# Design and Construction of a Vibration Data Logging Prototype Board for Overland Conveyor Belts

**R.A. Verrinder**

Department of Electrical Engineering

University of Cape Town

Rondebosch

7700



**Supervisor: Professor J. Tapson**

**July 2006**

This thesis was conducted in fulfilment of the requirements of a Masters of Science Degree in Electrical Engineering, MSc (Eng), at the University of Cape Town.

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

# Declaration

I, Robyn Verrinder, declare that is my own unaided work, apart from those sections that have been acknowledged. This thesis or any part of it has not been, nor will be submitted for any degree at this or any other university.



**Signed:** signature removed **Date:** 31st July 2006

# Acknowledgements

I would like to thank my supervisor Professor J. Tapson for suggesting this thesis topic and for his support and guidance throughout the duration of this project.

I am very grateful to the following people: Grant Stowe, Stephen Schrire and Samuel Ginsberg who have provided many useful suggestions and help in the design and construction of the prototype board.

I would like to thank Mandy Mason for her continuous support during this project. Many thanks to Kate and Shirley Mason who have also helped me throughout these two years.

My family has provided me with constant love and encouragement throughout my studies at UCT and this project would not have been possible with out them. I am very grateful to them for this and dedicate this project to them.

# Terms of Reference

Professor J. Tapson, from the Department of Electrical Engineering at the University of Cape Town, initiated this project in February 2004. A research group was formed to investigate different aspects of the system. Overland Conveyor Belts are used in mining and mineral processing industries to transport materials from one point in the processing infrastructure to another. It is important that the system is well monitored and maintained in order to minimize system down time and cost. Most of the existing condition monitoring methods only identify failure of the belt itself rather than identifying a problem before it can cause significant belt damage. The purpose of the research project was to design and implement a wireless vibration sensor which can be embedded in the actual belt. The venture was split into two different areas of research, the design of a vibration sensor and the design of the power generation system. The project was carried out in conjunction with S.A Williams. This thesis focused on the design and construction of a prototype Digital Signal Processing Vibration Sensor Board, while S.A. Williams' thesis investigates the design and implementation of a power generation and charging system for the embedded sensor.



The terms of reference were to:

1. Research the literature on various methods of online vibration monitoring techniques and gain a thorough understanding of the theory behind them.
2. Understand the basic design of an overland conveyor belt system.
3. Understand the hardware subsystems used in the design of an online vibration sensor.
4. Design and construct a vibration data logging board.
5. Write suitable programs with the embedded C programming language which will run and perform the required signal processing tasks.
6. Test the system in simulation.
7. Analyse the results of the testing.
8. Draw conclusions from the results and make suitable recommendations.

# Abstract

Overland conveyor belt systems form a vital part of modern transportation systems in the mining and mineral processing industries. It is vital that the system is well maintained in order to minimise system downtime and maximise profit. The conveyor belt is the single most expensive item in the system. It must be monitored to pick up potential problems before they cause belt failure. The majority of conveyor belt condition monitoring methods identify belt failure events rather than belt failure causes.

The purpose of this project was to research and design a belt condition monitoring board which could be physically embedded in a conveyor belt. This would then be used to monitor the condition of the conveyor idlers whose failure can result in major system damage. The venture was split into two areas of research: the design of a vibration data logging board and the design of a power generation system. This thesis focused on the design and construction of a DSP vibration data logging prototype board, while S.A. Williams investigated the design of a power generation system.

A battery powered vibration data logging board with digital signal processing board capability was designed and constructed. It contained three onboard condition monitoring sensors: a dual axis accelerometer which was used to monitor mechanical vibration, an electret condenser microphone which picked up audible sound and an analogue temperature sensor which measured the temperature of the board. Up to sixteen minutes of data can be recorded and stored onboard the device. The vibration data logger can upload data to a personal computer via a serial link.

The power supply system and anti aliasing filters were thoroughly tested. The low voltage dropout regulators significantly reduced noise output from the switch mode power supply. Therefore a switch mode power supply with low voltage dropout regulators forms low input current, efficient, low noise power supply system. The anti-aliasing filters functioned as required and successfully filtered out high frequency signal components.

The software modules and subsystems used onboard the vibration data logging board were each tested separately. The functionality of every module could then be verified. A project called Initial Testing was created in MPLAB® 7.30. Each test was written in a different file.

Software was written for the integration of the all the functions used to control the modules on the vibration data logging board. Due to the failure of the Write/Read operation of the AT454DB321C Dataflash® this integration program could not be implemented.

# Table of Contents

# List of Illustrations

## FIGURES:

## TABLES:

# Glossary

| | |
|---|---|
| **μA** | Microampere |
| **μF** | Microfarad |
| **μH** | Micro-Henry |
| **μV** | Microvolt |
| **Ω** | Ohm |
| **A** | Ampere |
| **A·h** | Ampere Hour |
| **AC** | Alternating Current |
| **ADC** | Analogue to Digital Converter |
| **AE** | Acoustic Emission |
| **AGND** | Analogue Ground Supply |
| **ALU** | Arithmetic Logic Unit |
| **ANSI** | American National Standards Institute |
| **ASCII** | American Standard Code for Information Interchange |
| **AVDD** | Analogue Positive Supply |
| **BPFI** | Inner Ball Pass Frequency |
| **BPFO** | Outer Ball Pass Frequency |
| **BSF** | Ball Spin Frequency |
| **°C** | Degrees Centigrade |
| **CH** | Channel |
| **cm** | Centimetre |
| **CKE** | Clock Edge |
| **CKP** | Clock Polarity |
| **CMOS** | Complementary MOS Field Effect Transistor |

| | |
|---|---|
| **CODEC** | Coder/Decoder |
| **CPU** | Central Processing Unit |
| **dB** | Decibel |
| **DC** | Direct Current |
| **DSP** | Digital Signal Processing |
| **EEPROM** | Electrically Erasable Programmable Read Only Memory |
| **F** | Farad |
| **FFT** | Fast Fourier Transform |
| **FR** | Flame Resistant |
| **FSF** | Frequency Scaling Factor |
| **FTF** | Fundamental Train Frequency |
| ***g*** | Unit of Gravity (9.8 m/s) |
| **GND** | Ground |
| **H** | Henry |
| **HFRT** | High Frequency Resonance Technique |
| ***Hz*** | Hertz |
| **I/O** | Input/Output |
| **IC** | Integrated Circuit |
| **ICD2** | In-circuit Debugger |
| ***i*MEMs** | Integrated Micro Electro Mechanical System |
| **kΩ** | Kilo-ohm |
| **kg** | Kilogram |
| **kHz** | Kilohertz |
| **kV** | Kilovolt |

**kW** ..... Kilowatt

**LCC** ..... Leadless Chip Carrier
**LCD** ..... Liquid Crystal Display
**LDO** ..... Low Dropout Regulator
**LED** ..... Light Emitting Diode

**m** ..... Metre
**MΩ** ..... Mega-ohm
**mA** ..... Milliampere
**mF** ..... Milli-farad
**mH** ..... Milli-Henry
**mil** ..... Thousandths of an Inch
**MIPS** ..... Million Instructions per Second
**mm** ..... Millimetre
**MOSFET** ..... MOS Field Effect Transistor
**MUX** ..... Multiplexer
**mV** ..... Millivolt
**mW** ..... Milliwatt

**N** ..... Newton
**nA** ..... Nanoampere
**NiMH** ..... Nickel Metal Hydride
**nV** ..... Nanovolt

**PC** ..... Personal Computer
**PCB** ..... Printed Circuit Board
**pF** ..... Picofarad
**PLL** ..... Phase-Lock Loop
**PSU** ..... Power Supply Unit
**PWM** ..... Pulse Width Modulation

**Q** ..... Quality Factor

| | |
|---|---|
| **Q1.15** | 16-bit Signed Fractional Format |
| **rad** | Radian |
| **rad/s** | Radian/Second |
| **RAM** | Random Access Memory |
| **RFID** | Radio Frequency Identification |
| **RMS** | Root Mean Square |
| **ROM** | Read Only Memory |
| **RPM** | Revolutions Per Minute |
| **SAR** | Successive Approximation Register |
| **SI** | International System of Units |
| **SMD** | Surface Mounted Device |
| **SMPS** | Switch Mode Power Supply |
| **SNR** | Signal to Noise Ratio |
| **SOIC** | Small Outline Integrated Circuit |
| **SOT** | Small Outline Transistor |
| **SPI** | Serial Peripheral Interface |
| **SRAM** | Static Random Access Memory |
| **TQFP** | Thin Quad Flat Pack |
| **TSOP** | Thin Small Outline Package |
| **UART** | Universal Asynchronous Receiver Transmitter |
| **USB** | Universal Serial Bus |
| **V** | Voltage |
| **VDD** | Positive Digital Supply Voltage |
| **VDL** | Vibration Data Logger |
| **ver.** | Version |
| **Vp-p** | Peak to Peak Voltage |

| | | |
|---|---|---|
| **W** | ..... | Watt |
| **WT** | ..... | Wavelet Transform |
| **XT** | ..... | Crystal |

# Symbol Definitions

| | |
|---|---|
| $\alpha$ | Contact Angle |
| $\phi$ | Phase Angle |
| $\omega$ | Angular Frequency |
| $\omega_s$ | Shaft Rotation Frequency |
| $\omega_c$ | Fundamental Train Frequency |
| $\omega_b$ | Ball Spin Frequency |
| $\omega_{id}$ | Ball Pass Frequency Inner |
| $\omega_{od}$ | Ball Pass Frequency Outer |
| $a$ | Acceleration |
| $A$ | Amplitude |
| $C$ | Capacitor Value |
| $Cf$ | Crest Factor |
| $d$ | Rolling Element Diameter |
| $D$ | Pitch Diameter |
| $f$ | Frequency |
| $f_c$ | -3dB Cut-off Frequency |
| $f_{cy}$ | Instruction Clock Frequency |
| $f_{max}$ | Maximum Frequency |
| $f_s$ | Sampling Frequency |
| $I_L$ | Load Current |
| $j$ | $\sqrt{-1}$ |
| $L$ | Inductor Value |

| | | |
|---|---|---|
| $N$ | ..... | Number of Samples |
| $Q$ | ..... | Quality Factor |
| $R$ | ..... | Resistor Value |
| $T$ | ..... | Period |
| $T_2$ | ..... | PWM Period ADXL202 |
| $T_{AD}$ | ..... | ADC Clock Period |
| $T_{conv}$ | ..... | Conversion Time |
| $T_{cy}$ | ..... | Instruction Clock Period |
| $T_{SAMP}$ | ..... | Sampling Period |
| $v$ | ..... | Velocity |
| $V_{Dropout}$ | ..... | Dropout Voltage |
| $V_n$ | ..... | Noise Voltage |
| $V_{in}$ | ..... | Input Voltage |
| $V_{out}$ | ..... | Output Voltage |
| $V_R$ | ..... | Regulator Voltage |
| $V_s$ | ..... | Signal Voltage |
| $x$ | ..... | Discrete Vibration Signal |
| $x_i$ | ..... | $i$th Sample |
| $\bar{x}$ | ..... | Mean of a Signal |
| $Z$ | ..... | Number of Rolling Elements |

# 1 Introduction

## 1.1 Background to the Research Project

Overland conveyor systems form an integral part of modern industrial transport. They are used in mining and mineral processing industries to move raw material from the mine to the processing plant. They can range in length from a few kilometres to as many as 15.6 kilometres, which is the length of the longest single flight curved steel cord conveyor in the world. This record is held by the Zimbabwe Iron and Steel Company (Zisco) overland conveyor which transports iron ore from Ripple Creek to a crushing plant in Redcliff in Zimbabwe (Fig. 1.1).

**Figure 1.1: Zisco Overland Conveyor System**

This overland conveyor system is 15.6 km long and is used to transport iron ore blends from Ripple Creek to a crushing facility of the Zimbabwe Iron and Steel Company (Zisco) located in Redcliff in Zimbabwe (Taken from Nordell L.K.) [1].

Conveyor systems have been gaining popularity in recent years as a reliable alternative to discontinuous transport systems such as trucks. High fuel, labour costs and increasing environmental concerns have made overland conveying a more desirable option as opposed to conventional truck hauling [2].

The initial capital outlay to install a long overland conveyer belt system is high, however once the system is in place it is one of the most economical methods of transporting material from one point to another. The belt is the single most expensive item in a conveyor system and can account for more than one third of the total installation cost [3]. Therefore, it is vital that it is well maintained in order to keep operating costs as low as possible. The conveyor belt system links two major parts of the processing infrastructure. It is essential that it operates continuously when necessary as a stoppage would result in large losses for the company involved. The system must be well monitored and maintained so that potential problems can be identified and corrected early.

There are several possible failure modes that can occur with overland conveyor belting. Some of these will be discussed in more detail in Section 3.1.2 and Chapter 4, and include:

- Belt Tracking Failure - leads to edge of belt damage
- Belt Reinforcing Cable Failure - leads to belt breakage
- Idlers Bearing Failure - leads to idlers seizing, which causes friction, abrasion, overheating, possible fire damage and ripping of the belt

Failure modes that result in significant belt damage are problematic as belts are usually custom made for a specific conveyor system and are costly and time consuming to repair and replace.

Most existing methods of wear detection focus on monitoring the condition of the belt itself. This is achieved by examining cover wear, carcass damage and cover hardness. This, together with rip detection, can increase the lifespan of a belt.

There are currently a number of companies who produce belt monitoring systems. They employ various techniques to determine the condition of the belt. These range from portable magnetic reluctance probes which measure distances to the steel cables in the belt [4] to ultrasonic transducers which measure belt width [3].

One of the most common methods of belt rip detection is to embed small figure eight conductive loops in the bottom cover of the belt (Fig.1.2).

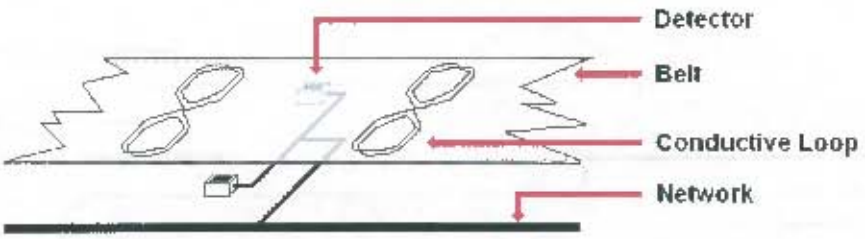



**Figure 1.2: Conductive Loop Rip Detection System**

Conductive loops are embedded through out the length of the conveyor belt. Pulses are outputted by the detectors when a continuous loop passes overhead, resetting a timer. If there is a break in a loop and therefore the belt, the timer will time out and stop the conveyor.

These loops are placed at regular intervals and extend across the entire belt width. Detectors are placed on the structure of the conveyor after an area of potentially high risk. The detectors output a pulse if a continuous loop passes over them which resets a timer. If there is a break in the loop, the timer will time out and stop the conveyor **[5]**.

The disadvantage with most of these systems is that they only identify a failure in the belt once it has occurred rather than isolating a potential problem before it has time to causes significant belt damage. It would be preferable to have a method which could detect idler and tracking failure before these failure modes can cause major system damage.

## 1.2 Objectives of the Research Project

### *1.2.1 Purpose of the Research Project*

Online vibration monitoring is one technique that could be used to detect idler bearing failure **[6]**. This system would monitor changes in the vibration profiles of the idler bearings, which would give an indication of their type and level of deterioration. In most machine condition monitoring systems a vibration sensor would be placed on each part of the system that needs to be analysed **[7]**. It is not possible to do so in this case due to the large number of idlers in the system. It would be beneficial if a suitable online vibration detection system could be implemented without the expense of having a vibration sensor on each idler.

A possible solution to this problem is to embed a wireless vibration sensor in the actual belt. The purpose of the research project is to investigate the feasibility of such a system by designing a vibration sensor that could be used to detect faulty idlers in a conveyor system.

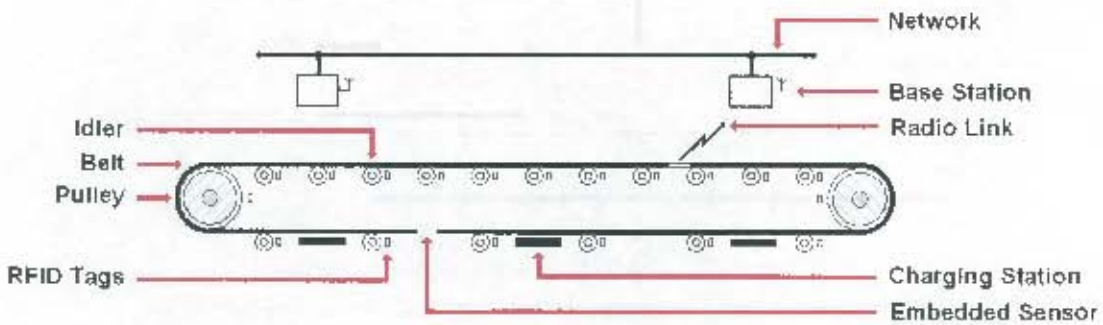



**Figure 1.3: A Simplified Schematic of the Embedded Vibration Sensor System**

A wireless vibration sensor could be embedded in the belt and powered by electromagnetic induction as it passes over charging stations. RFID tags are used to locate the position of the faulty idler. Readings are stored onboard the sensor and are downloaded to external base stations via a radio link. These base stations are linked to a database server where all data can be stored and analysed.

The embedded sensor would measure the vibration levels of the idler bearings and would be powered within the belt using electromagnetic induction. Passive Radio Frequency Identification (RFID) tags could be used as position indicators on each idler throughout the system (Fig. 1.3).

One or more external base stations would be placed around the conveyor system. When the sensor passes by these stations it would download the stored measurements via a radio link. The base stations would be connected to an external database server, where all the results can be stored and analysed.

### *1.2.2 Problems to be Investigated*

The objectives of the project were to:

1. Research the various online vibration monitoring methods in the literature and understand the theory and concepts involved.
2. Understand the basic design of an overland conveyor belt system.
3. Understand the hardware subsystems that are used in the construction of a data logging vibration sensor.

4. Design and build a digital signal processing (DSP) board which can be used to monitor vibration.
5. Write software which can run the vibration sensor in embedded C programming language. This software must control the functioning of the sensor and analysis of the results.
6. Test the system in simulation.
7. Analyse the results of the testing.
8. Draw conclusions from the results and make suitable recommendations.

## 1.3 Scope of the Thesis

This venture was carried out as part of a joint project with S.A. Williams. The focus of the project was to design an embedded vibration sensor for overland conveyor belts which could successfully locate faulty idlers in the system.

