%       Returns a vector with the element order reversed
%       Syntax:
%               upsidedown = flip(rightwayup)
%       Parameters:
%               rightwayup = the vector to be reordered
%       Example:
%               upsidedown = flip(rightwayup)
len=length(vec);
vec2=zeros(len,1);
for(i=1:len)
   vec2((len+1)-i)=vec(i);
end

function data_out = movingavr(data_in,order);
%
%       Returns a buffer containing the centred moving average
%       of an input buffer.  'NaN' values are left out of the
%       averaging but if the number of sequential 'NaN's is
%       greater than the moving average buffer width an error is
%       generated. The first and last data points are used to fill
```

```
%       the moving average buffer at the boundary conditions
%       Syntax:
%              data_out = movingavr(data_in,order)
%       Parameters:
%              data_in  = the data buffer (1-D vector) to be averaged
%              order    = the width of the moving average buffer.
%                         Order should be a positive, non zero number
%                         greater than the number of sequential 'NaN's
%                         Order may be an even number, in which case the
%                         moving average buffer is rounded up to the nearest
%                         odd number and the first and last values in the
%                         buffer give half contributions
%       Example:
%              smoothed = movingavr( lumpy, 10 );
%

%Initialise variables and buffers
isodd = 0;
window_half_size = ceil( order/2 );          % half the window size
window_size = 2 * ceil( (order+1)/2 )-1;     % Round up to nearest odd number
if( window_size == order )
   isodd = 1;
end
window  = zeros( window_size,1);             % Cyclic buffer
len     = length( data_in );                 % Length of the data_in buffer
inbuf = [data_in(1)*ones(window_half_size,1) % Extend data_in to avoid boundary
         data_in];                           % conditions

% Initialise the window
for( i = 1:window_size )
   window(i) = inbuf(i+isodd);
end

% Do the centered moving average
inbufptr = window_size + isodd + 1;
window_ptr = window_size;                    % Averaging Window cyclic pointer
for( i = 1:len )
   data_out(i) = winavr(window,window_size,window_ptr,isodd,order);
   window_ptr = cyclicInc(window_ptr,window_size);
   window(window_ptr) = inbuf(inbufptr);
   if( inbufptr < len+window_half_size )
      inbufptr = inbufptr + 1;
   end
end

function avrge = winavr(window, window_size, pointer, isodd, order );
%
%       Local function of movingavr.m
%
avrge = 0;
div = order;
if( isodd == 1 )       % then take a normal average skipping 'NaN's
   for( i = 1:window_size )
      if( isnan(window(i)) )
         div = div-1;
      else
         avrge = avrge + window(i);
      end
   end
   if( div == 0 )
       error('Window size too small or too many NaNs in a row');
   end
   avrge = avrge / div;
else                                  % the pointer value and one behind the pointer
value
                                      % only contribute half
   ptr = pointer;
   if( isnan(window(ptr)) )
      div = div-1;
   else
      avrge = window(ptr)/2;                  %last point only contributes half
   end
   ptr = cyclicInc(ptr,window_size);
   for( i = 1:window_size-2 )
      if( isnan(window(ptr)) )
         div = div-1;
```

```
      else
         avrge = avrge + window(ptr);
      end
       ptr = cyclicInc(ptr,window_size);
   end
   if( isnan(window(ptr)) )
      div = div-1;
   else
      avrge = avrge + window(ptr)/2;          %first point only contributes half
   end
       if( div == 0 )
       error('Window size too small or too many NaNs in a row');
   end
   avrge = avrge / div;
end

function newptr = cyclicInc(oldptr,top);
%
%       Local function of movingavr.m
%
newptr = oldptr+1;
if( newptr > top )
   newptr = 1;
end

function S_N = Makewaves( amp, period, phase, driftslope, driftoffset, noiseamplitude,
offset,
                          bufsize, xmin, xmax)
%
%       Returns a sine wave with noise and drift saved in a
%       text file in the format
%                       (time, signal, pure signal, noise signal)
%                       (time, signal, pure signal, noise signal)
%                       (time, signal, pure signal, noise signal)
%       Also returns the signal to noise ratio for the wave just created
%       Syntax:
%              S_N = Makewaves( amp, period, phase,
%                               driftslope, driftoffset
%                               noiseamp, offset, bufsize, xmin, xmax )
%       Parameters:
%              amp             = wave amplitude
%              period          = wave period
%              phase           = initial phase (0 -> 2*pi)
%              driftslope      = gradient of drift
%              driftoffset     = y intercept of drift
%              noiseamp        = amplitude of superimposed noise
%              offset          = center offset of the wave
%              bufsize         = length of the buffer containing the waveform
%              xmin            = initial x value for generating the wave
%              xmax            = final x value for generating the wave
%       Example:
%              SN = Makewaves(100, 0.7, 0, 0, 1, 0, 1, 2048, 1000, 1, 5);
%

% Initialise variables and buffers
stepsize       = (xmax-xmin)/bufsize;
sig            = zeros(1,bufsize);
noise          = whitenoise(bufsize) * noiseamplitude - noiseamplitude/2;
sigwn          = zeros(1,bufsize);

%Generate Signal
step = xmin;
for i=1:bufsize
   noise(i)= noise(i) + driftslope*step + driftoffset;
   if(period ~= 0)
       sig(i)=amp*sin(2*pi*step/period + phase);
   else
      sig(i)=amp;
   end
   sigwn(i)=sig(i)+noise(i)+offset;
   signal(i,1) = step;
   signal(i,2) = sigwn(i);
   step = step + stepsize;
end

%Calculate the signal to noise ratio
```

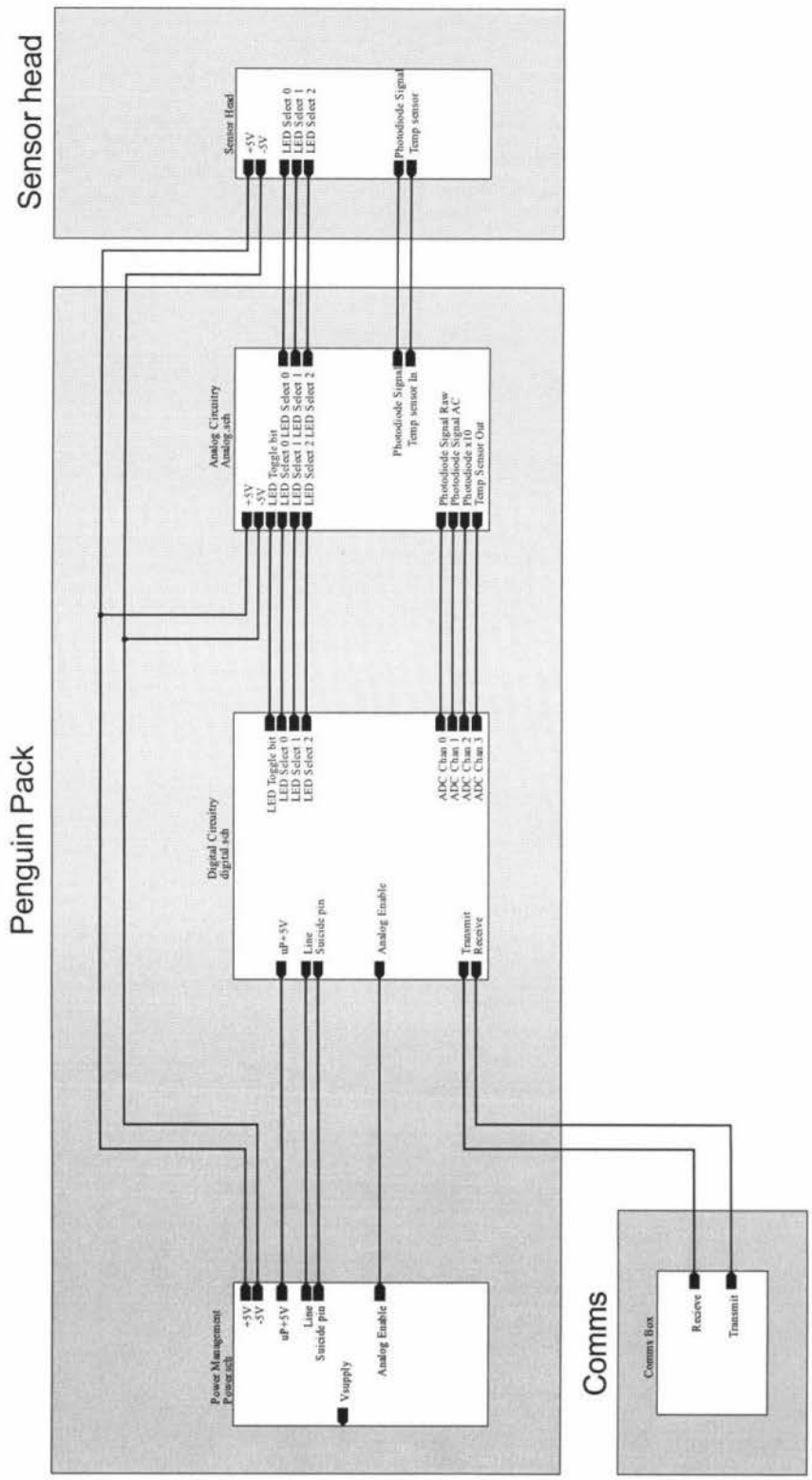
```
srms=rms(sig);
nrms=rms(noise);
if(nrms~=0)
   S_N = srms/nrms;
else
   S_N = inf;
end

% Display the signal created
plot(signal(:,1),signal(:,2));zoom on;

%Save the waveform in a text file
outf=fopen('c:\masters\sig.txt','w');
step = xmin;
for(i=1:bufsize)
   fprintf(outf,'%5.4f %f %f %f \n',step,sigwn(i),sig(i),noise(i));
   step = step + stepsize;
end
fclose(outf);

function S_N = MakeSqrwaves( amp, period, T1, phase, driftslope, driftoffset,
noiseamplitude,
                             offset, bufsize, xmin, xmax )
%
%       Returns a asymmetric wave with noise and drift saved in a
%       text file in the format
%                             (time, signal, pure signal, noise signal)
%                             (time, signal, pure signal, noise signal)
%                             (time, signal, pure signal, noise signal)
%       Also returns the signal to noise ratio for the wave just created
%       Syntax:
%              S_N = MakeSqrwaves( amp, period, T1, phase,
%                                  driftslope, driftoffset,
%                                  noiseamplitude, offset, bufsize, xmin, xmax)
%       Parameters:
%              amp             = wave amplitude
%              period          = wave period
%              T1              = fraction of period where signal is high
%              phase           = initial phase (0 -> 2*pi)
%              driftslope      = gradient of drift
%              driftoffset     = y intercept of drift
%              noiseamp        = amplitude of superimposed noise
%              offset          = center offset of the wave
%              bufsize         = length of the buffer containing the waveform
%              xmin            = initial x value for generating the wave
%              xmax            = final x value for generating the wave
%       Example:
%              SN = Makesqrwaves(100, 0.7, 0.85, 0, 0, 1, 0, 1, 2048, 500, 1, 5);
%

% Initialise variables and buffers
sig             = zeros(bufsize,1);
noise           = whitenoise(bufsize)*noiseamplitude-noiseamplitude/2;
sigwn           = zeros(bufsize,1);
stepsize        = (xmax-xmin)/bufsize;
numofperiods    = (xmax-xmin)/period;
stepsperperiod = floor(bufsize / numofperiods);
stepsperT1      = floor(stepsperperiod * T1);
stepsperT2      = stepsperperiod - stepsperT1;
cycle           = [amp*ones(1,stepsperT1) zeros(1,stepsperT2)]; %one period