The project was divided into two main areas of research:

1. Design and implementation of a digital signal processing board, which can measure, log and analyse vibration, temperature and sound.
2. Design and implementation of a power generation and charging station for the embedded sensor.

This thesis deals with the design, construction and testing of a DSP Board, while S.A. Williams' project focuses mainly on investigating possible power generation methods for the embedded sensor.



## 1.4 Limitations of the Research Project

The main limitation of this project was that the system could not be tested on an actual overland conveyor system.

## 1.5 The Plan of Development of the Thesis

The thesis begins with a look at the purpose and nature of the project in the introduction (Chapter 1). This is followed by a literature review which focuses on current maintenance policies and their place in overland conveyor belt systems (Chapter 2). Different condition monitoring techniques used on roller element bearings are then considered (Chapter 3). Overland conveyor belt systems are then discussed with a detailed look at an actual conveyor system called the Kangra Savmore 6.5 km Overhead Conveyor (Chapter 4). This system was used as a guideline to calculate many of the parameters used in this project.

The hardware design of the Vibration Data Logging Prototype Board is laid out in Chapter 5. The hardware design requirements are discussed and each subsystem on the board is looked at in detail. Next the software design of the Vibration Data Logging Prototype Board is considered and again each subsystem is analysed (Chapter 6).

The methods that were used to test the hardware and software systems and the results of the testing are described in Chapter 7 and Chapter 8. From the results, conclusions are drawn (Chapter 9), and recommendations and improvements for future designs and are given (Chapter 10).

# 2 Maintenance Methods and Strategies

In order for any system to run efficiently and reliably it must be well maintained and monitored through out its life cycle. This will ensure that a device or system will function as required. A good maintenance strategy must optimise each of the following qualities: reliability, availability, efficiency and capability according to the needs of the system **[8]**, while taking factors such as profitability, safety and environmental issues into account **[9]**. There are three main types of maintenance strategy: preventative, predictive and corrective, each with certain advantages and disadvantages **[10]**. Often these methods are not implemented independently of one another but in combination to provide the best overall maintenance plan. These methods will be discussed briefly in the following sections.



## 2.1 Corrective Maintenance

Corrective maintenance, also known as unplanned maintenance, is maintenance in its most basic form. A part of a system is utilised until it fails at which point it is repaired or replaced **[11]**. This failure can result in unscheduled down time and possible damage to other parts of the system which can be very costly. The result of the failure is often incorrectly identified as the cause of the problem **[12]**. It is therefore not the most ideal maintenance strategy unless a system is not significant or the cost involved in a preventative maintenance scheme is much higher than for a simple corrective maintenance plan **[11]**.

## 2.2 Preventative Maintenance

Unforeseen system failures can result in accidents, environmental damages and a decrease in product quality **[9]**. All of these factors are undesirable. It is important to limit the unpredictability of failures. This can be achieved by implementing a maintenance method that incorporates some form of planned maintenance. Preventative maintenance schemes were

introduced as a way to reduce unpredictable failures by performing routine maintenance on a system at regular intervals. Maintenance is conducted on the system irrespective of its condition and parts may be replaced even though they might still be functional. The main advantage of this form of time-based maintenance is that down-time can be planned in advance. On the other hand, preventative maintenance schemes require large labour forces and a great number of spare parts, and therefore may not be the most efficient form of maintenance unless the system's condition cannot be monitored because of logistics or expense.

## 2.3 Predictive Maintenance

During the 1970's a new type of maintenance plan was introduced known as condition-based maintenance [8]. In this strategy, different parameters such as vibration, temperature, noise, pressure etc. are monitored and compared to known failure patterns in order to determine the condition of the system. This type of predictive maintenance provides some "lead-time" to failure which means suitable maintenance can be preformed on the system before significant damage occurs [10]. Condition-based monitoring systems are becoming a vital part of any maintenance strategy as the data collected can be used to improve machinery and the current preventative maintenance schemes.

In the past, most monitoring was done with simple instruments used by highly skilled maintenance staff. Due to increases in sensor, computing and wireless technology, monitoring systems can now be remote with a link to a central computing system where the data can be analysed automatically [13].

Condition monitoring is being used more frequently in the mining industry and in overland conveyor belt systems. The main advantages of this type of maintenance strategy with respect to overland conveying in Australia, are laid out by [3] and include:

- A reduction in the time it takes to discover the fault behind a belt stoppage.
- Notification of maintenance personnel of the problem before they get to the site of the fault so that they can take the right equipment with them.

# 3 Rolling Element Bearing Monitoring

## 3.1 Rolling Element Bearings

### *3.1.1 An Introduction to Rolling Element Bearings*

A rolling element bearing is made up of round or rolling elements placed between two pieces of material. The most common type of rolling element bearing is the ball bearing (Fig. 3.1) and this, like most other bearings in this class, is made up of an inner and outer race with the rolling elements found in the groove formed between the races. A bearing cage holds each of the rolling elements in place within the bearing.



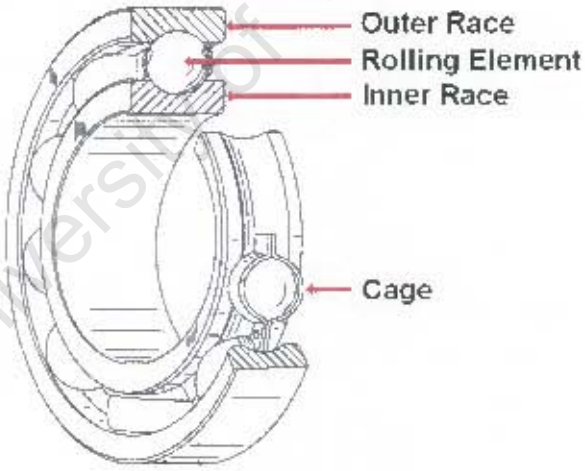



**Figure 3.1: A Schematic Diagram of a Ball Bearing**

A rolling element bearing is made up of an inner and outer race with rolling elements, in this case balls, found in the groove between them (Modified from Mutou, Y. *et al.* 2000) **[15]**.

Rolling element bearings are critical mechanical parts in most industrial equipment. Their failure is considered to be one of the largest causes of breakdown in rotating machinery **[16]**. Rolling element bearings support the idlers and pulleys in overland conveyor belts. A brief outline of the main rolling element bearing monitoring methods is laid out in Sections 3.2 and 3.3.

- Long term breakdown data can be used to analyse downtime and this then allows management to plan better preventative maintenance strategies.

Monitoring the condition of the actual conveyor belt in an overland conveyor system is vital as belting is the single most expensive item in the system. It therefore needs to be kept in good working condition. Idler bearing failure has been identified as one of the common causes of belt failure **[14]** and is therefore a main focus in this thesis. The conveyor belt system and its components are examined in more detail in Chapter 4.

### 3.1.2 *Failure Modes of Rolling Element Bearings in the Idlers of Overland Conveyor Belts*

The lifespan of a rolling element bearing is affected by certain factors such as the load on the bearings, the speed that they run at, alignment of the bearings, lubrication and cleanliness of the lubricant inside the bearing **[10, 17]**.

There are several failure modes that can occur in idler rolling element bearings. A few of these are listed below **[17, 18, 19]**:

1. *Rolling element contact fatigue* – This occurs when the constant cycling load of the rolling element causes fatigue cracks and pits to form in the rolling elements and races. The life of a correctly mounted and sealed rolling element bearing is rated in hours to the first sign of contact fatigue and is rated to be 20 000 to 50 000 hrs for typical idler bearings **[20]**.

2. *Inadequate lubrication* – This will cause greater surface contact between rolling elements and the races causing greater wear and excessive heating of the bearing parts. This eventually leads to bearing failure. Relubrication of bearings reduces the risk of them running dry, however this is a very expensive process in overland conveyor idlers and therefore most systems make no provision for this **[17]**.

3. *Lubrication failure and contamination* – If the incorrect lubricant is used, or if the lubricant is contaminated by any form of abrasive particle, it will cause accelerated wear of the bearing which could result in a failure of the part. If large amounts of contaminate have entered the bearing it might cause the part to seize leading to the failure of the idler. Lubrication contamination causes the side idlers to fail more often than main carrying idler, as particles settle on the side bearing seals and enter the bearing during expansion and contraction **[17]**. This is the most common failure mode in rolling element bearings in overland conveyor belt systems **[21]**.

4. *Misalignment of the idler bearings due to shaft deflection* – This causes unbalanced internal load distribution in the bearing which will cause excessive wear and may lead to bearing fatigue. It will also increase bearing friction which causes excessive heating and

possible bearing seizure. In the case of taper bearings it can lead to uneven pressure distribution across the length of the idler which can cause idler skew.

### *3.1.3 Damaged Rolling Element Bearing Vibration*

All rolling element bearings rotating at a constant speed produce specific periodic vibration signatures [22]. Damaged bearings produce significant increases in these signatures and this information provides engineers with a useful means of detecting wear and damage.

There are many different types of vibration from sound, to electromagnetic waves to the vibration of material objects such as a plucked guitar string. Any system that has inertia and stiffness will oscillate about its position of equilibrium when an external force is applied to it [10].

The time that it takes to complete one cycle is known as the period of oscillation ($T$). The number of vibrations per unit time is called the frequency of oscillation ($f$) and is calculated using Equation 3.1.

$$f = \frac{1}{T} \tag{3.1}$$

The SI unit for frequency is Hertz ($Hz$) and is defined as the number of completed oscillations per second. The amplitude of a vibration ($A$) is the maximum displacement of a vibrating body from its rest position.

A state known as resonance occurs when the driving force coincides with the natural frequency of the system. This causes the oscillations to reach their maximum amplitude. In practical vibratory systems, mechanical energy is lost to the surroundings as heat or sound energy. This is known as damping of the system. The amplitude and duration of the vibration depends on the damping of system and the response of the system to an external force.

If vibration reaches unacceptable levels in a component, it will cause accelerated wear and possible failure of that part. It is therefore important to limit vibration in a system by monitoring and analysing its causes.

Bearing defects can be classified as either distributed defects which are due to surface roughness or waviness, off-size rolling elements or misaligned races, or localised defects which are due to fatigue pits or cracks [23]. Typically, bearings fail due to localised defects, therefore most rolling element condition monitoring methods focus on their detection. When a rolling element comes into contact with a localised defect in a bearing, it produces an impulse which excites the resonances of the structure. The vibration signature of a damaged bearing, travelling at a constant rotational speed, is a series of periodic exponentially decaying ringing impulses whose period is determined by the geometry of the bearing, load on the bearing and the location of the defect [24, 25]. McFadden and Smith (1984) [26] proposed a model for the vibration produced by a single point defect in a rolling element bearing and White (1984) [27] explains a method to simulate machinery fault impulse signals. Localised defects are usually classified according to the area where damage has occurred i.e. outer race damage, inner race damage and rolling element damage. The frequencies at which the impulses occur are linked to the motion of the rolling elements in relation to the other bearing parts [23]. This motion can be expressed in terms of the fundamental frequencies of the bearing parts and can be calculated using the equations found in Appendix A.



## 3.2 Vibration Monitoring

Mechanical vibration (Section 3.2), sound, ultrasonic and acoustic emissions (Section 3.3) are all forms of vibration but differ in frequency range and the medium through which they are transmitted. The typical frequency range of mechanical vibration measurements is considered to be 1 Hz – 10 kHz [28]. These measurements are taken on the actual vibrating component.

There have been many studies over the past few decades concerning different measurement methods which can be used to analyse vibrational data from damaged rolling element bearings and Tandon and Choudhury (1999) [22] provide a comprehensive review of the different ideas used in the field. These methods can be split up into two main groups; time domain methods and frequency domain methods. These ideas are discussed in the following sections.

### 3.2.1 *Time Domain Methods*

Time domain methods are usually the simplest to implement and understand but are not always the most reliable and effective in noisy environments. They do have several advantages which are described by Birch (1994) **[19]**, these include speed and efficiency of computation, smaller memory requirements as calculations can be performed on the data as it is collected, and reduced data volume from N samples to one, therefore providing a simpler vibration parameter to interpret. Time domain methods include the following techniques: RMS vibration measurement, peak level, crest factor analysis, kurtosis analysis and shock pulse counting. A brief description of the some of the time domain methods follows:

1. *RMS Vibration Measurement*– This time domain technique is the easiest to implement and understand and therefore finds widespread use in condition monitoring equipment. The RMS value provides an accurate measure of a signal's energy and when a damaged bearing's vibration level increases so should the RMS vibration value. This means that it can be used as an indicator for bearing wear and damage and is given by Equation 3.2

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2} \tag{3.2}$$

Where $N$ is the number of samples and $x$ is the discrete vibration signal.

However, noise energy is also included in the RMS vibration value and therefore advanced damage has to have occurred before the RMS value shows a definite change which makes it an ineffective indicator of early damage. Vibrational acceleration measurements are often converted to an RMS measurement and usually provide more information than RMS vibrational displacement measurements **[25]**.

2. *Peak Level* – The peak level of a signal is defined as the maximum value that a signal reaches in a given time interval (Eq. 3.3).

$$peak = x_{\max}[n] \tag{3.3}$$

Gustafsson and Tallian (1961) [29] suggested a method of counting the number of peaks which crossed a predefined voltage level as a means of measuring rolling element bearing damage. The peak level is more often used in calculating the crest factor.

3. *Crest Factor Analysis* – The crest factor (*Cf*) of a signal is the peak value normalised by dividing by it the RMS value (Eq. 3.4).

$$Cf = \frac{peak}{RMS} \tag{3.4}$$

In this method the *Cf* is calculated and compared to the expected crest factor for that particular signal. If there is a discrepancy, it indicates that there is damage in the bearing.

4. *Kurtosis Analysis* – This statistical method was first suggested as a means to detect bearing wear by Dyer and Stewart (1978) [30]. Kurtosis is defined as the fourth central moment about the mean of a Gaussian distribution, normalised by dividing it by the forth power of the standard deviation (Eq. 3.5). It is a measure of the "peakedness" of a signal i.e. more of the variance of the signal is due to infrequent extreme deviations from the mean.

$$kurtosis = \frac{N\sum_{i=1}^{N}(x_i - \bar{x})^4}{\left(\sum_{i=1}^{N}(x_i - \bar{x})^2\right)^2} \tag{3.5}$$

Where $N$ is the number of samples, $x_i$ is the $i$th element of a signal $x$ and $\bar{x}$ is the mean.

The probability density function of the acceleration of a good bearing has been found to be Gaussian. When the bearing is damaged it produces an increased number of high levels of acceleration causing the probability density function to become non-Gaussian with dominant tails [22]. Kurtosis can therefore be used as an indicator of bearing wear as it is measure of the "peakedness" of the tails. An undamaged bearing produces a kurtosis value of approximately three and as the bearing wears, the kurtosis value increases. However, the kurtosis value decreases when the bearing becomes very damaged, due to an increase

in the rate and size of the peaks in the signal [30]. This method is not a very popular method in the condition monitoring industry [22].

5. *Shock Pulse Counting* – The shock pulse method was developed by SPM Instruments Ltd. U.K. and is used widely in industry [22]. The impulses generated by a damaged bearing are high in frequency while the vibrations from other sources are low in frequency. It is therefore possible to use band passing techniques to filter out the desired high frequencies. The shock pulse method uses a piezoelectric transducer (with a resonant frequency of 32 kHz) to produce an electronic filter which filters out the low frequency vibrations while impulses created by defects in a bearing produce damped oscillations in the transducer at the resonant frequency. The magnitude of these shock pulses is an indication of bearing condition, with 35dB indicating developing damage, 35-50dB indicating visible damage and 50-60dB indicating risk of failure [10].

### 3.2.2 *Frequency Domain Methods*

Frequency domain methods are often more difficult to implement and interpret than time domain methods [19]. Unlike in time domain methods, the shaft speed and bearing dimensions are needed for analysis. They are, however, possibly the most widely used methods in bearing damage detection as they provide large amounts of diagnostic information and perform well in noisier environments.

When a rolling element strikes a defect in a bearing it produces an impulse of very short duration. The impulse' s energy is spread over a wide band of frequencies at a very low level which means it can be hidden in vibration generated by other machine elements. This makes it difficult to detect using standard spectral techniques. Frequency domain methods utilise the fact that the impulses excite the resonances of the bearing. These resonances are at a much higher frequency than the vibration generated by other machine elements. This provides a narrower band of energy which can be detected more easily than the wide band energy of the impulse [31]. Three frequency domain methods are discussed below:

1. *High Frequency Resonance Technique (HFRT)* – This technique, also known as envelope detection, uses fact that defect impulses in rolling element bearings produce resonances which are amplitude modulated at the characteristic defect frequency. HFRT is a three

stage process. First the vibration signal is band-pass filtered around the resonance frequency removing frequencies from other sources. It is then demodulated using an envelope detector which rectifies the signal and smoothes it using a low-pass filter to remove the band-pass resonant frequency. This improves the spectral peaks produced by the damage in the bearing. A Fast Fourier Transform (FFT) is now taken of the demodulated signal and the power spectral peaks at the defect frequencies can now be analysed. McFadden and Smith (1984) **[31]** provide a comprehensive review of the technique.

2. *Cepstrum Analysis* – The cepstrum is defined as the inverse Fourier Transform of the log of the Fourier Transform of a signal. The power cepstrum, which is the power spectrum of the logarithmic power spectrum, can be used as a bearing diagnostic method **[22]**. The cepstrum of a signal shows a peak at the time or quefrency which corresponds to the period of the signal. It might be simpler to detect a peak in the power cepstrum than multiple peaks in the power spectrum therefore simplifying detection of the defect frequencies **[19]**.

3. *Wavelet Analysis* – This mathematical method is one of the fastest evolving signal processing tools **[32]**. The wavelet transform (WT) produces a variable time-frequency distribution from which periodic structural ringing can be detected **[22]**. The modulus of the wavelet transform can be used for fault diagnosis and shows how the energy of a signal varies with time and frequency.

## 3.3 Sound and Acoustic Emission Monitoring

Sound, also known as acoustic noise, is considered to be audible vibrations in the range 20 Hz to 20 kHz. It is expected that sound measurements will provide the same information as mechanical vibration measurements as they both lie in the same frequency range and mechanical vibration is partially transmitted from the machine to the air as sound **[28]**. These vibrations are influenced differently by external sources, with sound being especially susceptible to interference in a noisy environment. There has been little research carried out on sound measurement as a detection method for damaged rolling element bearings **[33]** although two techniques, sound pressure and sound intensity, are occasionally used.

Acoustic Emission Monitoring is another method which is widely used to monitor the condition of rolling element bearings. Transient elastic waves are produced in a material under stress and this is known as acoustic emission (AE) [34]. These signals lie in the higher vibrational frequency range (> 100 kHz) and this phenomenon occurs when a material undergoes plastic deformation or when cracks grow in a material. It can be used to detect the growth of subsurface cracks in bearings before these defects are able to produce significant vibration signals and therefore provide a useful early warning detection system.

# 4 The Conveyor Belt System

## 4.1 A General Overview of the System

Conveyor belt systems play a vital role in most modern processes, whether it is to transport coal to a power station, rotate grain in silos or even move baggage in an airport; they form the backbone of many different types of transport systems.

As stated by G. Davies (1981) **[35]**,

"*The conveyor is taken for granted as an obedient servant ready to move mountains at the touch of a button.*"

It is therefore, essential that the conveyor belt system runs as smoothly as possible without any major problems. Conveyor belts were first used as early as 1795 to transport grain over comparatively short distances. They have since grown in length, size and carrying capacity **[35]**.

A conveyor belt is made up of two end pulleys with some form of continuous material, known as a belt, looped around them. The belt rotates around the pulleys which are driven by drive motors and is supported, along its distance, by stands which contain idlers. This system provides a means of transporting material from one point to another and is used in many different applications in industry and agriculture. Due to advances in belt material design, it is now possible to make longer and stronger conveyor belting, therefore making overland conveyor belt systems a viable alternative to rail and road in bulk material transportation in the mining and mineral processing industry **[36]**.

A conveyor belt system can be divided into three main sections:

1. *The Tail* – The material to be transported is fed onto the conveyor belt in this section. It is important that the rate of feed is controlled to prevent damage to the belt and the system.

2. *The Middle* – This is the longest section in a conveyor belt system and contains all the support stands and idlers.

3. *The Head* The material is ejected at the head of the system. In a single drive system, the main drive is located at the head and this drives the primary drive pulley.

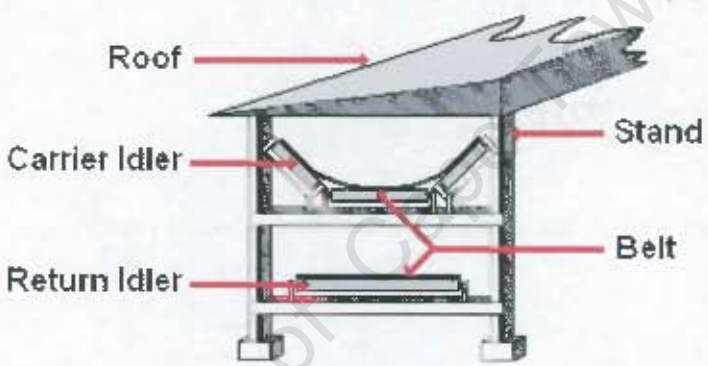



**Figure 4.1: A Simplified Diagram of the Conveyor Belt System**

The stands form the structural support of the system. They are placed at regular intervals throughout the length of the conveyor belt system. The typical distance between stands is about 4 m.



The conveyor belt specifications for the belt conveyor CV-08 for the Savmore Colliery are given in Appendix B and were used as a guideline in the vibration sensor design [37]. These values are typical of conveyor belts of comparable length and those that transport similar material. A brief description of the subsystems in an overland conveyor belt is given in the following sections.

### *4.1.1 The Belt*

The belt is the most expensive item in a conveyor belt system and can constitute up to 40 percent of the total installation cost [2]. Stresses form in the belt as it moves through the system. These include the following: [35, 38]

1. *Transient Stresses* form in the belt during starting and stopping of the system.
2. *Longitudinal Stresses* are generated in the belt by the drive pulleys during operation.
3. *Bending Stresses* arise in the belt as it moves around a pulley
4. *Flexing Stresses* occur as the belt moves over idlers forming a flexural transverse wave. This also adds a large dynamic loading on the idlers and idler bearings which increases their chance of failure [38].
5. *Indeterminate Stresses* are brought about by the load impacting on the belt.

It is important that the nominal tensile strength rating of the belt is at least 6.7 times greater than the operating tension of the belt to accommodate all unknown dynamic effects [39].

The belt is made up of a top and bottom cover with a belt carcass in between them. Historically, the belt carcass was made from multiple layers of woven cotton. Today, belt carcasses are made of layers of mono or multi-ply semi synthetic or synthetic woven fabrics separated by layers of rubber; or alternatively, a single layer of parallel galvanised steel cables embedded in rubber [40]. Steel cord belts, illustrated in Figure 4.2, have many advantages over other belts and these include: an increased tensile strength which enables them to be used over longer distances; increased flexibility due to the rubber and steel carcass which allows them to be used with deep troughing idlers; and thicker covers due to the construction of the carcass which increases belt life [35].

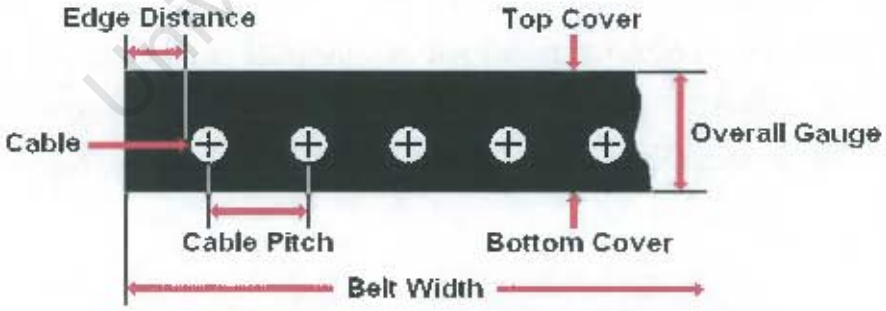



**Figure 4.2: The Structure of a Steel Cord Belt**

The belt is made up of a top cover, a bottom cover and steel cables. The steel cables form the belt carcass and reinforce the belt (Modified from Flexsteel belt construction from Goodyear Industrial Products) [41].

The type of material that covers are made out of depends on the function of the conveyor belt. Belt covers protect the belt carcass from weather damage and impacts from material being transported. They must also be resistant to high temperatures and substances such as oil.

### 4.1.2 *The Idlers*

An idler is made up of a roll or rolls held in place by a bracket which may or may not be supported on a base. The cylindrical roll consists of four main parts as shown in Figure 4.3: the outer shell, the shaft, the bearings and the seals.

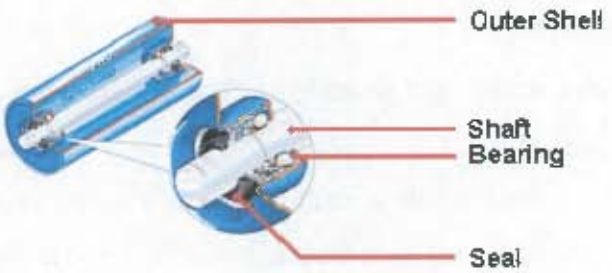



**Figure 4.3: The Structure of an Idler Roll**

An idler roll consists of four main components, the outer shell, the shaft, the bearings and the seals. (Modified from Osborn Engineered Products SA (Pty) Ltd. Series 25 conveyor idler specifications) [42].

The outer shell is made from steel piping and its diameter varies according to the requirements of the conveyor. Outer shell diameters can range from about 100 mm to 180 mm. The shaft is made from mild steel and is supported by the bearings, which are housed inside the shell. The bearings are shielded from dust and dirt by the seals.

The roll arrangement in an idler depends on the whether the idler is on the carrying or return side of the conveyor system. Carrying idlers are usually troughed in some way to ensure that the transported material remains centred on the belt. The typical carrying idler arrangement consists of one horizontal centre roll with a roll on either side of it as illustrated in Figure 4.4. The trough angle between the centre and the side rolls can vary between 20° and 45° [21]. The return idler arrangement consists typically of one flat horizontal roll or of two rolls angled in a 'V' shape.

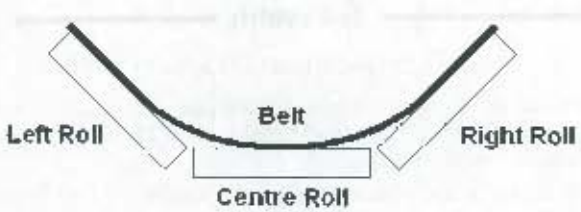



**Figure 4.4: The Carrier Idler Arrangement in the Conveyor Belt**

The side idler rolls are arranged at an angle of 20° to 45° to the centre idler roll to ensure that the material being transported remains centred on the belt.

It is vital that the belt runs centrally on the idlers and pulleys throughout the conveyor belt system as an untrained belt will run into the steel frame work damaging the edge of the belt. A belt is trained by adjusting the positions of the idlers, pulleys and load [43]. The two main methods used to train the belt with troughing idlers are: shifting the idler axis with respect to the direction of travel of the belt and tilting the troughing idler forward which causes the belt to self-align [44].

### *4.1.3 The Drives and Pulleys*

The drive motor or motors drive the pulleys which move the conveyor belt around the system. A conveyor belt system can either have a single head drive or multiple drives. A single head drive is expensive as it has to be large in order to provide enough power to move the belt. It does have the advantage, however, that all the major operating machinery is located in one position which reduces maintenance operations. Multiple drive systems may have drives located in the head, tail or some intermediate position. They are less expensive than single head drives and the rating of the belt used in the system can be lower as there is less tension on it [2].

The conveyor belt is pulled through the system by the drive pulleys. It is important that the diameter of the pulley is not too small as this will cause a large amount of stress and distortion of the belt. Goodyear Industrial Products [44] recommends that the pulleys are lagged, in other words that they are covered with some material. This will improve the coefficient of friction between the belt and the pulley, reduce slippage of the belt in wet conditions, increase the life of the pulley and the bottom cover of the belt and help prevent the build up of material on the pulley which causes belt wear. The face width of a drive pulley in a conveyor system is usually made 50 mm greater than the belt width for belts of 1000 mm and less, and 75 mm greater than the belt width for belts greater than 1000 mm [43].

### *4.1.4 Failure Modes of the Components in Overland Conveyor Belts*

There are several failure modes that can occur with overland conveyor belts. Some of these were briefly mentioned in Section 1.1. Goodyear Industrial Products [44] provides a comprehensive troubleshooting manual which gives possible problems that can occur with

overland conveyor belt systems and solutions to these problems. This list can be found in Appendix C.

Some of these problems include: untrained belts running into the conveyor structure damaging the edge of the belt; extreme belt stretch putting strain on the belt and on the belt cables; excessive belt cover wear in various forms from grooving of the cover to belt covers cracking; pulley lag wear causing belt damage; and idler failure which causes overheating and belt cover damage.

These problems may be prevented by monitoring the condition of the system. The following chapters focus on the design and construction of a vibration data logging board which can be used to monitor mechanical vibration, sound and temperature which are good indicators of system wear.

# 5 Hardware Design: Vibration Data Logger

## 5.1 An Overview of the Design and Requirements

Vibration sensing was proposed as a possible technique for monitoring the condition of idlers in a conveyor belt system [6]. The purpose of this project was to design and build a prototype vibration data logging board with digital signal processing capability that could be used as a condition monitoring system. This chapter provides a detailed look at the hardware of the sensor and each of the subsystems as well as the design considerations behind the selection of the components for system.

### *5.1.1 General Requirements of the Data Logging System*

A data logging device is used to measure, record and store a specified number of samples of a particular parameter in a system. It requires the following subsystems:

1. *A sensor* measures the desired parameter with some form of transducer which converts a real world parameter into an electrical signal.

2. *An analogue to digital converter (ADC)* converts the analogue electrical signal which comes from the sensor into a digital signal. The sampling rate of the ADC has to satisfy the Nyquist-Shannon Sampling Criterion (Eq. 5.1) which states that the sampling frequency ($f_s$) must be at least twice the maximum frequency ($f_{max}$) in a system to prevent aliasing of the signal.

$$f_s \geq 2 f_{max} \tag{5.1}$$

The ADC can be a stand alone device or built into a microcontroller chip.

3. *A processor* controls the actions of the device and can be anything from a microprocessor to a digital signal controller to some form of higher processing unit. It is used to perform calculations on the measured data, to transfer and retrieve data from memory and to control communication.

4. *Memory* is used to store the collected data until it is needed. It is interfaced with the processor and must have a large enough capacity to store the required number of samples.

5. *An interfacing and indicator system* provides the user with a means to interact with the data logging system. The interface system can consist of input and output systems. The input system enables the user to select or alter onboard parameters while the output system connects the data logging device to an external system, often some form of personal computer (PC). All the collected data can then be transferred to the external device for further processing and analysis. The indicator system provides the user with a visual or aural representation of the condition of the device

6. *The power supply* provides the power to run the device. The type of power supply depends on the nature of the data logging instrument; for example, a stand alone system requires a form of battery power.

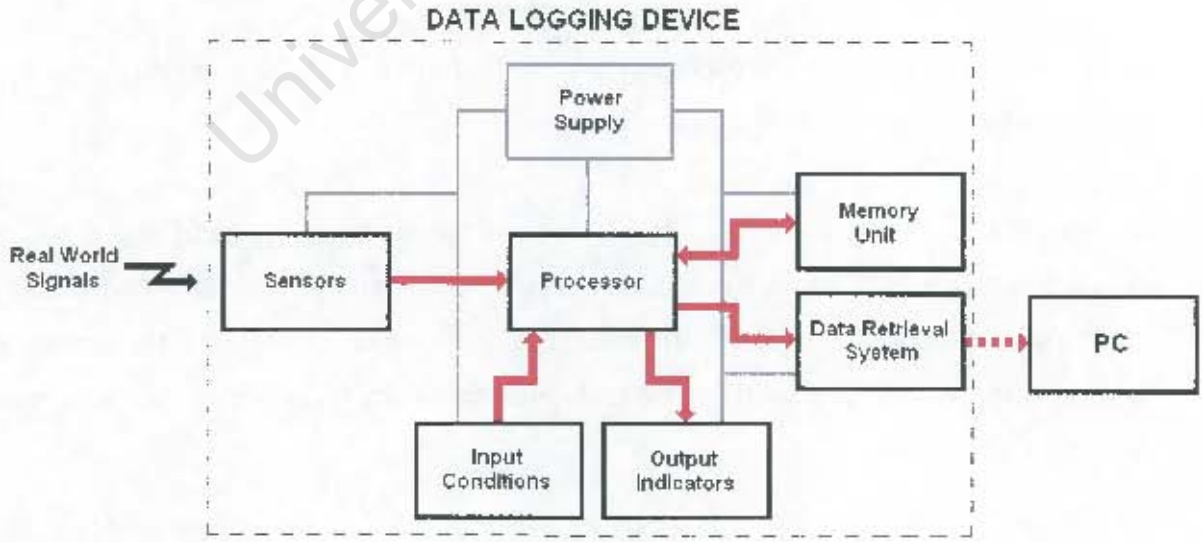



**Figure 5.1: A Simplified System Diagram of a Data Logging Device**

A data logger is used to measure, record and store a parameter or parameters. Data are measured with sensors, analysed by the processor and stored in some form of memory unit. Data can be uploaded to a personal computer through a data retrieval system. The power supply unit powers all the subsystems.

### *5.1.2 Sensor and Data Capture Requirements*

The vibration data logging prototype board must be able monitor the condition of the idlers and their bearings to determine their state of wear. The following three parameters were selected as wear indicators:

1. *Vibration* – As described in Chapter 3, mechanical vibration of the bearing and bearing housing is a good indicator of bearing defects. Due to the position of the board any vibrations coming from failing idler bearings will be damped as they pass through the belt, as rubber is a damping medium. The belt will experience many different vibrations and stresses during the operation of the conveyor such as: small random vibrations from the movement of the transported material; vibration spikes as the belt travels over idlers; and transient stresses during starting and stopping of the conveyor; amongst others (Section 4.1.1). Signal damping and surrounding environmental vibrations will hide small vibration signals emanating from failing bearings. Transforming a signal from the time to the frequency domain can counteract this by isolating the bearing frequencies from surrounding noise. However, the frequency of a signal from a vibrating source, in this case the bearing, will still be subjected to a Doppler shift relative to a moving device. This will cause some frequency modulation of the bearing frequencies which will complicate analysis using high frequency resonance techniques (Section 3.2.2).

   It is assumed in this project that only late damage will be able to be successfully detected by time and frequency domain methods as these techniques are unable to distinguish the small vibration signals that occur during early damage from environmental noise. This method will still provide maintenance managers with enough early information to stop potential belt failure due to faulty idlers. The vibration data logging board's mechanical vibration sensor design takes these limitations into account.

   A time based technique, in this case RMS vibrational acceleration, can be used to measure mechanical vibration. This technique has been selected as it is simple to implement; is not affected to the same extent by conveyor movement as frequency measurements; and has been shown to be successful in detecting late damage in bearings **[19]**.

As a belt moves over an idler, it experiences horizontal and vertical displacement relative to the belt's motion (Fig. 5.2). This change in the direction and velocity of the belt results in acceleration in both horizontal and vertical directions.

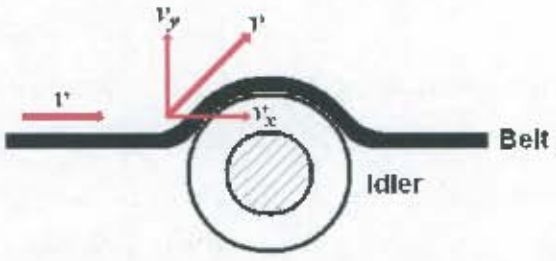



**Figure 5.2: Movement of Conveyor Belt Around an Idler**

As a belt moves over an idler, it experiences a vertical and horizontal displacement relative to the belt's movement. This change in the direction and velocity of the belt results in acceleration in both vertical and horizontal directions.

A faulty idler bearing may seize or cause excessive idler vibration resulting in increased horizontal and vertical acceleration of the belt. Vibrational acceleration must therefore be measured in the direction of the belt travel, represented by *y* in Figure 5.3 and vertical to the belt, represented by *z*.