% The waveform has a phase shift roll the copy of one period
% round to the correct phase
if(phase ~= 0)
   j=floor(stepsperperiod*(phase/(2*pi)));
   for(i=1:stepsperperiod)
      inphasecycle(i)=cycle(j);
      j=j+1;
      if(j==stepsperperiod) j=1; end
   end
   cycle=inphasecycle;
end

%Generate Signal
j=1;
step = xmin;
```

```
for i=1:bufsize
   noise(i)=noise(i) + driftslope*step + driftoffset;
   sig(i)=cycle(j);
   sigwn(i)=sig(i)+noise(i)+offset;
   j=j+1;
   if(j==stepsperperiod) j=1; end
   signal(i,1) = step;
   signal(i,2) = sigwn(i);
   step = step + stepsize;
end

% Calculate the signal to noise ratio
srms=rms(sig);
nrms=rms(noise);
if(nrms~=0)
   S_N = srms/nrms;
else
   S_N = inf;
end

% Display the signal created
plot(signal(:,1),signal(:,2));zoom on;

% Save the waveform to a text file
outf=fopen('C:\masters\sig.txt','w');
step = xmin;
for(i=1:bufsize)
   fprintf(outf,'%5.4f %f %f %f \n',step,sigwn(i),sig(i),noise(i));
   step = step + stepsize;
end
fclose(outf);
```

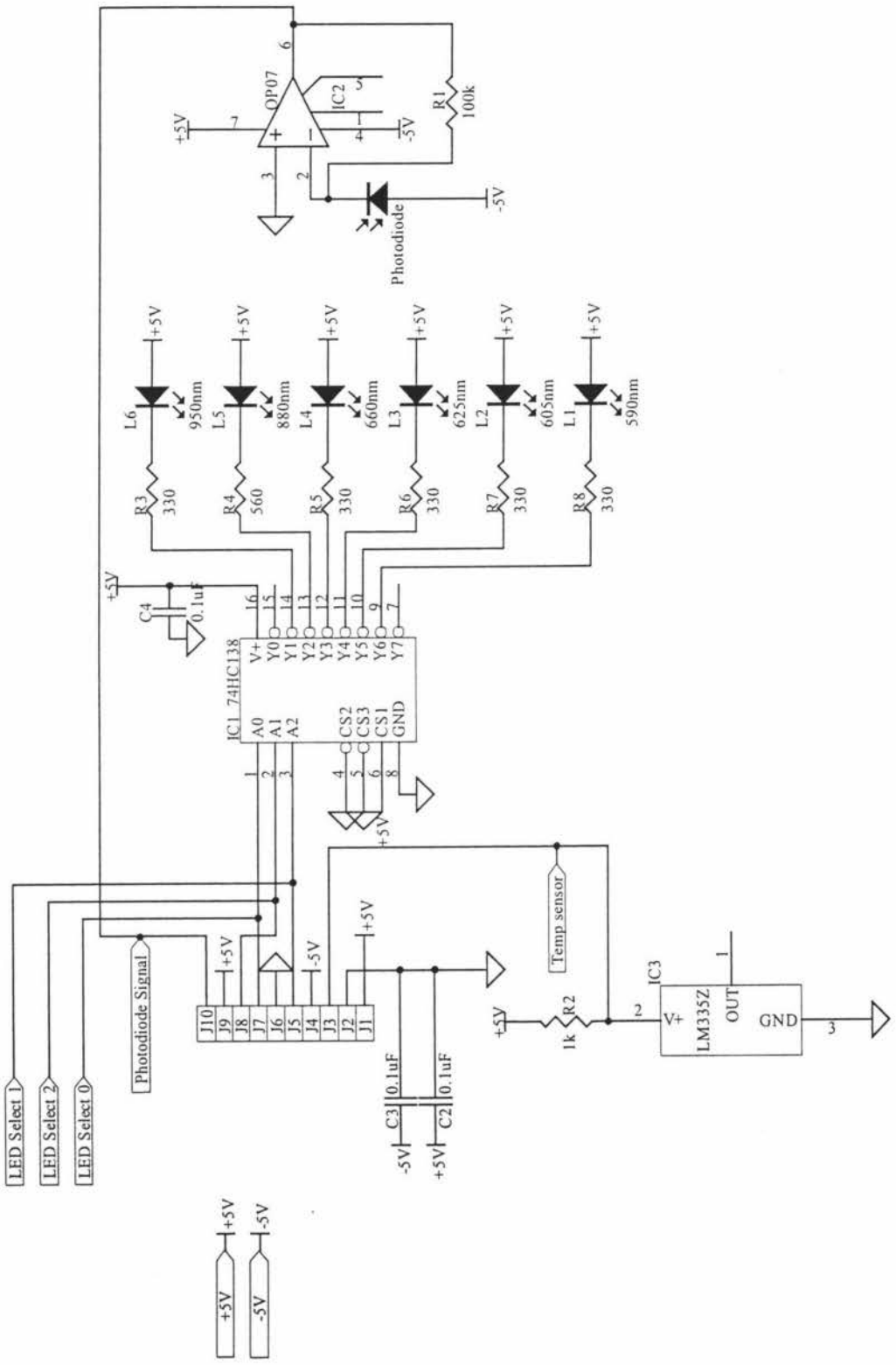
Appendix C – Circuit Diagrams

All circuit diagrams and PCB layouts were created using the EDA/Client 98 software package produced by Protel International Pty Ltd.