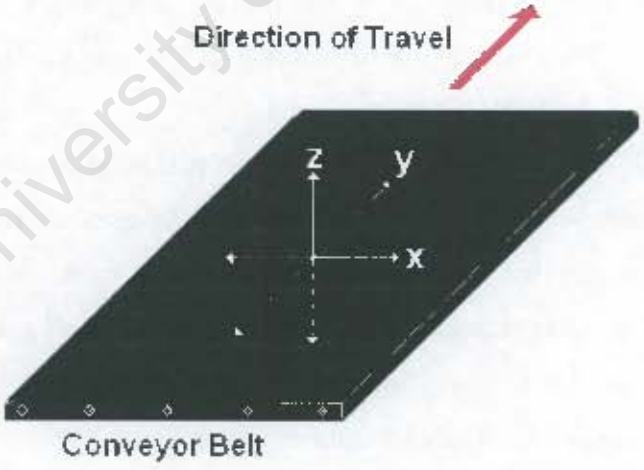



**Figure 5.3: Directions of Motion on the Belt**

Vibrational acceleration must be measured in the direction of the belt travel (*y*) and vertical to the belt (*z*).

In this study, an ADXL202E ± 2g Dual-Axis Accelerometer **[45]**, is used as the transducer to pick up mechanical vibration. This is discussed in more detail in Section 5.3.

2. *Sound* – This parameter can be used as an indicator of advanced bearing wear as a badly worn bearing may create an audible sound. An electret condenser microphone is used to pick up sound (Section 5.4).

3. *Temperature* – The temperature of the belt will increase as it begins to fail due to seized or badly worn idlers. Temperature is therefore a parameter that can be used as an indicator of idler failure. A MAX6608 low voltage analogue temperature sensor [46], is used to measure temperature (Section 5.5).

Each of the three parameters requires a different sampling rate as the transducers each have different maximum frequencies that they can measure. The maximum frequencies and the required sampling rates are given in Table 5.1. The maximum frequencies of the signals have been given once they have passed through anti-aliasing filters.

**Table 5.1: Sampling Frequencies for Each Transducer**

Each transducer requires a different sampling frequency as they each have different maximum frequencies. The selection of sampling frequency must satisfy the Nyquist-Shannon Criterion (Eq. 5.1). The maximum frequencies shown in this table are for the signals once they have passed through anti-aliasing filters

| TRANSDUCER | MAXIMUM FREQUENCY (kHz) | SAMPLING FREQUENCY (kHz) | DESCRIPTION |
|---|---|---|---|
| *ADXL202E* | 5 | 32 | This is a dual-axis accelerometer and the -3 dB bandwidth at pins $x_{fb}$ and $y_{fb}$ is 6 kHz. The sampling frequency at each pin must be at least 12 kHz (Eq. 5.1) and is selected to be 16 kHz per channel for ease of calculation. |
| *Electret Condenser Microphone* | 10 | 20 | The bandwidth of an electret microphone is typically 50 Hz – 13 kHz. The sampling frequency is selected to be 20 kHz which satisfies the Nyquist-Shannon Criterion. |
| *MAX6608* | DC | Every 10 seconds | The temperature sensor takes a temperature reading every 10 seconds. |

Anti-aliasing filters must be used on the output signals of the accelerometer and electret condenser microphone before they enter the ADC to limit high frequency noise and ensure that aliasing of the signal does not occur. Aliasing arises when a continuous signal is sampled

at too low a frequency causing the resulting signals to be indistinguishable from one another. The signal therefore cannot be properly reconstructed from its samples.

It is important to select a high resolution ADC to limit the quantisation error in a signal once it has been digitised. The quantisation error is the difference in value between the actual analogue signal value and the approximated digital value of that signal once it has been converted. The 12-bit analogue to digital converter onboard the dsPIC30F6014A digital signal controller [47] was selected for use on the vibration data logging board, providing a resolution of 806 $\mu$V for a 3.3 V supply.

### 5.1.3 *Processing Requirements*

The processor controls all the system functions in a device and performs all the necessary calculations on the measured data. It must be fast enough to handle all these processes. A processor can have extra onboard peripheral modules, such as an analogue to digital converter (ADC), a universal asynchronous receiver transmitter (UART) or a serial peripheral interface (SPI). This simplifies the programming process as the required peripheral registers and function libraries already exist and do not have to be constructed. The processor in this design must have digital signal processing capabilities as this simplifies the programming of more complicated mathematical techniques, such as Fast Fourier Transforms (FFTs) and moving average calculations.

A dsPIC30F6014A digital signal controller [47] was selected as the processor for the vibration data logging board as it had: a large selection of onboard peripheral modules; the capability to perform a variety of digital signal processing calculations; a relatively low cost; a large program memory storage capacity (144-kbytes); numerous input/output (I/O) pins; and a maximum processing speed of 20MIPS (million instructions per second) for a supply voltage of 3.3V (Section 5.6).

### 5.1.4 *Memory and Data Storage Requirements*

The various memory systems in a data logger are required to store the program used to run the system, and to hold all the captured data. These systems must have a large enough capacity in order to fulfil these requirements.

The program memory is stored in the processor unit which must have a suitable memory size. The program memory must be non-volatile, in other words, the data must not be lost if there is no power supplied to the device.

The collected data must be stored in a non-volatile external memory device. The resolution and sampling rate of the signals will determine the size of the memory. There are a number of non-volatile memory devices. These include battery powered static RAM, EEPROM and Flash. A 32-Mbit Serial Dataflash® chip AT45DB321C [48] was chosen because of its high-density, low power consumption, low cost, and reduced number of data and address lines. The capacity was selected based on calculations found in Appendix D and it provides approximately sixteen minutes of data recording time. The specifications for the belt conveyor CV-08 were used in the calculations (Section 5.7).

### *5.1.5 Data Retrieval Requirements from the Data Logging Device*

The data which are collected in a data logging system must be transferred to a PC for further analysis. One of the requirements in this design is to use some form of serial link between the device and the PC. A serial data link has fewer connections than an equivalent parallel link. The data are transferred bit by bit down one data line as opposed to having a data line for every bit as is the case with a parallel connection. This reduces the complexity of the connection between the two communicating devices at cost of communication speed. An asynchronous serial link requires no clock line which further reduces the number of connections between devices, although the transfer speed must be set in both devices before communication commences. A RS-232 link was selected as it is an asynchronous serial protocol which is often used when an external device is linked to a personal computer's serial communication port. This requires that all data signals are converted to the RS-232 standard before they are sent down the data line. This is achieved with a SP3222E true −3.0V to −5.5V RS-232 Transceiver [49].

The data must be transferred at the fastest possible rate. The maximum transfer rate that could be handled by the device is 57600 baud which means that 32-Mbits of data will be transferred to a PC in approximately nine and a half minutes (Section 5.6).

### 5.1.6 *Input Parameters and Output Indicators Requirements*

The vibration data logging board must be able to record, send and erase data. A user must be able to select when the device performs each of these operations. A dip switch with four switches was used as the input to the dsPIC30F6014A processor on the prototype board. Only three of the four switches were used to select the input parameters.

It is always useful on any prototyping board to have some form of visual indication of the onboard processes. The user of the vibration data logging device needs to know when the board is recording, sending or erasing data and three light emitting diodes (LEDs) are used for this purpose. Four extra light emitting diodes are also provided as general purpose indicator lights.

### 5.1.7 *Power Requirements of the Device*

The vibration data logging prototype board is a stand alone device which means that it must be battery powered. A power supply must have some form of voltage regulation which will provide a constant output voltage despite fluctuations of the input. Battery life must be maximised by minimising the amount of current drawn from it while still providing enough power to the device. This can be achieved by using some form of switch mode power supply which is discussed in more detail in Section 5.8.

Most digital integrated circuits (ICs) now run off 3.3V as opposed to the traditional +5V, thus the power supply on board the device must provide an output voltage of +3.3V. A 3V lithium coin cell is used with a LM2623 DC/DC Boost Converter **[50]** to produce +5V which is then converted to the required +3.3V using two Low Voltage Dropout Converters, the TC1264 **[51]** and the TC2185 **[52]** which provide power for the digital and analogue circuits.

### 5.1.8 *Physical Requirements of the Device*

The devices circuits must be made on a printed circuit board (PCB) as this makes the device more robust. The digital and analogue circuitry must be isolated from each other in order to

prevent interference between the signals. The accelerometer circuit must be constructed on a separate board as its axes must be vertical and parallel to the main board.

## 5.2 A General Overview of the Vibration Data Logging System

The vibration data logger was designed as a prototype board which could be used to monitor the condition of failing bearings. It was not designed to be embedded in an actual belt but provide a guideline of a possible system that could be used. A complete system diagram of the vibration data logging prototype board is illustrated in Figure 5.4.

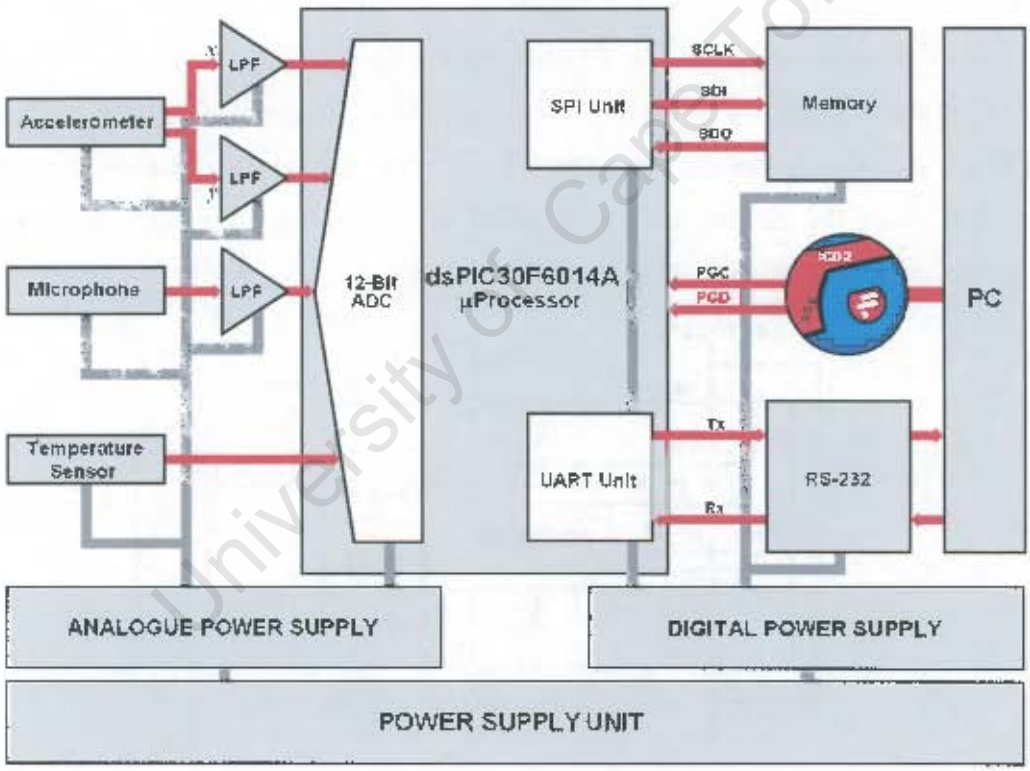



**Figure 5.4: The System Diagram of the Vibration Data Logging Board**

The vibration data logging prototype board has three sensors which measure mechanical vibration, sound and temperature. These analogue parameters are filtered to reduce high frequency noise and prevent aliasing. They are then converted to digital values by a 12-bit ADC which is built into the dsPIC30F6014A digital signal controller. The dsPIC30F6014A analyses the data and stores it in an Atmel 32-Mbit Dataflash® chip via an onboard serial peripheral interface. The data can then be uploaded to a personal computer via a universal asynchronous receiver transmitter interface.

The complete circuit diagrams of the vibration data logging prototype board can be found in Appendix F.

## 5.3 The Vibration Sensor

### *5.3.1 An Introduction to the ADXL202E*

Accelerometers are transducers designed to measure vibrational acceleration and are often used in vibration monitoring systems. There are different types of accelerometers. These include: piezoelectric, integral electronics piezoelectric, piezoresistive, variable capacitance and servo force balance **[53]**.

An ADXL202E ± 2g Dual-Axis Accelerometer was used as the transducer to monitor vibration on the vibration data logging board. It is an integrated Micro Electro Mechanical System (iMEMS®) and consists of a micro machined spring-mass system and a variable capacitor. The mass is connected to a movable central plate which lies between two fixed plates of a differential capacitor. The fixed plates are driven by an 180° out of phase square wave. The mass is deflected when the accelerometer experiences acceleration causing the central plate to move, producing an output square wave whose amplitude is proportional to the acceleration of the device. It is then possible to use phase sensitive demodulation to establish the direction of the acceleration **[45]**.

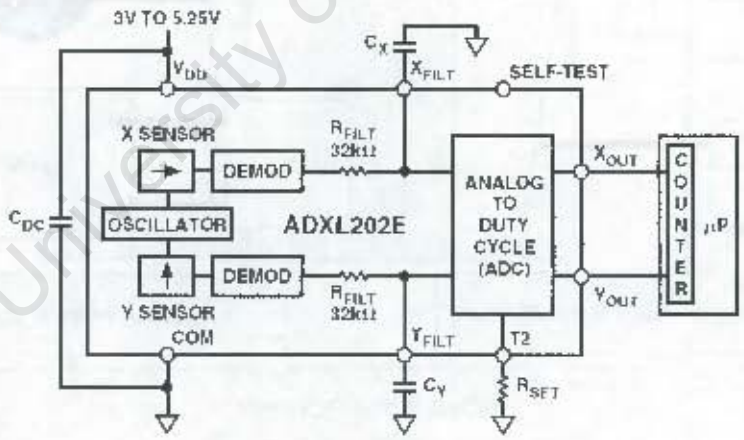



**Figure 5.5: The ADXL202E Functional Diagram**

The ADXL202E has both analogue and digital outputs. The analogue outputs are obtained at the $x_{fil}$ and $y_{fil}$ pins. A 32kΩ internal resistor together with an external capacitor set the bandwidth of the accelerometer. The period of the pulse width modulated square wave is set by a resistor connected to Pin 2 (Taken from Analog Devices Incorporated Data Sheet, 2000) **[45]**.

The ADXL202E provides two output options: an analogue voltage or a digital pulse width modulated (PWM) signal whose duty cycle is proportional to acceleration. It can measure acceleration in two directions and their orientations are shown in Figure 5.6.

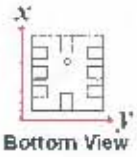



**Figure 5.6: The Orientations of the ADXL202E Acceleration Axes**

The ADXL202E can measure acceleration in two directions. The orientation of these axes is shown from the bottom side of the chip.

It comes in a 5 x 5 x 2 mm 8-lead hermetic leadless chip carrier (LCC) package and can be powered off of +3V to +5.25V.

### 5.3.2 *A Description of the Vibration Sensor Circuit used in the Vibration Data Logging Device*

The ADXL202E circuit (Fig. 5.7) is powered off the +3.3V analogue supply (AVDD) generated on the main vibration data logging board and is connected to the analogue ground plane (AGND). A small 75Ω ferrite bead was inserted in the supply line together with decoupling capacitors to form a low-pass filter to reduce any high frequency noise on the power supply. The accelerometer circuit was designed according to the procedures laid out in the ADXL202E datasheet [45].

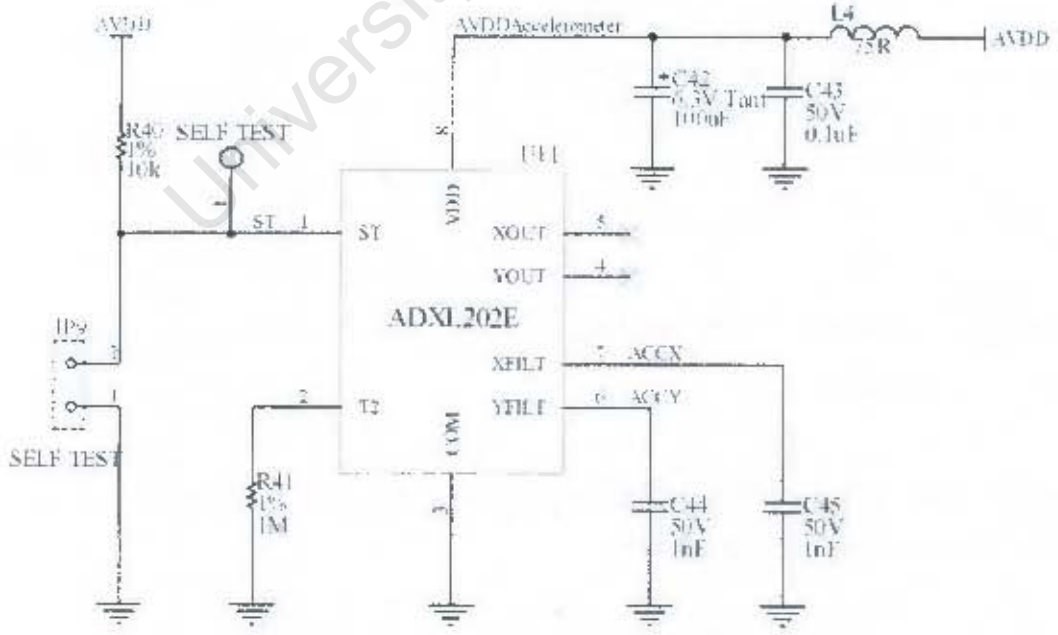



**Figure 5.7: The Vibration Sensor Circuit**

An ADXL202E ± 2g Dual-Axis Accelerometer was used as the transducer on the vibration data logging board to monitor vibration.

As mentioned in Section 5.3.1, the ADXL202E has both analogue and digital outputs. The analogue output was selected as it provides a bandwidth of up to 6 kHz as opposed to 500 Hz of the digital PWM output. This will provide enough bandwidth to measure the mechanical vibrations of faulty bearings. However, this did have the disadvantage of added complexity on the filtering and analogue to digital conversion side.

The analogue outputs are obtained at the $x_{filt}$ and $y_{filt}$ pins. A 32kΩ internal resistor together with an external capacitor forms a passive low-pass filter which sets the bandwidth of the accelerometer (Eq.5.2).

$$f_c = \frac{1}{2\pi RC} \tag{5.2}$$

$f_c$ is the cut off frequency, $R$ is the value of the resistor and $C$ is the value of the capacitor in the RC filter circuit.

A 1nF ceramic capacitor was selected to set the devices bandwidth to 5 kHz using Equation 5.2.

$$f_c = \frac{1}{2\pi RC}$$

$$C = \frac{1}{2\pi R \cdot f_c}$$

$$= \frac{1}{2\pi \cdot (32\times10^{3})(5\times10^{3})}$$

$$= 0.9947nF$$

$$\approx 1nF$$

The analogue outputs must be buffered by a voltage follower system as they are not designed to drive a load directly.

The period of the pulse width modulated square wave is set by a resistor connected to Pin 2 and is calculated with Equation 5.3 **[45]**.

$$T_2 = \frac{R_{out}(\Omega)}{125M\Omega} \tag{5.3}$$

$T_2$ is the PWM period and $R_{set}$ is the value of the resistor connected to Pin 2 in Ohms.
The resistor is necessary even if the digital output is not required. A 1MΩ resistor was connected to this point giving a PWM period of 8 ms.

$$T_2 = \frac{R_{set}}{125M\Omega}$$
$$\frac{1M\Omega}{125M\Omega}$$
$$8ms$$

The ADXL202E has a self test pin (Pin 1) which can be used to test the functioning of the chip. When it is connected to AVDD it causes a deflection of 800mg. In this system a jumper (JP9) is used to test the ADXL202E. When the jumper is connected, the self test function is activated. The self test circuit is located on the main vibration data logging board.

Vibrational acceleration must be measured vertical and parallel to the vibration data logging board. The ADXL202E circuit had to be built on a separate PCB in order to achieve this as the axes are parallel to the chip. It was connected using a 10 x 2 pin right angled connector which fits in socket on the main board. It can be removed when it is not needed.

The outputs from the accelerometer circuit are connected to anti-aliasing filters before entering an inverting amplifier, biased at 1.65V, with a gain of 2.2. The filtering system consists of a 5-pole low-pass Bessel Filter with a cut off frequency 5 kHz (Fig. 5.8). A Bessel filter was selected because its performance in the time domain is very good as it provides a constant time delay in the pass-band which corresponds to a linear phase-shift for all frequencies in the frequency domain [54].

The first pole of the anti-aliasing filter is the RC output stage of the accelerometer, connected to a voltage follower circuit. This is then followed by a 4-pole Sallen-and-Key low-pass Bessel filter. It is possible to amplify the signal in the filter by adding resistors to each stage to create non-inverting amplifiers. However, it was decided that the signals were to be amplified by a separate inverting amplifier which meant that the gain of the circuit could be altered simply by changing the value of two resistors as opposed to all of the resistors and capacitors in the filter stages. This made gain altering easier.

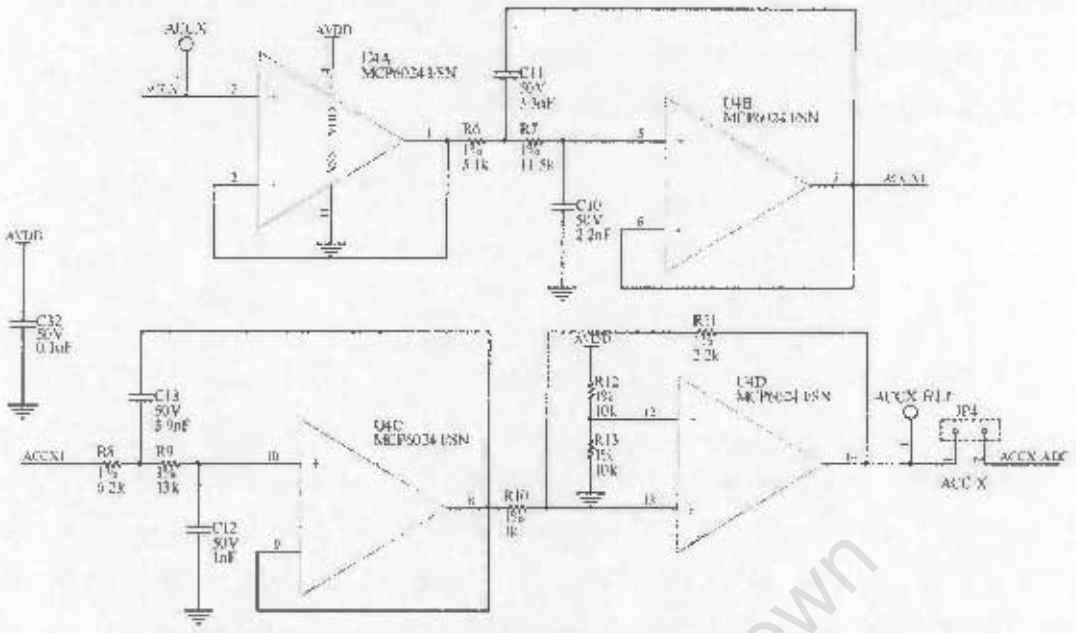



**Figure 5.8: The Anti-Aliasing Filters used on the Outputs from the ADXL202E**

5-pole Bessel low-pass filters are used on the outputs of the ADXL202E to filter out high frequency noise components and prevent aliasing. A Bessel filter was selected as it performs well in the time domain.

The values of the resistors and capacitors where calculated using the standard Sallen-and-Key low-pass filter equations (Eq. 5.4 and Eq. 5.5) and the Bessel Filter table found in Appendix E **[55]**.

$$f_c = \frac{1}{FSF \cdot 2\pi\sqrt{R_1 R_2 C_1 C_2}} \tag{5.4}$$

$$Q = \frac{\sqrt{R_1 R_2 C_1 C_2}}{R_1 C_1 + R_2 C_2} \tag{5.5}$$

$f_c$ is the required cut off frequency, $FSF$ is the frequency scaling factor, $Q$ is the quality factor, and $R_i$ and $C_i$ are the resistors and capacitors making up the Sallen-and-Key Filter Circuit.

The outputs of the filter are connected to the processor via jumpers JP4 and JP5, which means that they can be disconnected when they are not needed or for circuit troubleshooting. Test pins are provided on the signals ACCX and ACCY which come from the accelerometer, and ACCX_FILT and ACCY_FILT which are the outputs of the anti-aliasing filters. An oscilloscope probe can be connected to these points to view the signals.

The anti-aliasing filter is constructed using Microchip's MCP6024 operational amplifiers [56]. These were selected as they offer single supply operation down to a voltage as low as 2.5V, have a gain bandwidth product of 10 MHz and a rail-to-rail output swing. They contain four operational amplifiers per chip which reduces the size of the circuit on the PCB and come in surface mount 14-lead plastic small outline (SOIC) 150 mil packages.

## 5.4 The Audio Sensor

### *5.4.1 An Introduction to Electret Condenser Microphones*

Microphones convert sound into electrical energy. There are many different types of microphones most of which use some form of flexible diaphragm. Condenser microphones use capacitance to convert sound waves into electrical signals. The diaphragm acts as one plate of a capacitor and vibrations caused by sound cause the distance between the plates to vary. The plates are kept at constant charge and therefore the voltage across the capacitor changes as the distance changes. Electret condenser microphones have a dielectric in between the capacitive plates. The dielectric is given a permanent charge and is known as an electret. These microphones do not require power to polarise the plates, however they do often contain a preamplifier which requires power.

### *5.4.2 A Description of the Audio Sensor Circuit used in the Vibration Data Logging Device*

An electret condenser microphone is used on the vibration data logging board to pick up audible noise. The microphone is connected to a stereo jack plug which can be inserted into the stereo socket on the board. The socket has three connections: bias, signal and ground.

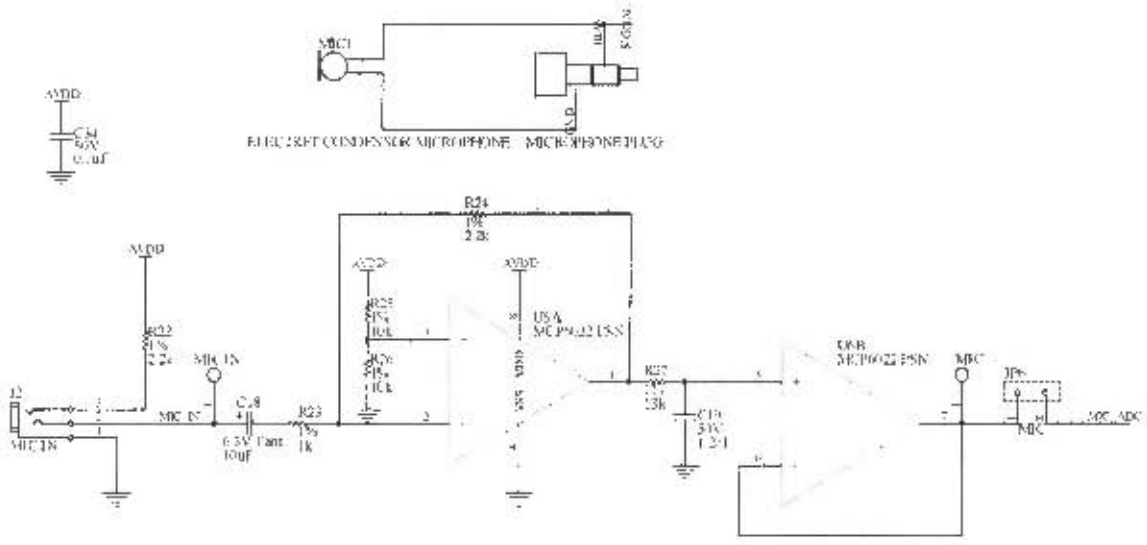


**Figure 5.9: The Audio Sensor and Anti-aliasing Filter Circuit**

An electret condenser microphone was used as the audio sensor onboard the vibration data logger and is connected to the board via a stereo socket. The output signal is amplified, filtered and then buffered before it enters the ADC of the processor.

The microphone's preamplifier is powered by +3.3V (AVDD) connected via a current limiting 2.2kΩ resistor. The ground signal is connected to the analogue ground plane (AGND). The output signal from the electret condenser microphone is AC coupled to an inverting amplifier, biased at 1.65V, with a gain of 2.2 by a 10$\mu$F capacitor. The capacitor lets the AC signal pass through it while blocking the DC bias voltage. The signal then passes into a single-pole low-pass filter with a cut off frequency of 10 kHz. It is then buffered before it enters the ADC module of the dsPIC30F6014A.

The amplifying and buffering circuit is constructed using MCP6022 dual operational amplifiers [**56**].

The amplified and filtered output signal is connected to the processor via a jumper JP6 which means it can be disconnected if necessary. Test pins are provided on signals MIC IN, the output of the microphone, and MIC_ADC which is the amplified and filtered output.

## 5.5 The Temperature Sensor

Temperature sensors take many forms. An analogue temperature sensor is an integrated circuit whose output voltage is proportional to temperature.

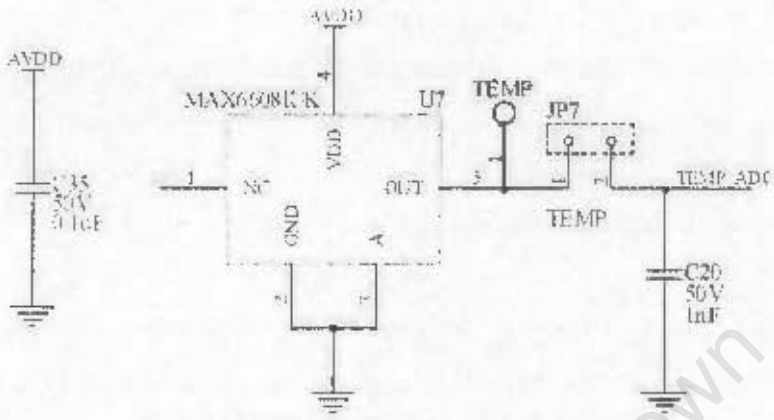



**Figure 5.10: The Temperature Sensing Circuit**

A MAX6608 low voltage analogue temperature sensor is used to measure the temperature of the vibration data logging board. It senses its own die temperature and outputs a voltage proportional to that temperature.

A MAX6608 low voltage analogue temperature sensor **[46]** is used to measure the temperature of the vibration data logging board. It senses its own die temperature and can accurately sense temperature through its leads as these provide a good thermal path to the IC. The transfer function of the chip is given in Equation 5.6 where 0°C equals 500 mV and there is a change of 10 mV/°C. $T$ is the die temperature.

$$V_{out} = 500\text{mV} + \left(T \times 10\text{mV/°C}\right) \tag{5.6}$$

The MAX6608 circuit is setup as suggested in the datasheet **[46]**. The output of the MAX6608 is filtered with a 1nF ceramic capacitor. This removes any AC signals on that line. It is connected to the processor via a jumper JP7 and a test point is provided on the signal TEMP_ADC which is the output of the device (Fig. 5.10).

The MAX6608 comes in 5-lead Small Outline Transistor Package (SOT-23)

## 5.6 The Digital Signal Processor

### *5.6.1 An Introduction to the dsPIC30F6014A Digital Signal Controller*

Digital signal controllers are specialised microprocessors with digital signal processing capability. Microchip's dsPIC30F6014A general purpose digital signal controller has 16-bit modified Harvard Architecture which means that it uses separate storage and signal pathways for instructions and data. This has the advantage that instructions can be implemented and data accessed at the same time. The dsPIC30F6014A contains 144 Kbytes of Flash program memory, 4 Kbytes of data EEPROM and 8 Kbytes of RAM.

The DSP engine contains a 17-bit x 17-bit multiplier, 40-bit arithmetic logic unit (ALU), two 40-bit saturating accumulators and a 40-bit bi-directional barrel shifter [47]. These units make DSP operations more efficient.

The dsPIC30F6014A contains a number of peripheral modules which simplify design and programming. These include: a 12-bit ADC, timer units, SPI modules and UART modules amongst others.

The dsPIC30F6014A can be powered off a +2.5V to a +5.5V supply. The supply voltage will affect the speed of the processor and a dsPIC30F6014A powered with +3.3V can reach a maximum speed of 20MIPS.

The dsPIC30F6014A comes in a 14 x 14 x 1 mm 80-lead plastic thin quad flat package (TQFP).

Microchip's dsPICDEM™ 1.1 Development Board [57] was used to learn about the dsPIC30F6014 processor during the design phase of this project. It contains a dsPIC30F6014 processor which is connected to a variety of different devices such as light emitting diodes (LEDs), potentiometers, push buttons, RS-232 devices, a CODEC and an external microcontroller which connects to an LCD display. The device was programmed using the ICD2 In-Circuit Debugger which is connected to a PC with a USB connector. It provides in-circuit debugging from the MPLAB® graphical user [58].

### 5.6.2 *A Description of the Processing Circuit used in the Vibration Data Logging Device*

The dsPIC30F6014A processor circuit was designed based on the system integration suggestions found in the dsPIC30F6014A datasheet [47] and the circuit diagrams for the dsPICDEM™ 1.1 Development Board [56].

The dsPIC30F6014A processor has five positive power supply and five ground connections, all of which must be connected. Each side of the chip has a digital supply and a ground connection. These are connected to the +3.3V digital supply (VDD) and the digital ground plane (GND) respectively. The remaining two power supply connections provide power to the 12-bit analogue to digital converter module and are connected to the +3.3V analogue supply (AVDD) and analogue ground plane (AGND). This ensures that the high speed digital circuits do not interfere with the analogue signals.

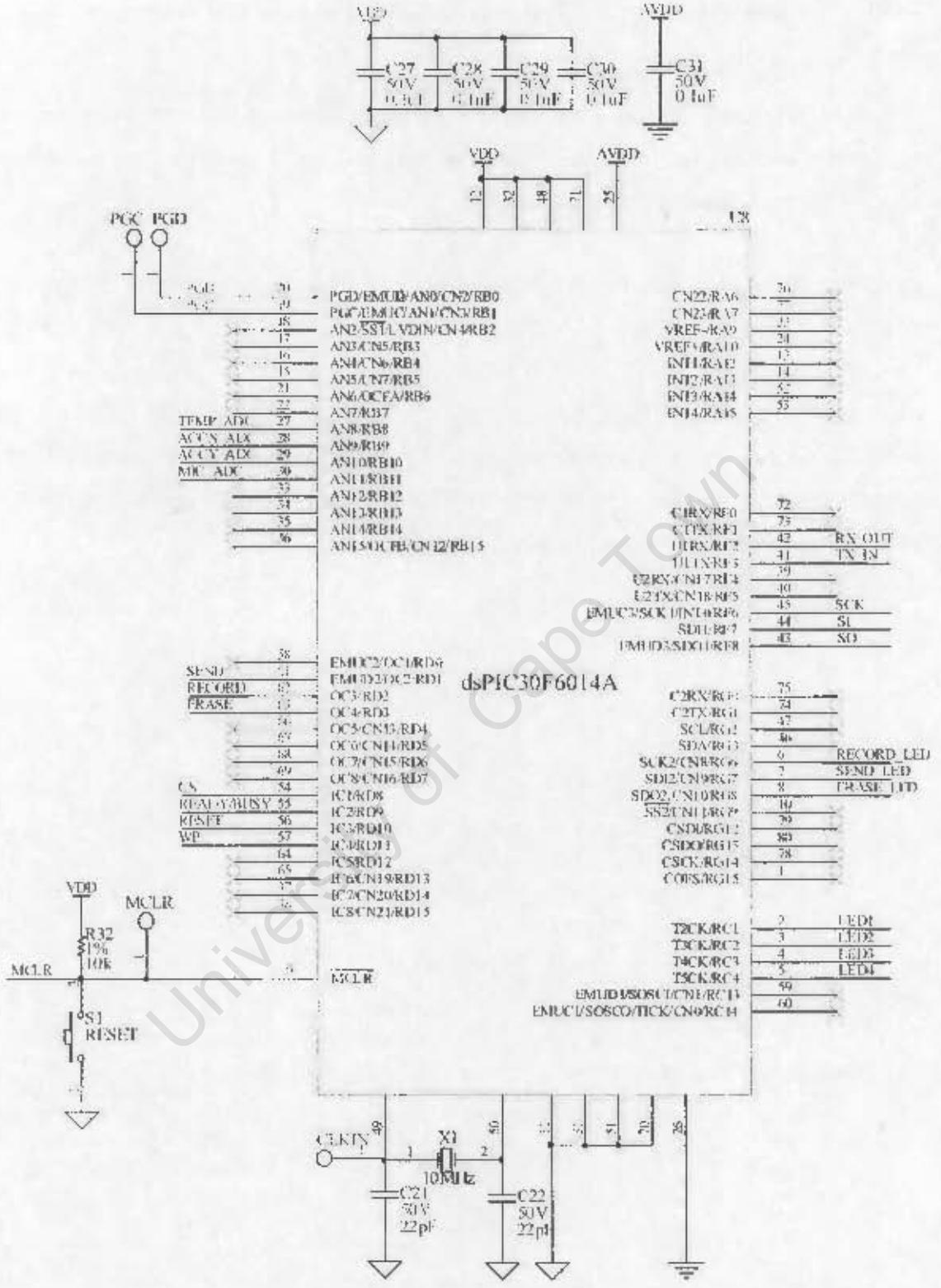



**Figure 5.11: The Processor Circuit**

A dsPIC30F6014A digital signal controller is used as the processing device onboard the vibration data logging board. All the connections to the processor are shown in the diagram above.