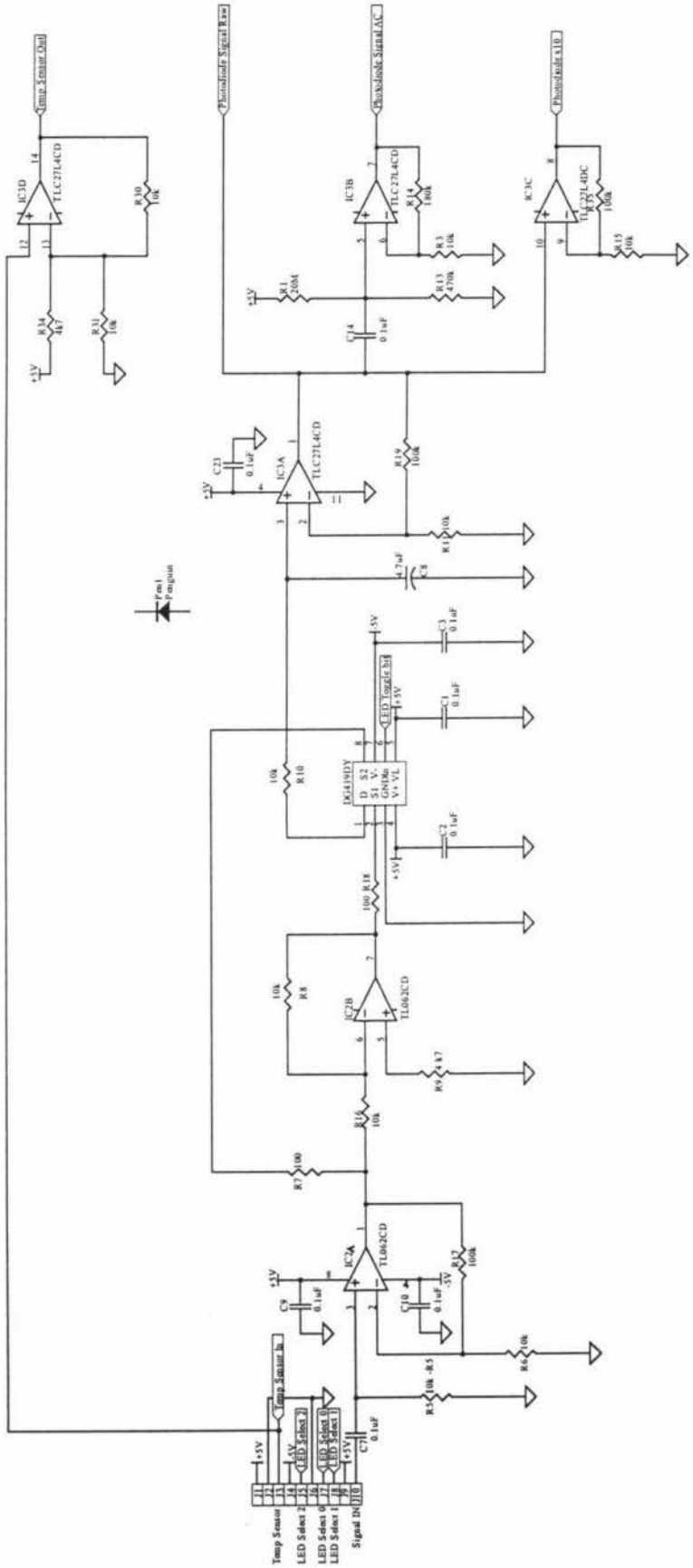
C.1 Overview



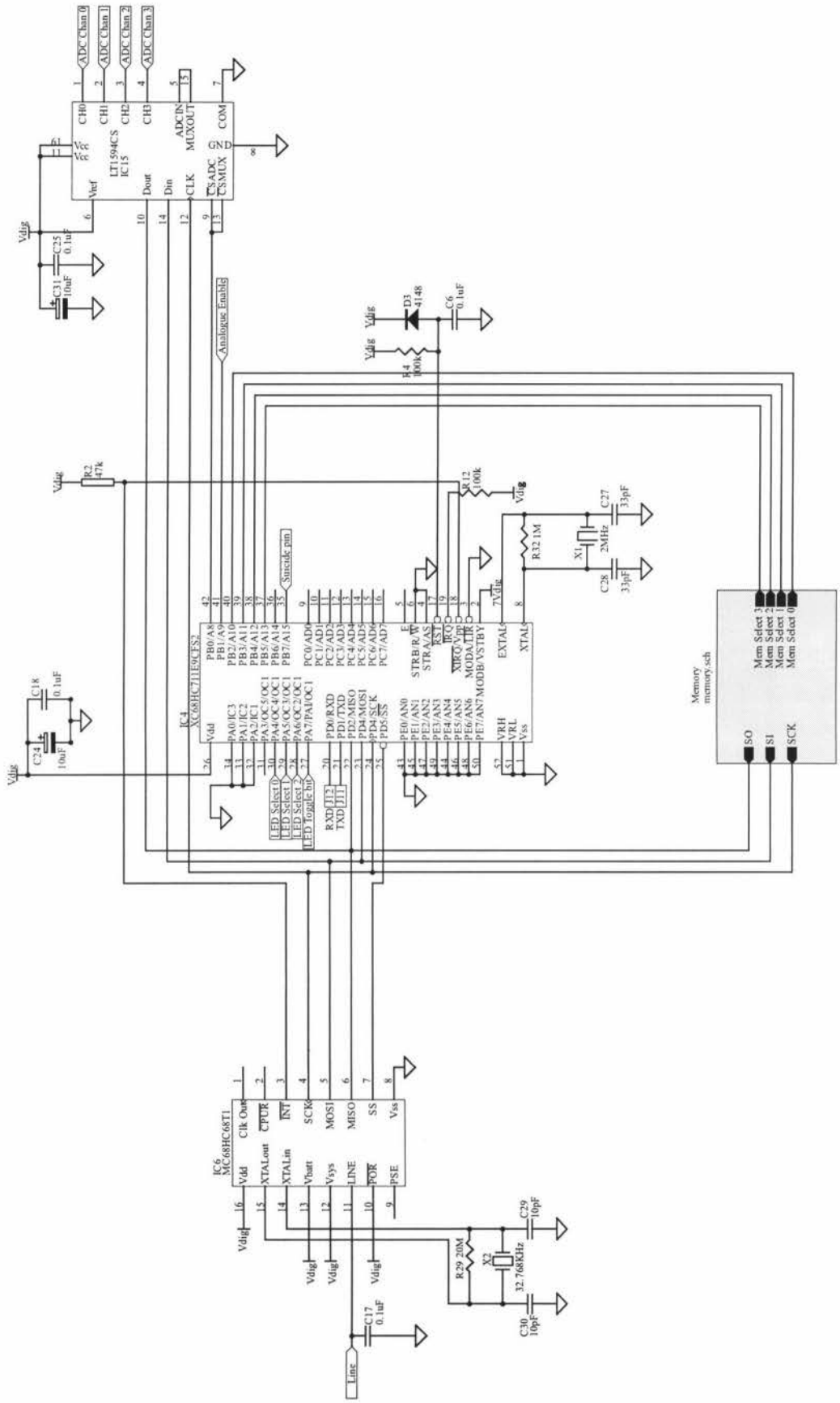
C.2 Sensor head



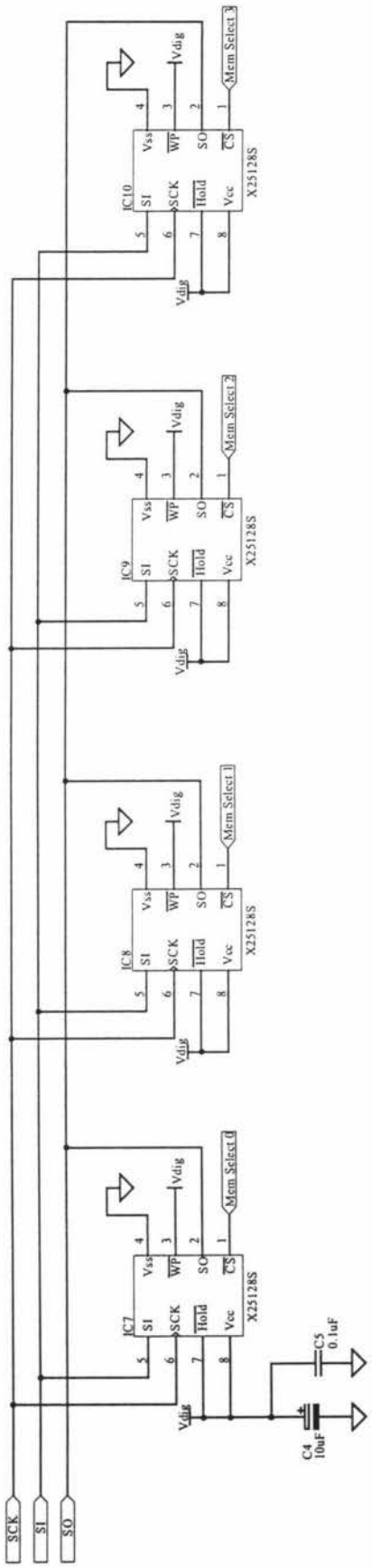
C.3 Analogue Stage



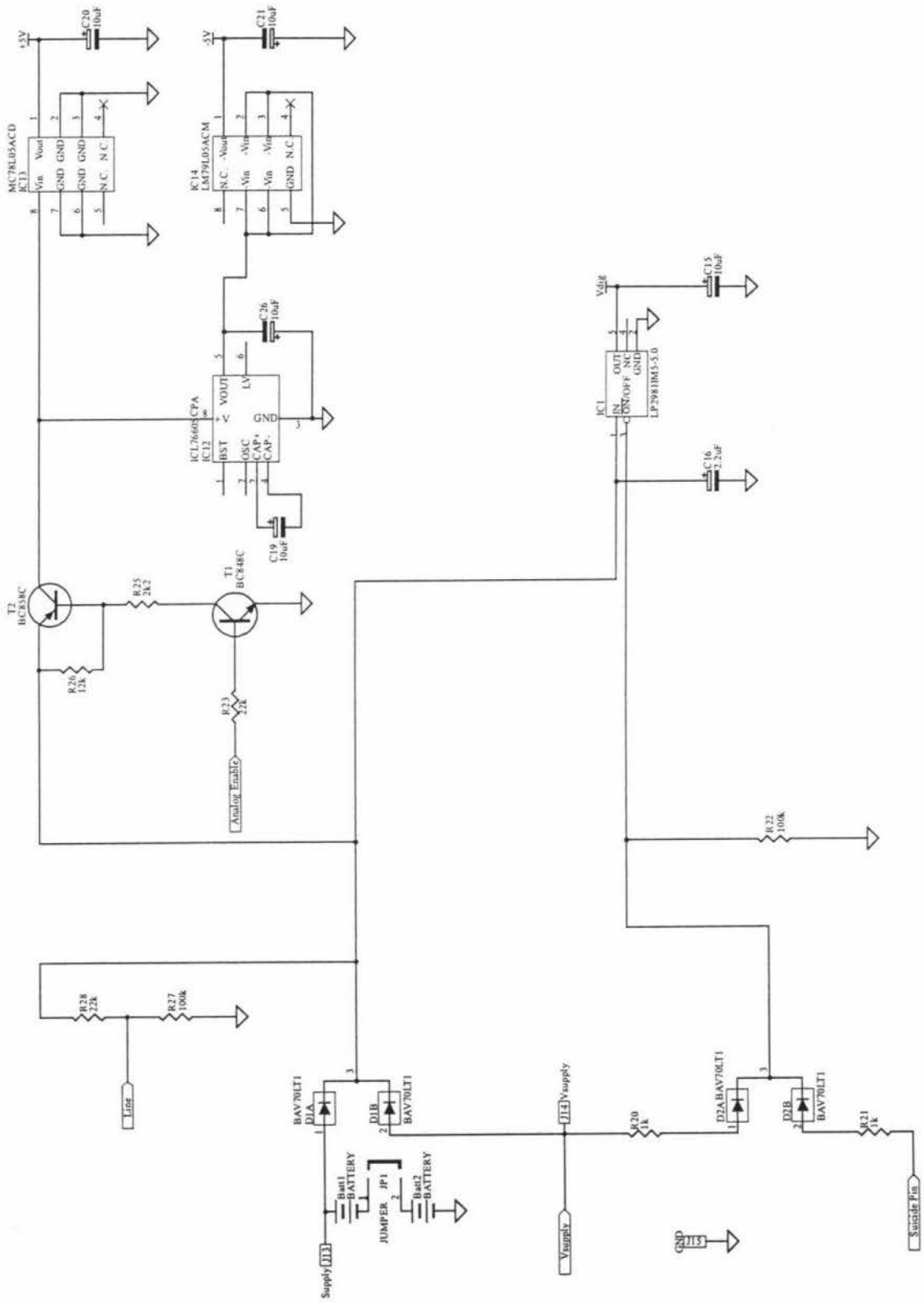
C.4 Digital Stage



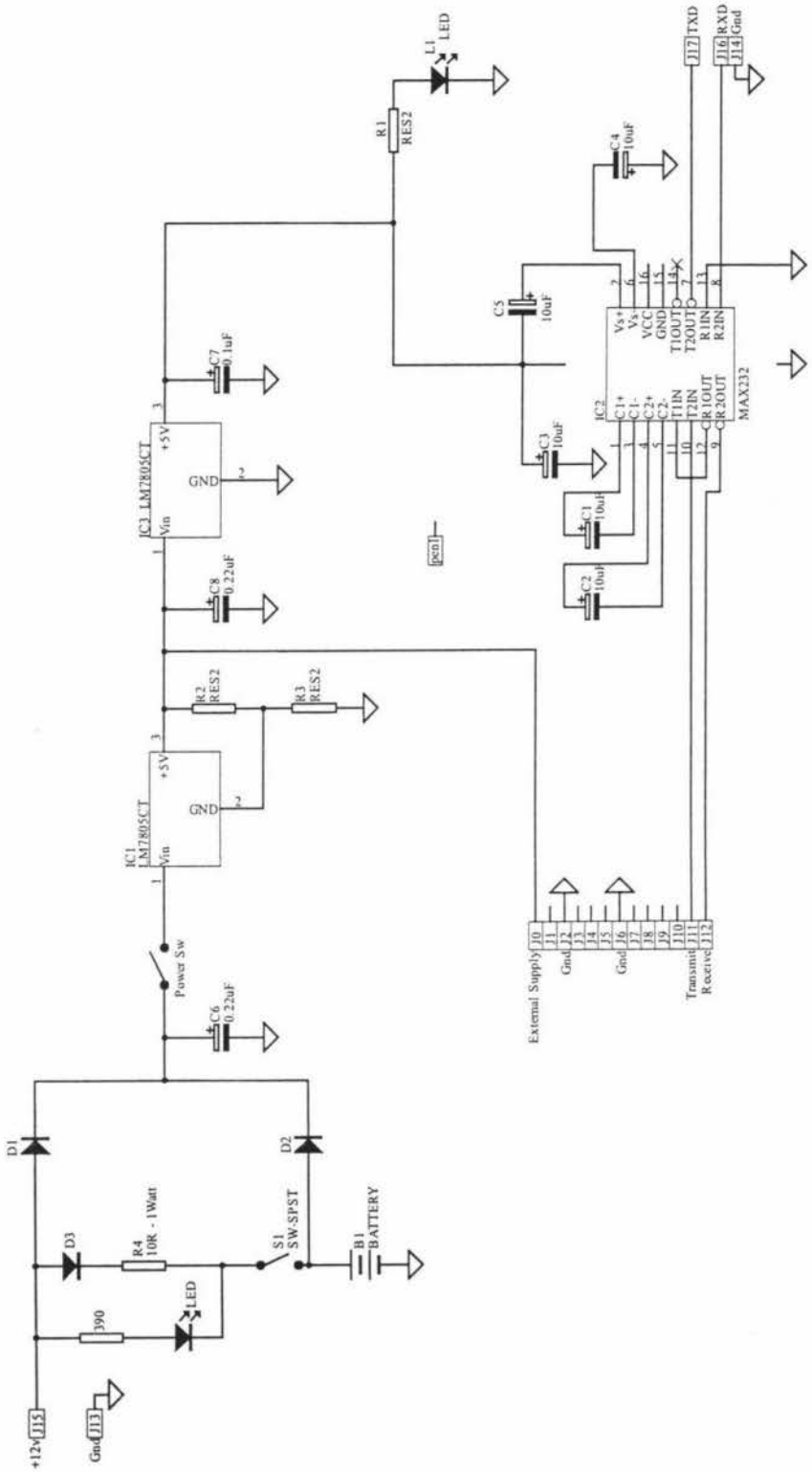
C.5 Memory


SCK
SI
SO
IC7
IC8
IC9
IC10
X25128S
Mem Select 0
Mem Select 1
Mem Select 2
Mem Select 3
Vdig
C4
10uF
C5
0.1uF

C.6 Power Supply



C.7 RS-232 Interface Unit



Appendix D – Printed Circuit Board Layouts

All circuit diagrams and PCB layouts were created using the EDA/Client 98 software package produced by Protel International Pty Ltd.

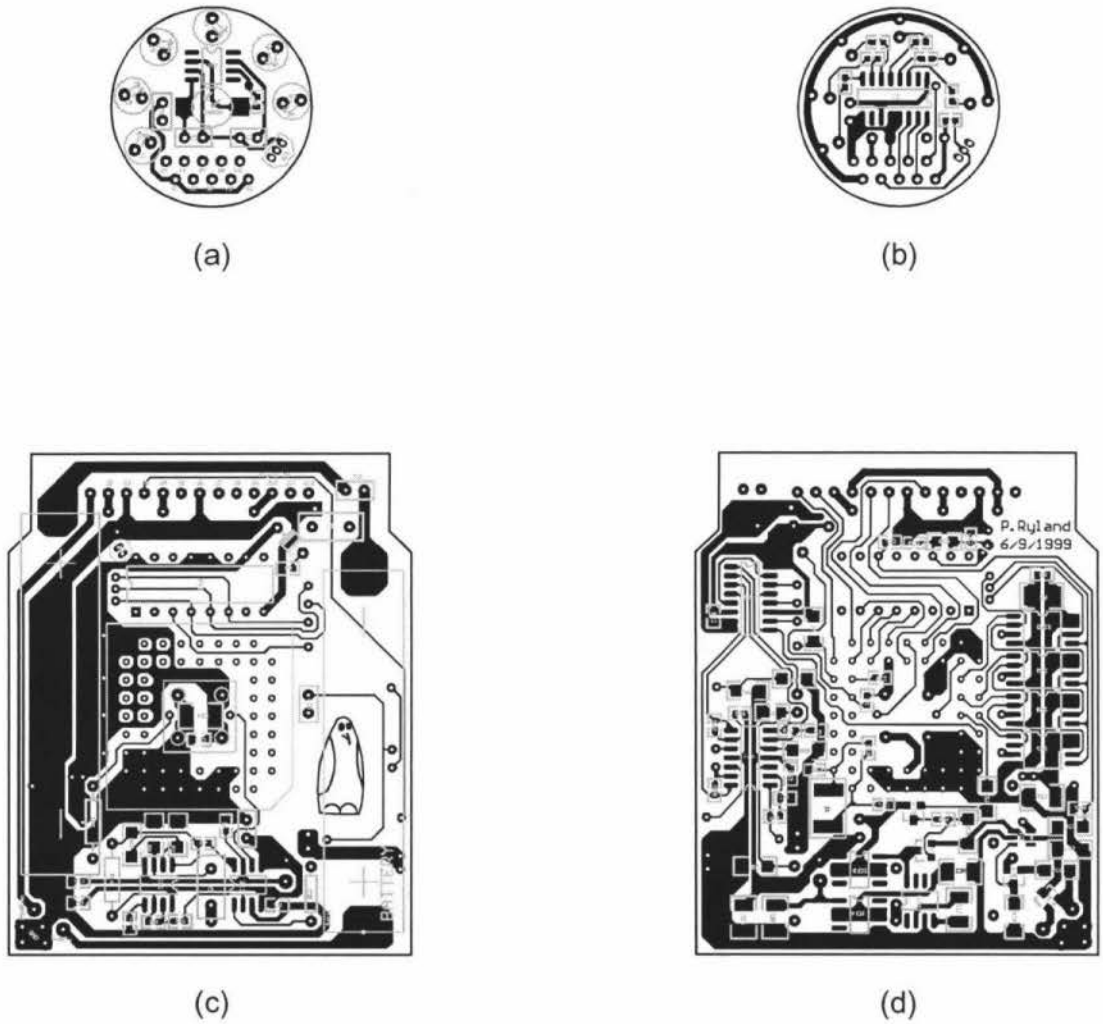
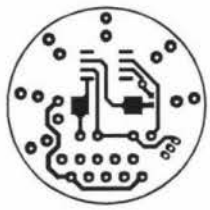
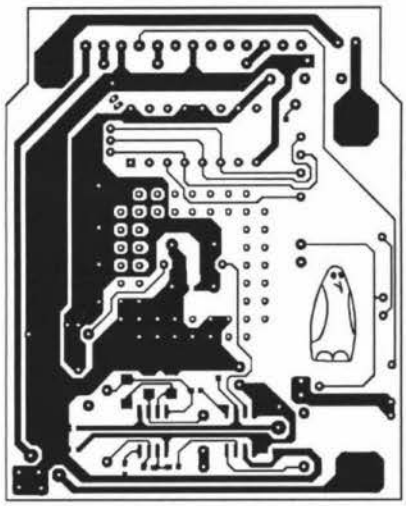


Figure D.1: Printed circuit board layouts with component overlays. (a) The sensor head from above, (b) the sensor head from below, (c) the control unit from above and (d) the control unit from below.

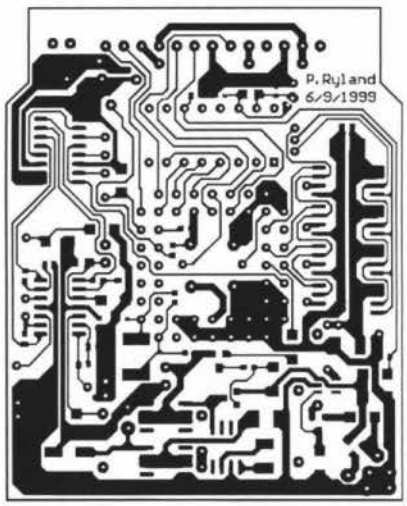


(a)

(b)



(c)



(d)

Figure D.2: Printed circuit board layouts. (a) The sensor head from above, (b) the sensor head from below, (c) the control unit from above and (d) the control unit from below.