A 10 MHz crystal is connected across pin 49 (OSC1) and pin 50 (OSC2). A 22pF ceramic capacitor is connected from each pin of the crystal to ground (GND). This together with an internal feedback resistor and an inverter in the dsPIC30F6014A form a crystal oscillator circuit which produces the system clock signal (Figure 5.12).

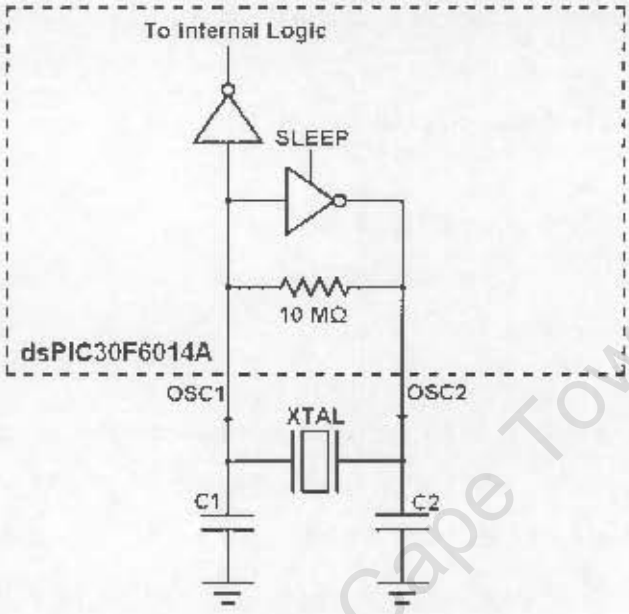



**Figure 5.12: The Crystal Oscillator Circuit**

A crystal oscillator consists of a quartz crystal, two capacitors, a feedback resistor and an inverter. The dsPIC30F6014A's clock signal can be generated using the circuit above.

The dsPIC30F6014A contains an internal phase lock loop (PLL) circuit which can be used to multiply the primary oscillator's frequency by 4, 8 or 16. In this circuit the clock mode is set to XT w/PLL x 8. A test pin is connected to OSC1 and can be used to view the clock signal or to connect an external clock.

The processor is programmed using an ICD2 in-circuit debugger module. The device is programmed through pin 19 (PGC) and pin 20 (PGD). These lines together with the devices master reset, pin 9 ($\overline{\text{MCLR}}$) are connected to the ICD2 in-circuit debugger module using a 6 pin RJ-12 right angled modular jack plug. An external push button is connected via a 10kΩ pull-up resistor to the $\overline{\text{MCLR}}$ signal line and can be used to reset the device. Test points are provided on the PGC, PGD and $\overline{\text{MCLR}}$ signal lines.

The dsPIC30F6014A contains a 12-bit analogue to digital converter module which uses a successive approximation register (SAR) to convert signals. All sixteen ADC inputs are multiplexed into a sample and hold register which is the input to the converter which generates the result [47]. The analogue supply lines are used as the voltage references in conversions. The outputs from the sensors are connected to the 12-bit ADC at the following locations: the temperature sensor is connected to pin 27 (AN8), the outputs from the accelerometer circuit are connected to pin 28 (AN9) and pin 29 (AN10), and the output from the microphone circuit is connected to pin 30 (AN11).

A dip switch provides three input signals which are used as SEND (pin 61 – RD1); RECORD (pin 62 – RD2) and ERASE (pin 63 – RD3) signals. These are used to start the recording, sending and erasing processes.

Seven light emitting diodes (LEDs) are used as indicator lights onboard the vibration data logging board. A green LED (pin 7 – RG7) is used to indicate SEND, an orange LED (pin 8 – RG8) is used to indicate ERASE and a red LED (pin 6 – RG6) is used to indicate RECORD. The four remaining red LEDs are general purpose and used in the testing of the board. They are connected to pins 2-5 (RC1-RC4). All LEDs have current limiting 1kΩ resistors.

The dsPIC30F6014A is connected to a 32-Mbit Dataflash® chip, AT45DB321C. These two devices communicate using a specific synchronous serial connection known as the serial peripheral interface (SPI). An SPI has 3-wires: serial data input (SDI); serial data output (SDO); and serial clock (SCK). A serial connection means that data are transmitted bit by bit down a line. A clock signal (pin 45 – SCK1) is connected between the devices in order to synchronise data transfer. The device that provides the clock signal and controls the data transfer is known as the master while the other device is known as the slave. The dsPIC30F6014A is the master device in this case. The SDI pin from the dsPIC30F6014A (pin 44 – SDI1) must be connected to SDO pin on the AT45DB321C and the SDO pin from the dsPIC30F6014A (pin 43 – SDO1) must be connected to SDO pin on the AT45DB321C. Test pins are provided on SDI1, SDO1 and SCK.

The dsPIC30F6014A contains two universal asynchronous transmitter receiver (UART) modules. Data are retrieved from the vibration data logging board to a PC via one of these

modules. UART is a serial interface with separate lines for transmission and reception. As no clock line is connected between devices it is important that the transfer speed is set in both devices before transmission takes place. Data are transferred at a rate of 57600 baud, with 8 data bits, one stop bit and no parity bit.

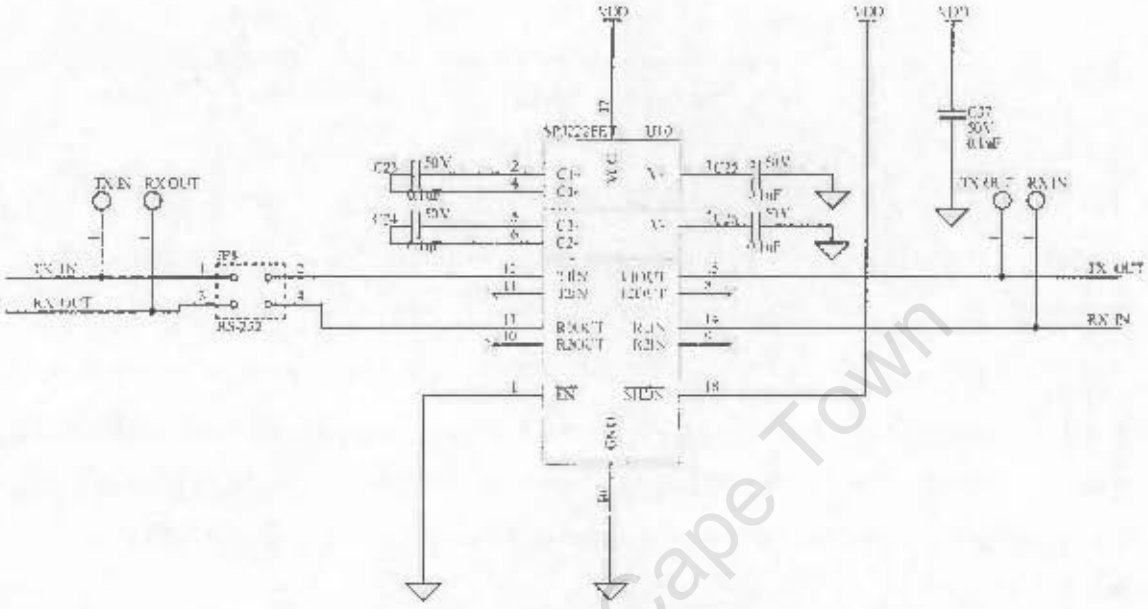



**Figure 5.13: The Data Retrieval Circuit**

Data are retrieved from the vibration data logging board via a universal asynchronous transmitter receiver (UART) link. The UART module onboard the dsPIC30F6014A is connected to a SP3222 E RS-232 transceiver which is in turn connected to a 9-pin female serial communication port which can be connected to a PC.

The dsPIC30F6014A's UART module is connected to a SP3222E true +3.0V to +5.5V RS-232 Transceiver [49] via TX_IN (pin 41 – U1TX) and RX_OUT (pin 42 – U1RX). This RS-232 system converts a +3.3V logic one to -5V and a 0V logic zero to +5V using a series of charge pump converters. The SP3222E is connected to a right-angled 9-pin female serial communication port with signal lines TX_OUT and RX_IN.

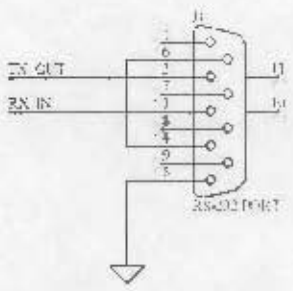


**Figure 5.14: Serial Communications Port Connections for a RS-232 link to a PC**

Data are sent to the PC on pin 2 and received from the PC on pin 3. Pins 4 and 6 are linked together and pin 5 provides the ground signal.

The SP3222E chip was selected as it can run off of a +3.3V supply and can work with +3.3V digital logic levels. It was constructed according to the datasheet specifications [49]. It is powered with the +3.3V digital supply (VDD) and is connected to the digital ground plane (GND). The disable pin 1 ($\overline{EN}$) is connected directly to ground and the do not shutdown pin ($\overline{SHDN}$) is connected directly to +3.3V. Four 0.1$\mu$F ceramic capacitors are used in the charge pump converters. The TX_IN and RX_OUT signals are connected to the device via a jumper (JP8) and test pins are provided on the following signal lines; TX_IN, RX_OUT, TX_OUT and RX_IN.

## 5.7 The Memory System

### *5.7.1 An Introduction to the AT45DB321C 32-Mbit Dataflash® Chip*

Memory devices come in many forms, some of which are volatile which means they lose their data when no power is received, or non-volatile which means they store their data even in the absence of power. Flash memory is a type of non-volatile memory which is electrically programmed and erased in blocks. It provides a cheaper alternative to EEPROM. Typical Flash devices use a parallel interface and address lines to program and read data.

AT45DB321C 32-Mbit Dataflash® chip is a serial flash chip which can be interfaced to a processing device using a 3-wire SPI. The AT45DB321C's memory is organised into 8192 pages of 528 bytes each. It also contains two 528 byte SRAM buffers which can be used to store data while the main memory is programmed [48]. The Dataflash® chip does not require +5V digital logic levels and can be used with logic levels of +2.7V or +3.3V. It requires a power supply of +2.7 to +3.6V and can operate at a maximum clock rate of 40 MHz.

The AT45DB321C 32-Mbit Dataflash® chip comes in a 28-lead plastic thin small outline package (TSOP).

### 5.7.2 *A Description of the Data Storage Circuit used in the Vibration Data Logging Device*

The AT45DB321C 32-Mbit Dataflash® circuit was designed based on recommendations in the datasheet [48]. It is powered off the −3.3V digital power supply and is connected to the digital ground plane. The devices logic signals are therefore +3.3V (logic one) and 0V (logic zero).

As described in Section 5.6.2, the Dataflash® chip is connected to the dsPIC30F6014A with the SDI, SDO and SCK pins. The SDI pin (pin 6) of the Dataflash® chip is connected to the SDO pin (pin 43 – SDO1) of the processor and the SDO pin (pin 7) is connected to the SDI pin (pin 44 – SDI1) of the processor. The SCK pins of the two devices are linked and this line provides the clock signal for shifting data in and out of the two devices.

The Dataflash® is enabled through the chip select pin ($\overline{CS}$) and a high-to-low transition on this line will begin data transfer between the two devices. The $\overline{CS}$ pin (pin 4) is connected to pin 54 (RD8) of the dsPIC30F6014A.

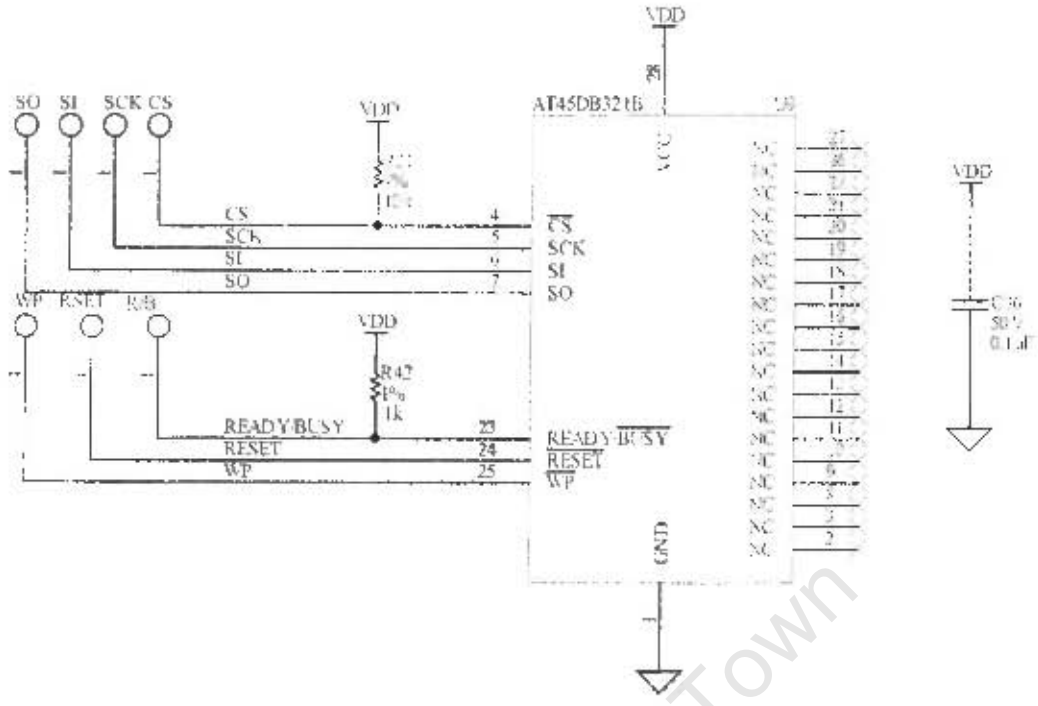



**Figure 5.15: The Memory Storage Circuit**

An AT45DB321C 32-Mbit Dataflash® chip is used as an external data storage chip onboard the vibration data logging board. It is connected to the dsPIC30F6014A using a 3-wire SPI.

The write protect pin ( $\overline{WP}$ ) of the AT45DB321C can be used to prevent the first 256 pages of the main memory from being programmed. The $\overline{WP}$ pin (pin 25) is connected to pin 57 (RD11) of the dsPIC30F6014A and for normal programming operation this pin is pulled high.

The reset pin ( $\overline{RESET}$ ) is used to stop the current operation and reset the internal state of the device to idle. This pin (pin 24) is connected to pin 56 (RD10) of the dsPIC30F6014A and is driven high for normal operation.

The AT45DB321C's READY/$\overline{BUSY}$ (pin 23) indicator line gives the status of the device. This line is pulled low internally when the device is in operation. It is connected to pin 55 (RD9) of the dsPIC30F6014A via a 1kΩ pull-up resistor.

Test pins are provided on all the AT45DB321C's connections to the dsPIC30F6014A.

## 5.8 The Power Supply System

Power supplies are used to provide electrical power to a load or many loads. They come in a variety of forms. The three that were used on the vibration data logging prototype board are discussed in more detail in the following sections.

### *5.8.1 An Introduction to Batteries*

A battery is a device that converts stored chemical energy into electrical energy and consists of one or more voltaic cells. A battery's capacity is measured in ampere hours (A·h) and this means that a battery with a capacity of 1A·h can supply a load with a current of 1A for one hour. Batteries lose charge even if they are not used due to chemical reactions within the cell. These reactions occur at a greater rate as temperature increases and therefore it is important to store batteries in a cool environment.

There are many different types of batteries which use different chemical reactions to store charge. Batteries can be split into two main groups; disposable, which are discarded once they have been used, and rechargeable which can be recharged and used again. There are many different types of disposable batteries. These include: zinc-carbon; zinc-chloride; zinc-air; alkaline; silver-oxide; mercury; and lithium. These different types of batteries each have different charge storing potential and lifetimes.

A 3V lithium cell is used to power the vibration data logging board. This type of cell was selected as it produces twice the voltage of a typical zinc-carbon battery; has a lifetime of approximately 10 years; has a relatively flat discharge curve which means that it maintains a constant voltage as it discharges **[54]** and is made in a 20mm coin cell form which provides a smaller and lower profile on the PCB.

### *5.8.2 An Introduction to Switch-Mode Power Supplies*

A switch-mode power supply (SMPS) uses a saturated transistor or MOSFET to switch the full unregulated voltage across an inductor for short intervals. This causes the inductor's current to build up during each pulse; storing energy in a magnetic field. This energy is then

transferred to a smoothing capacitor at the output. A control circuit alters the pulse width or switch frequency in order to maintain a constant output [54].

Switch-mode power supplies are more efficient than linear regulators as they switch the full supply voltage with a variable duty cycle as opposed to being permanently on. This results in less energy being wasted in the form of heat. They do have the disadvantage that they often contain some switching noise on the output. SMPS can be used as DC/DC voltage converters and these are divided into three main groups:

1. *Step-down Regulators* convert a higher DC voltage into a lower DC voltage. A Buck converter is a type of step-down regulator.
2. *Step-up Regulators* convert a lower DC voltage into a higher DC voltage. A Boost converter is a type of step-up regulator.
3. *Inverting Regulators* invert a DC voltage.

A LM2623 general purpose DC/DC Boost converter [50] is used onboard the vibration data logging board to regulate and step-up +3V from a lithium cell. It was selected as the voltage from the −3V lithium cell has to be stepped up to +5V which is then inputted to two low dropout regulators which regulate the voltage to +3.3V, the voltage that all the integrated circuits on the board are powered. A SMPS maximises the batteries life as it is highly efficient, up to 85% efficiency at an input voltage of +3V. This particular SMPS has few external components and has internal MOSFET switches which simplify circuit design.

This chip is specifically designed for low input voltage applications and can accept an input voltage between 0.8V to 14V and convert it to a regulated output of 1.24V to 14V. It draws a typical operating current of 80$\mu$A which means a battery with a capacity of 100 mA·h would last 1250 hours which is approximately 52 days.

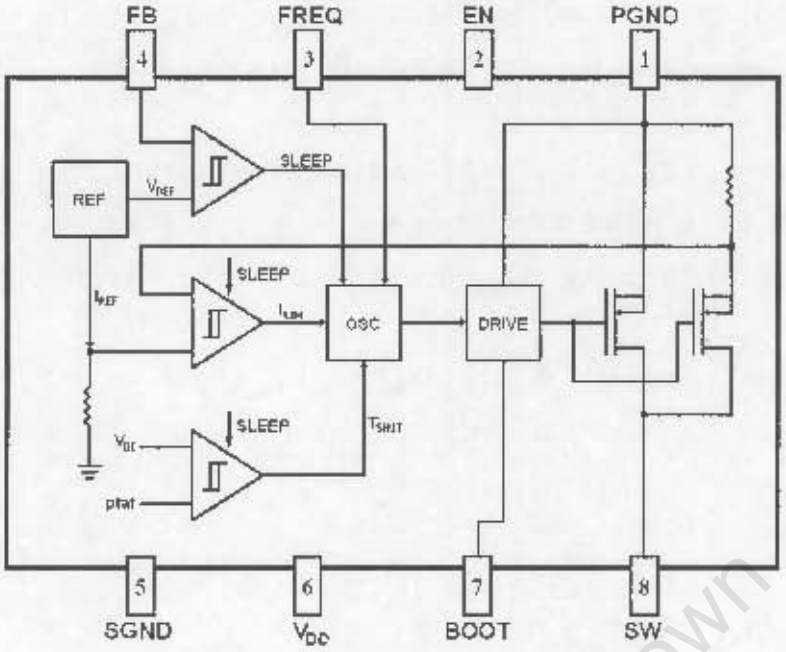



**Figure 5.16: A Functional Diagram of the LM2623 DC/DC Boost Convertor**

The LM2623 is a general purpose gated oscillator based DC/DC Boost converter. It is used onboard the vibration data logging board to convert +3V for a lithium cell to +5V (Taken from National Semiconductor Corporation Data Sheet, 2004) **[50]**.

A functional diagram of the LM2623 is given in Figure 5.16. The LM2623 uses a 'Ratio Adaptive Gated Oscillator Circuit' to compensate for changes in the input/output voltage ratio **[50]**. An internal N-channel power MOSFET is used as the switch in the regulator.

The LM2623 comes in an 8-pin mini small outline (SO) package.

### *5.8.3 An Introduction to Low Dropout Regulators*

A low dropout regulator (LDO) is a type of linear voltage regulator which can have a small difference between its input and output voltages. Linear voltage regulators use some form of transistor or field effect transistor which acts as a variable resistor. A lot of power is dissipated across this device which reduces its efficiency. However, they do not use any form of switching device which means there is significantly less noise on their output as opposed to a SMPS.

Two LDOs are used onboard the vibration data logging board as post regulators from the SMPS. They are used to reduce the ripple on the output of the SMPS so producing stable

−3.3V supplies for the digital and analogue circuits. Noise is reduced in the system by having different voltage regulated supplies for the analogue and digital circuitry.

A TC1264 800mA +3.3V fixed output CMOS LDO **[51]** is used to regulate the +5V from the SMPS to +3.3V for the digital supply. It was selected as it provides a fixed +3.3V output and is able to provide enough current to the digital circuits. It comes in a 3-pin DDPAK.

A TC2185 +3.3V 150mA CMOS LDO [**52**] is used to regulate the output of the SMPS to +3.3V for the analogue supply. It comes in a 5-lead plastic small outline transistor package (SOT-23).

### *5.8.4 A Description of the Power Supply Circuit used in the Vibration Data Logging Device*

The power supply circuit onboard the vibration data logging board consists of a lithium cell, a LM2623 Boost converter, some post regulation LC filters, two low dropout regulators, TC1264 and TC2185, and decoupling capacitors.

The LM2623 Boost converter circuit used on the board was taken from National Semiconductors Ratio Adaptive Gated Oscillator Cookbook **[59]**. The SMPS switching frequency is set with R3, a 150kΩ resistor. A 4.7pF ceramic capacitor allows the duty cycle of the internal oscillator to be programmable and this together with R4 allows the duty cycle to dynamically compensate for changes in the input/output voltage ratio **[50]**.

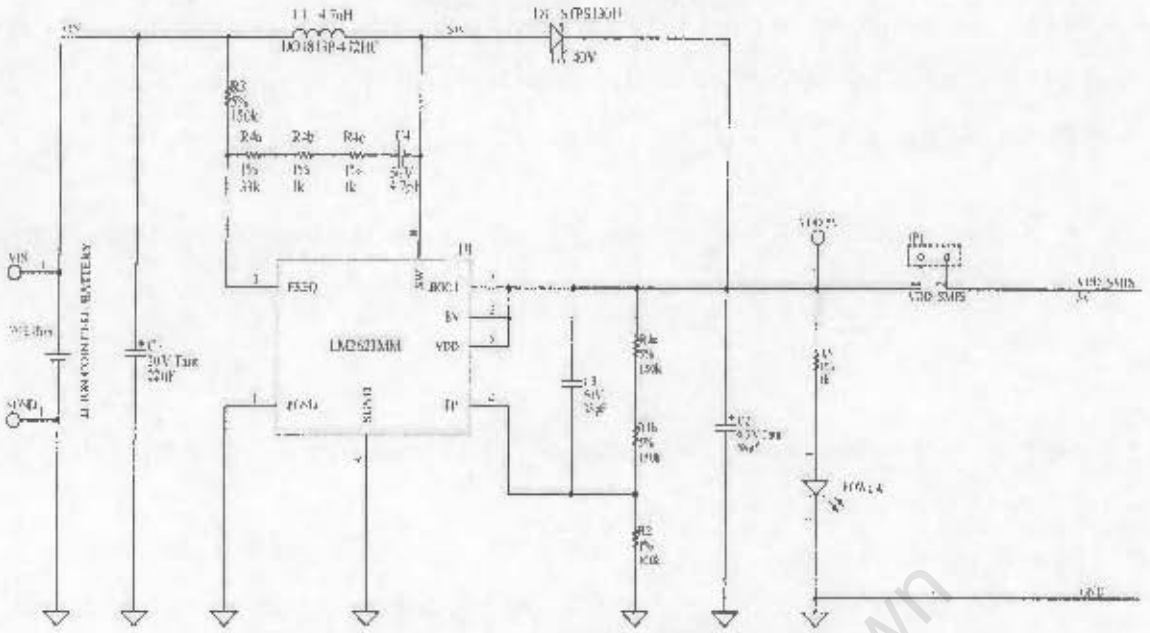


**Figure 5.17: Switch Mode Power Supply Circuit**

A LM2623 general purpose DC/DC Boost converter is used onboard the vibration data logging board to regulate and step-up the +3V from a lithium cell to +5V which then inputted to two low dropout regulators which regulate the voltage to +3.3V as this is the voltage that all the integrated circuits on the board are powered off.

The output voltage is set using a resistor divider made up of R1 and R2 which are connected between the output, the feedback pin (pin 4) and ground. R2 is selected to be a 100kΩ resistor and R1 is chosen based on Equation 5.7.

$$R_1 = R_2\left[\left(\frac{V_{out}}{1.24}\right)-1\right] \tag{5.7}$$

A300kΩ resistor was selected for R1 in order to get an output voltage of +5V.

$$V_{out} = 5V$$
$$R_2 = 100k\Omega$$

$$R_1 = R_2\left[\left(\frac{V_{out}}{1.24}\right)-1\right]$$
$$= \left(100\times10^3\right)\left[\left(\frac{5}{1.24}\right)-1\right]$$
$$= 303225.8$$
$$\approx 300k\Omega$$

A STMicroelectronics STPS130U 40V, 1A Schottky diode is used as the output diode as it can be switched at high speeds. A 4.7$\mu$H DO1813P-472HC Coilcraft inductor is used as the current storing element in the SMPS.

A 22$\mu$F tantalum decoupling capacitor is connected across the battery's inputs and a 68$\mu$F tantalum decoupling capacitor is placed across the output of the SMPS in order to reduce output ripple.

A green led is connected with a 1k$\Omega$ current limiting resistor across the output of the SMPS and is used as a power on indicator light.

The SMPS is connected via a jumper (JP1) to the low dropout regulators. This means that it can be disconnected from the system for troubleshooting or if an alternative voltage input is required to the LDOs.

Connector pins are placed on the input voltage (VIN) and ground (GND) connections. These mean that an external DC power supply unit can be used as the input voltage source instead of the lithium cell. A test pin is provided on the output of the SMPS (VDD_SMPS).

Two small 75$\Omega$ ferrite beads were inserted in the output of the SMPS and this, together with two 100$\mu$F tantalum decoupling capacitors, form low-pass filters which are used to reduce any noise on the power supply.

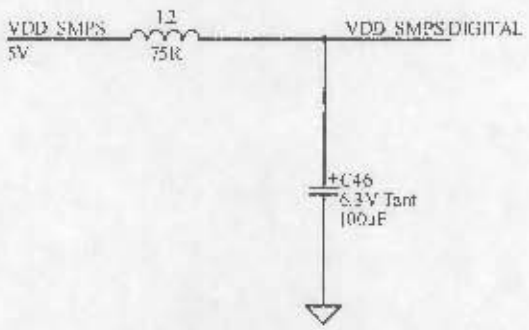



**Figure 5.18: A LC Low-pass Filter**

A small 75$\Omega$ ferrite beads is inserted in the output of the SMPS and this together, with a 100$\mu$F tantalum decoupling capacitor, form low-pass filters which are used to reduce any ripple on the power supply.

The outputs of the LC low-pass filters are then inputted into the two low dropout regulators, TC1264 and TC2185.

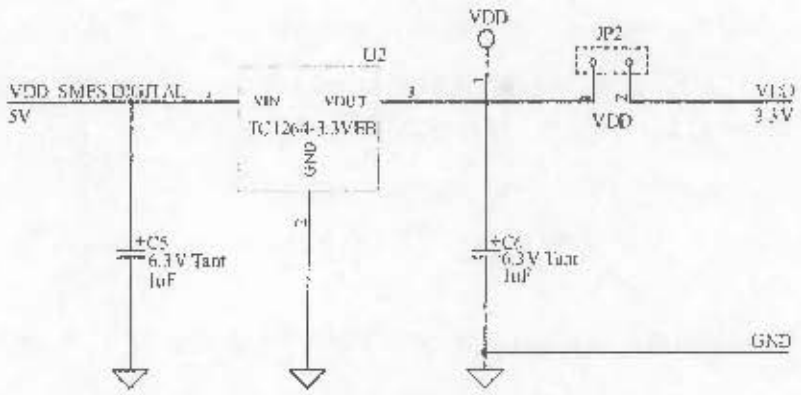



**Figure 5.19: The Digital Supply Circuit**

A TC1264 800mA +3.3V fixed output CMOS low dropout voltage regulator is used to regulate the +5V from the SMPS to +3.3V for the digital supply.

A $1\mu F$ tantalum decoupling capacitor is placed at the input to the TC1264 (pin 1) and pin 2 is connected to the digital ground plane. The back tab of the chip is also connected to the digital ground plane. A $1\mu F$ tantalum decoupling capacitor is used as the output decoupling capacitor for the TC1264. A jumper (JP2) links the output of the TC1264 to the digital power supply line which is the power supply line for all the digital circuits onboard the vibration data logging board. A test point is provided on the output of the TC1264 (VDD).

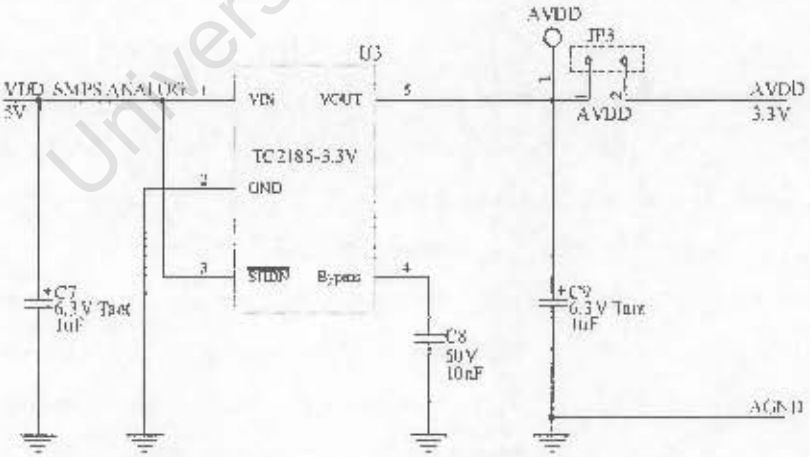



**Figure 5.20: The Analogue Supply Circuit**

A TC2185 +3.3V 150mA CMOS low dropout voltage regulator is used to regulate the output of the SMPS to +3.3V for the analogue supply.

A 1$\mu$F tantalum decoupling capacitor is placed at the input to the TC2185 (pin 1) and pin 2 is connected to the analogue ground plane. The do not shutdown pin (pin3 – $\overline{SHDN}$) is connected directly to the input to ensure normal operation and the Bypass pin (pin 4) is connected to ground using a 10nF ceramic capacitor which improves the power supply rejection ratio. A 1$\mu$F tantalum decoupling capacitor is used as the output decoupling capacitor for the TC2185. A jumper (JP3) links the output of the LDO to the analogue power supply line that supplies all the analogue circuits onboard the vibration data logging board. A test point is provided on the output of the TC2185 (AVDD).

Decoupling capacitors are supplied throughout the board between all power signals and ground planes. Small 0.1$\mu$F ceramic capacitors are used to filter out high frequency noise while larger 100$\mu$F tantalum capacitors are used to filter out lower frequencies. Decoupling capacitors are placed as close to the power pins of the devices as possible. They are then connected directly to the ground plane through a via. Every IC has at least one decoupling capacitor.



## 5.9 Printed Circuit Board Design

### *5.9.1 An Introduction to the Printed Circuit Board Design*

Electrical circuits are constructed on printed circuit boards (PCB) which are essentially layers of copper laminated on to layers of insulator. Electronic components are soldered onto the PCB and the connections between them are etched out to form traces. Layers are connected to each other through small plated holes known as vias. Originally, all components had pins which were soldered into plated holes on the PCB. Surface mounted devices are now becoming the new means of connecting components to PCBs. These devices are connected to the PCB by soldering a lead onto a pad. This means that the devices can be made very small as pin size is not a limiting factor. PCBs are graded according to the laminate material. This determines the quality of the board. PCBs can be single-sided, double-sided through holes, or multilayer boards depending on the application and complexity of the circuit.

To ensure that noise in a circuit is kept at a minimum, it is advised that ground lines are made into ground planes which mean that an entire plane of copper forms the ground connection.

Connections to the ground plane are made through vias which shortens the path to ground. The ground plane also provides some shielding against noise from external sources [60].

It is important to separate the analogue and digital ground planes and connect them to the power supply using a single point connection. This further reduces noise on the board. The analogue ground plane must not overlap the digital ground plane as the distributed capacitance will couple high speed digital noise into the analogue circuitry [60].

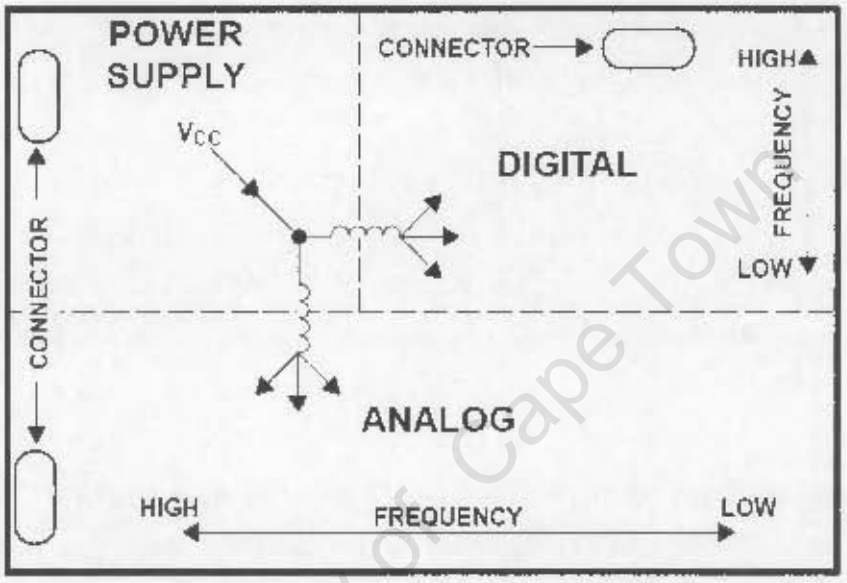



**Figure 5.21: An Example of an Ideal Layout for a Printed Circuit Board**

Separating the analogue and digital ground planes reduces noise on a PCB. These planes are connected to the power supply via a single point connection. It is recommended that components are placed on the PCB according to their speed (Taken from Mancini, R., 2002) [60].

Mancini (2002) [60] recommends that components are placed on the PCB according to their speed and an example of a well designed circuit board layout is given in Figure 5.21.



### 5.9.2 *A Description of the Printed Circuit Board used in the Vibration Data Logging Device*

The vibration data logging prototype board was designed with Altium Design Explorer ver. 7.2.85. This program was used to draw the schematics, create printed circuit board drawings and finally to produce the manufacturing Gerber files. All the printed circuit board diagrams can be found in Appendices G and H. Photographs of the boards can be found in Appendix I.

The vibration data logging printed circuit was manufactured on 150 x 120 x 3 mm double sided plated through-hole board. The PCB material was FR-4 which is flame resistant and made from glass cloth and epoxy. The mini accelerometer board was made from the same material on a 25 x 25 x 3 mm double sided plated through-hole board. The copper thickness was 70 $\mu$m. Both top and bottom sides were covered with a solder mask.

Ground planes were made on both sides of the board to provide maximum shielding. There are three separate ground planes: power, analogue and digital. The power ground plane is linked to the digital ground plane at the top edge of the board and to the analogue plane at the bottom edge of the board through single point connections.

All of the digital components and digital signal traces are placed on the digital ground plane. The dsPIC30F6014A is placed on the edge of the analogue and digital planes as the analogue plane lies under the ADC pins of the device. All the analogue components and analogue signal traces are found on the analogue ground plane. ICs are all placed on the top side of the board.

Signal traces are placed, where possible, on the top side of the board while the power supply lines are placed on the bottom of the board and linked to the components using vias. Signal traces are on average 0.4 mm wide with the higher current power supply traces being 1 mm wide.

The PCB traces for the SMPS were laid out according to the specifications found in National Semiconductors 'Layout guidelines for switching power supplies' **[61]**. This suggests that planes are used instead of traces to connect signals and that the high frequency components are placed far away from the feedback line. It is also import to make sure the traces in-between components are as short as possible.

The SMPS is linked to the LC filters and then to the LDOs using a 5 mm trace. The components used in the analogue and digital supplies are connected to the ICs using planes rather than traces which further reduces noise.

Signal lines are kept as short as possible to prevent degradation of the signal through noise. It is very important that the traces from the crystal oscillator are as short and direct as possible as acute angles might cause reflections which prevents the oscillator from starting up.

All signal trace corners are angled at 45°, as this insures that there will be no reflections of high speed signals at the corners.