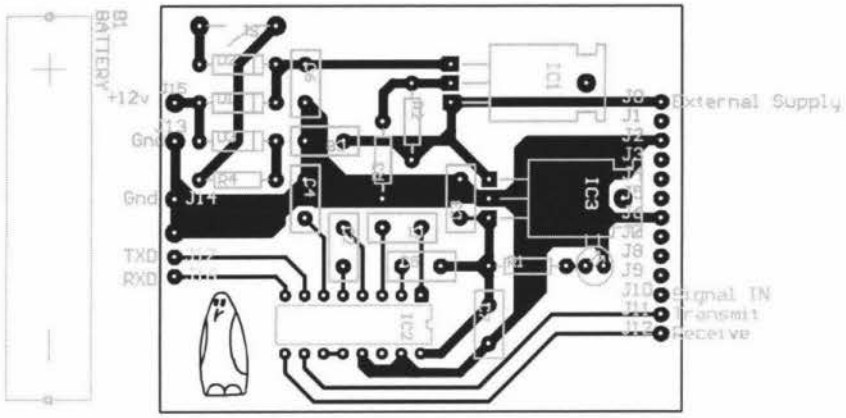


Figure D.3: Serial interface unit PCB layout.

Appendix E – Assembly Source Code

The assembly source code files for the device software can be found on the accompanying CD. They are located in the directory labelled 'Appendix E - Assembly Code'. The files have been formatted for ease of reading and are saved as Microsoft Word 97 documents and as Adobe Acrobat documents.

References

[1] Ainley, D.G., LeResche, R.E., & Sladen, W.J.L., (1983). *Breeding Biology of the Adélie Penguin,* University of California Press, Berkeley.

[2] Quaresima, V., deBlasi, R. A., & Ferrari, M., (1995). Customised optrode holder for clinical near-infra-red spectroscopy measurements, *Medical & Biological Engineering & Computing,* **33**, 627 – 628.

[3] May, J., (1988). *The Greenpeace book of Antarctica*, Macdonald Publishers (NZ) Ltd, Auckland, New Zealand.

[4] United States Geological Survey, (1995). *Atlas of Antarctic Research*, United States Antarctic Research Centre, http://usarc.usgs.gov/antarctic_atlas/start.html, (June 2000).

[5] Cope, M., & Delpy, D. T., (1988). System for long-term measurement of cerebral blood and tissue oxygenation on newborn infants by near infrared transillumination, *Medical & Biological Engineering & Computing,* **26**, 289 – 294.

[6] Shiga, T., Tanabe, K., Nakase, Y., Shida, T., & Chance, B., (1995). Development of a portable tissue oximeter using near infrared spectroscopy, *Medical & Biological Engineering & Computing*, **33**, 622 – 626.

[7] Mendelson, Y., & Ochs, B. D., (1988). Noninvasive pulse oximetry utilizing skin reflectance photoplethysmography, *IEEE Transactions on Biomedical Engineering*, **35**, 798 – 805.

[8] Wilson, B. C., Sevick, E. M., Patterson, M. S., & Chance, B., (1992). Time-dependent optical spectroscopy and imaging for biomedical applications, *Proceedings of the IEEE,* **80**, 918 – 930.

[9] Wilson, K., & Walker J., (2000). *Principles and Techniques of Practical Biochemistry*, *Fifth edition,* Cambridge University Press, United Kingdom.

[10] Liu H., Miwa M., Beauvoit B., Wang N. G., & Chance B., (1993). Characterization of absorption and scattering properties of small-volume biological samples using time-resolved spectroscopy, *Analytical Biochemistry,* **213**, 378 – 385.

[11] Geddes, L. A., (1997). Heritage of the tissue-bed oximeter, *IEEE Engineering in Medicine and Biology,* **16**, 87 – 91.

[12] Kumar, G., & Schmitt, M., (1997). Optimal probe geometry for near-infrared spectroscopy of biological tissue, *Applied Optics*, **36**, 2286 – 2293.

[13] Williams, T. D., (1995). *The Penguins,* Oxford University Press, New York, USA.

[14] Culik, B. & Wilson, R. P., (1991). Swimming energetics and performance of instrumented Adélie penguins (*Pygoscelis adeliae*), *Journal of Experimental Biology,* **158**, 355 – 368.

[15] Wilson, R. P., Spairani, H. J., Coria, N. R., Culik, B. M., & Adelung, D., (1990). Packages for attachment to seabirds: what color do Adélie penguins dislike least? *Journal of Wildlife Management*, **54**, 447 – 451.

[16] Louw, G J., (1992). Functional anatomy of the penguin flipper, *Journal of the South African Veterinary Association,* **63**, 133 – 120.

[17] Simpson, R. E., (1987). *Introductory Electronics for Scientists and Engineers. Second edition,* Allyn and Bacon Inc. Newton, Massachusetts, USA.

[18] Brigham, E. O., (1988). *The Fast Fourier Transform and its Applications*, Prentice Hall, Englewood Cliffs, New Jersey 07632, USA.

[19] Motorola, (1989). *M68HC11 Reference Manual,* Prentice Hall, Englewood Cliffs, New Jersey 07632, USA.

[20] Motorola, (1988). *M68HC11E9 Advanced Information. HCMOS Single-Chip Microprocessor,* Prentice Hall, Englewood Cliffs, New Jersey 07632, USA.

[21] Linear Technology Corporation, (1996). *LTC1594 4-Channel, 3V Micropower Sampling 12-Bit Serial I/O A/D Converter,* LT/GP 0596 5K, USA.

[22] Motorola, (1996). *Real-Time Clock plus RAM with Serial Interface,* MC68HC68T1/D, Rev. 2.

[23] Atmel Corporation, (1998). *AT25256 SPI Serial EEPROMs*, Rev. 0872E-08/98.

[24] Peacock, C., (1995). Beyond Logic, http://www.beyondlogic.org/, (Aug 1999).

[25] Stanley, W. D., Dougherty, G. R., & Dougherty, R., (1984). *Digital Signal Processing, Second edition*, Prentice Hall, Englewood Cliffs, New Jersey 07632, USA.

[26] Schott, A., & Snell, E. H., (1963). *Cardiographic Technique, Second edition*, Whitefriars Press Ltd., London.

[27] Kerry, K. R., Agnew, D. J., Clarke, J. R., & Else, G. D., (1992). Use of morphometric parameters for the determination of sex of Adélie penguins, *Wildlife Research,* **19**, 657 – 664.

[28] Wilson, R. P., Wilson, M.-P. T. J., (1989). Tape: A package-attachment technique for penguins, *Wildlife Society Bulletin*, **17**, 77 – 79.