Off board connectors such as the RS-232 serial communications port and the RJ-12 ICD2 in-circuit debugger connector are placed at the top end of the board for ease of use. The electret condenser microphone socket is placed on the bottom left corner of the board to separate analogue and digital components. The ADXL202E accelerometer board is connected to the main PCB using a 10 x 2 socket and can be disconnected when it is not needed.

Jumpers have been used throughout the board to separate sections of the board for trouble shooting and to isolate signals. Test pins have been provided so that an oscilloscope probe can be connected allowing all the major signals to be viewed.

# 6 Software Design: Vibration Data Logger

## 6.1 An Overview of the Software Requirements

Software onboard a data logging device is responsible for interfacing all the different hardware modules; collecting data samples; performing calculations and analysis on data; communicating between devices; and for general control of the system. This chapter provides a detailed look at the software systems implemented in the vibration data logging board. The software code for the vibration data logging board can be found in Appendix K.

The software was written in C using the MPLAB® C30 C compiler [62], which is Microchip's ANSI compliant C compiler for 16-bit devices such as the dsPIC30F family of digital signal processors.

This software was written for the integration of the all the functions used to control the modules on the vibration data logging board. The program for each module was written and tested separately. Due to the failure of the Write/Read operation of the AT454DB321C Dataflash® this integration program could not be implemented. A description of each of the tests and the results can be found in Chapter 8.

The vibration data logging board must record data from the sensors, analyse it and then send it to a personal computer for further analysis. The main software program consists of four parts. They are each discussed in more detail in the following sections (Fig. 6.1):

1. *Setup* sets up the dsPIC30F6014A digital signal controller's input/output (I/O) ports, analogue to digital controller module, universal asynchronous receiver transmitter module and serial peripheral interface module for the vibration data logging board.

2. *Record* is used to collect data samples from either the accelerometer, microphone or temperature sensor at the specified sample rate; perform the required conversions on the data and then store it in the AT45DB321C Dataflash® chip.

3. *Send* is used to transmit the data stored in the AT45DB321C Dataflash® chip via the dsPIC30F6014A's UART module to a personal computer running Microsoft® Hyper Terminal.

4. *Erase* is used to clear the entire contents of the AT45DB321C Dataflash® chip.

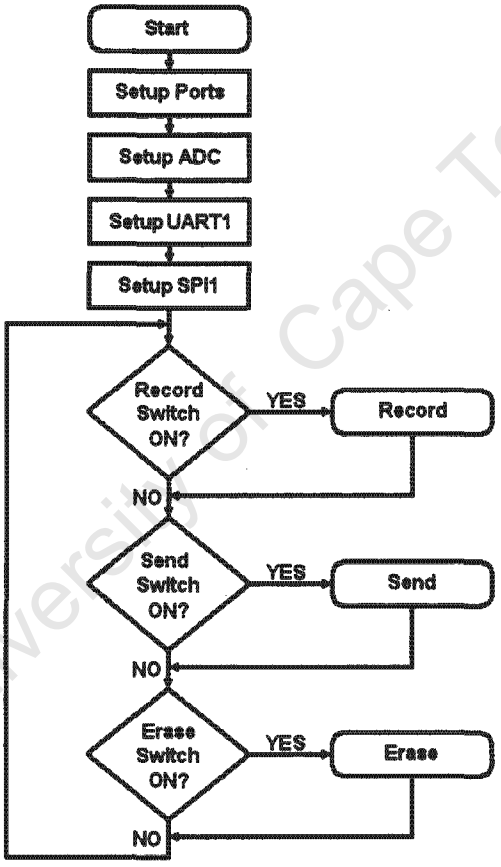



**Figure 6.1: The System Diagram of the *Main* Function**

The main software program consists of four parts: the setup functions; the record function; the send function and the erase function. The processor runs through the main program and checks to see whether any of the functions switches have been set. If so it jumps to the appropriate function.

Record, send and erase are selected by setting the appropriate dip switch. Record is set with switch 1; Send with switch 2; and Erase with switch 3. Each switch is prioritised according to function, in other words if the record switch is activated all the other switches are deactivated.

If the send switch is activated, the erase switch is deactivated but the record switch can still override the send function and so on. The record function has the highest priority, followed by the send function. The erase function has the lowest priority.

## 6.2 Include Files

The main software used to run the vibration data logging board includes five header files which are used to define common parameters used throughout the program. They are:

1. p30f6014a.h **[64]** is the standard MPLAB-C30 dsPIC30F6014A processor header which contains the definitions for all of the function registers and macros to setup the system specifications of the dsPIC30F6014A.

2. math.h contains the standard C math library definitions.

3. dsPIC30F6014APins.h defines the pin names used on the vibration data logging board.

4. AT45DB321C.h defines the opcode commands used by the AT45DB321C Dataflash® chip.

5. common.h defines all the common parameters, such as instruction clock frequency and baud rate, used in the main software program.

These header files are used to simplify the programming process and shorten troubleshooting time as constants are defined in one place and can therefore be changed easily.

## 6.3 The Setup Functions

The setup functions are used to initialise the dsPIC30F6014A's peripheral modules which are used on the vibration data logging prototype board. Each of the peripheral modules has 16-bit function registers which are used to configure and control the operation of the module. The dsPIC30F6014A digital signal controller's input/output (I/O) ports, analogue to digital

controller module (ADC), universal asynchronous receiver transmitter module (UART) and serial peripheral interface module (SPI) are initialised with four setup functions. These are discussed in more detail in the following sections.

### *6.3.1 Setup Ports Function – InitPorts*

The input/output (I/O) pins onboard microcontrollers are used to monitor input signals and control external devices connected to them. Most of the dsPIC30F6014A's I/O pins are shared with onboard peripheral modules. The I/O functionality is generally disabled when the peripheral module is initialised. The I/O ports are controlled by three registers **[63]**. These are:

1. *TRISx* (Data Direction Register) is used to initialise the port pins as inputs or outputs by writing a 1 (input) or a 0 (output) to the appropriate bit.

2. *PORTx* (I/O Port Register) is used to set and access data on the I/O port pins. A read of the port register gives the values on the I/O lines and a write to the port register writes the value to the port latches.

3. *LATx* (I/O Latch Register) is used to set and access the values of the port latches. A read of the latch register gives the value of the port latches, while a write to the latch register writes the value to the port latches.

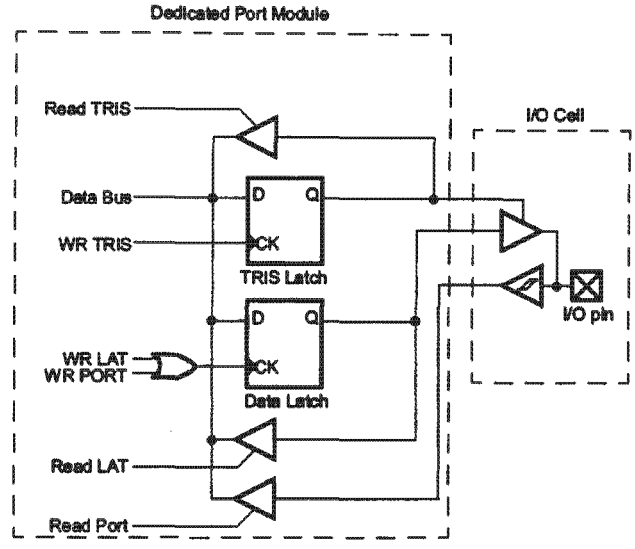



**Figure 6.2: Structure Block Diagram of a General Input/Output Port**

*TRISx* controls the data direction of a general I/O pin. *PORTx* and *LATx* are used to set and access data on the I/O pins and the data latches respectively. A read of the port register gives the values on the I/O lines and a write to the port register writes the value to the port latches. A read of the latch register gives the value of the port latches, while a write to the latch register writes the value to the port latches (Adapted from Microchip Technology Incorporated Reference Manual, 2003) **[63]**.

A function called *InitPorts* was created and used to set up the directions and values of all the ports on board the dsPIC30F6014A. The names, port numbers and initial value of the general input/output pins used onboard the vibration data logging board are given in Table 6.1.

**Table 6.1: Name, Direction and Initial Value of the General I/O Ports**

The direction and initial value of general input/output ports used onboard the vibration data logging board are set up with a function called *InitPorts.c*. This table gives the names of the signals and of the ports to which they are connected. The initial values are set using the PORTx registers in the dsPIC30F6014A.

| INPUTS | | | OUTPUTS | | |
|---|---|---|---|---|---|
| SIGNAL NAME | PORT NAME | INITIAL VALUE | SIGNAL NAME | PORT NAME | INITIAL VALUE |
| *SEND* | RD1 | 0 | *LED1* | RC1 | 0 |
| *RECORD* | RD2 | 0 | *LED2* | RC2 | 0 |
| *ERASE* | RD3 | 0 | *LED3* | RC3 | 0 |
| | | | *LED4* | RC4 | 0 |
| *READY/* $\overline{BUSY}$ | RD9 | 0 | | | |
| | | | $\overline{CS}$ | RD8 | 1 |
| | | | $\overline{RESET}$ | RD10 | 1 |
| | | | $\overline{WP}$ | RD11 | 1 |
| | | | *RECORD_LED* | RG6 | 0 |
| | | | *SEND_LED* | RG7 | 0 |
| | | | *ERASE_LED* | RG8 | 0 |

Port directions were set using the appropriate *TRISx* data direction register. The values of the ports were initialised using *PORTx* registers. All unused pins were made outputs and their value was set to zero.

**Table 6.2: Function Registers for the General I/O Ports**

The values of the function registers used to configure the general input/output ports in the dsPIC30F6014A digital signal processor are given in hexadecimal and binary.

| REGISTER NAME | HEXADECIMAL VALUE | BINARY VALUE |
|---|---|---|
| *TRISA* | 0x0000 | 0000 0000 0000 0000 |
| *TRISB* | 0x0F03 | 0000 1111 0000 0011 |
| *TRISC* | 0x0000 | 0000 0000 0000 0000 |
| *TRISD* | 0x020E | 0000 0010 0000 1110 |
| *TRISF* | 0x0104 | 0000 0001 0000 0100 |
| *TRISG* | 0x0000 | 0000 0000 0000 0000 |
| *PORTA* | 0x0000 | 0000 0000 0000 0000 |
| *PORTB* | 0x0000 | 0000 0000 0000 0000 |
| *PORTC* | 0x0000 | 0000 0000 0000 0000 |
| *PORTD* | 0x0F00 | 0000 1111 0000 0000 |
| *PORTF* | 0x0000 | 0000 0000 0000 0000 |
| *PORTG* | 0x0000 | 0000 0000 0000 0000 |

### 6.3.2 *Setup Analogue to Digital Converter Module – InitADC12.c*

The dsPIC30F6014A contains a sixteen input 12-bit analogue to digital converter module which shares its input pins with PORT B.

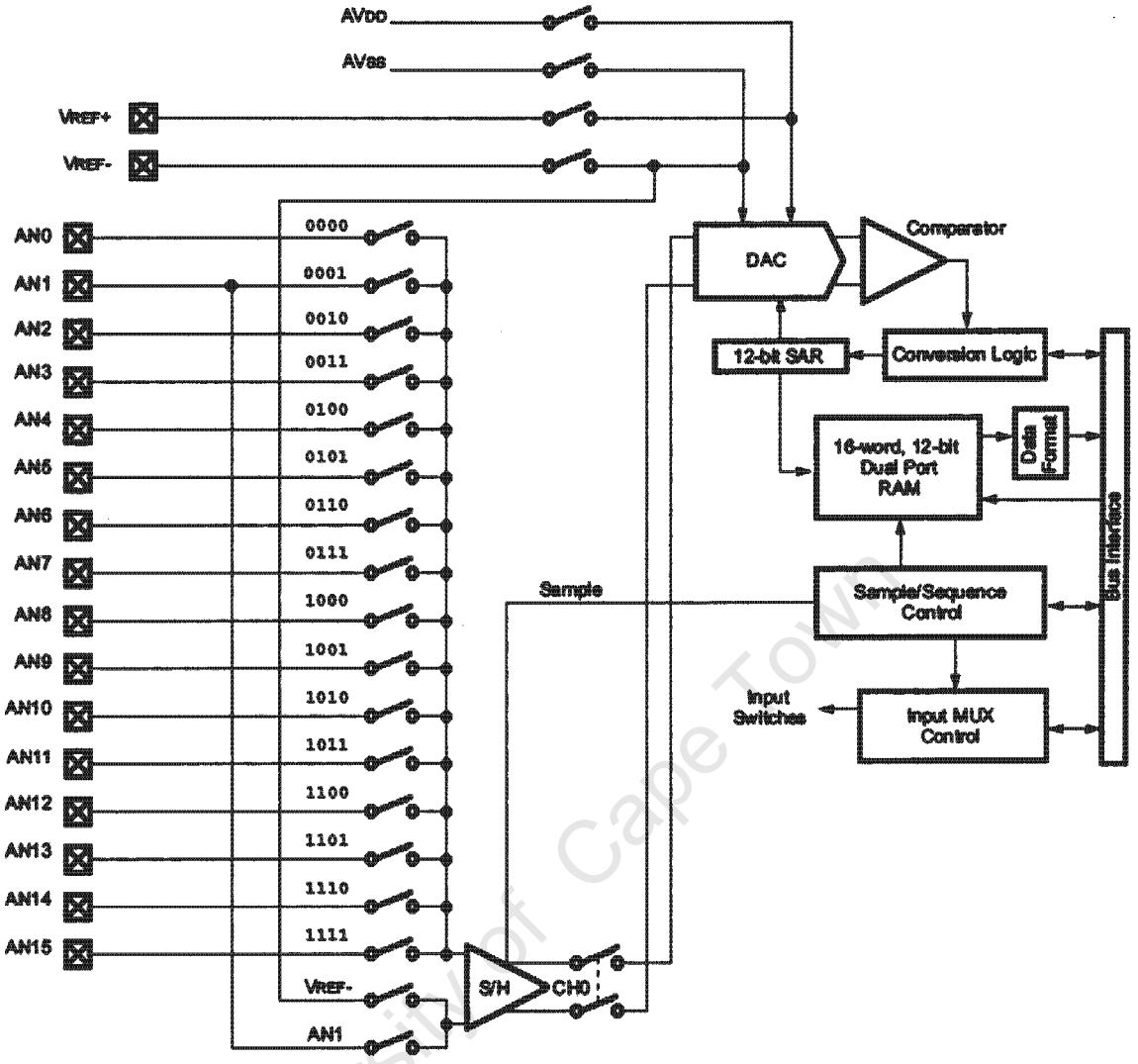



**Figure 6.3: Structure Block Diagram of the 12-bit ADC Module**

The dsPIC30F6014A contains a 12-bit analogue to digital converter module which uses a successive approximation register (SAR) to convert signals. All sixteen ADC inputs are multiplexed into a sample and hold register that is input to the converter which generates the result. The analogue supply lines are used as the voltage references in conversions. The outputs from the sensors are connected to the 12-bit ADC at the following locations: the temperature sensor is connected to pin 27 (AN8), the outputs from the accelerometer circuit are connected to pin 28 (AN9) and pin 29 (AN10), and the output from the microphone circuit is connected to pin 30 (AN11) (Adapted from Microchip Technology Incorporated Data Sheet, 2005) [47].



This module has six 16-bit function registers which are used to control the functioning of the ADC module. They are:

1. *ADCON1* (ADC Control Register 1) is used to enable the ADC Module; set the data output format; set the conversion trigger source; and enable auto or manual sampling.

2. *ADCON2* (ADC Control Register 2) is used to select the voltage reference source; set the number of samples per interrupt; set the buffer configuration; and set whether to use multiplexer A (MUXA) settings or to alternate between MUXA and multiplexer B (MUXB) settings.

3. *ADCON3* (ADC Control Register 3) is used to select ADC conversion clock source; and the acquisition and conversion times. These are used to set the automatic sampling rate of the ADC module.

4. *ADCHS* (ADC Input Select Register) is used to select the input pins to the sample and hold amplifiers.

5. *ADPCFG* (ADC Port Configuration Register) is used to configure the input pins as either analogue or digital. A digital pin acts as a general input/output pin of PORT B.

6. *ADCSSL* (ADC Input Scan Selection Register) is used to set the sequence of the scanned inputs.

Sampling can be manual or automatic. Manual sampling is achieved by clearing the *SAMP* bit in the *ADCON1* register. This starts data conversion. Sampling continues once the *SAMP* is set again. Auto sampling is enabled by setting the *ASAM* bit in the *ADCON1* register. Sampling begins immediately after conversion. The auto sampling rate is set using the *SAMC*<4:0> (Auto Sample Time Bits) and the *ADCS*<5:0> (ADC Conversion Clock Select Bits). An analogue clock ($T_{AD}$) must have a minimum period of 668ns as defined in the Microchip Technology Incorporated Data Sheet (2005) [**47**]. This ensures that the sampling capacitors within the ADC do not lose charge. $T_{AD}$ controls the conversion timing which must be at least fourteen clock periods long. Equations 6.1 and 6.2 are used to calculate the conversion clock period [**63**].

$$T_{AD} = \frac{T_{CY}\left(ADCS<5:0>+1\right)}{2} \tag{6.1}$$

$$T_{SAMP} = T_{AD}\left(14+SAMC<4:0>\right) \tag{6.2}$$

Where $T_{AD}$ is the period of the analogue to digital converter clock in seconds; $T_{cy}$ is the instruction clock period in seconds and $T_{SAMP}$ is the total sampling period in seconds. *ADCS*<5:0> and *SAMC*<4:0> are the decimal values of these bits.

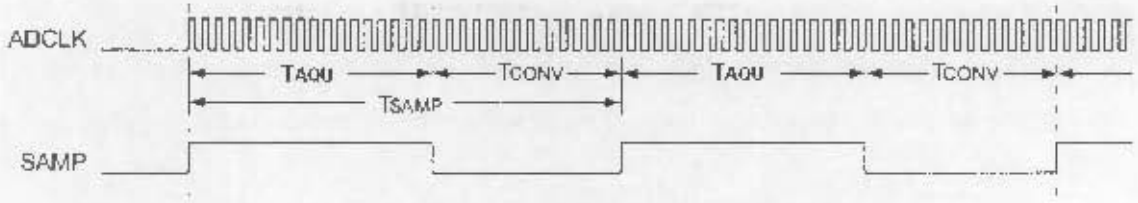



**Figure 6.4: Timing Diagram of the Auto Sampling Mode of the ADC Module**

Auto sampling is enabled by setting the *ASAM* bit in the *ADCON1* register. Sampling begins immediately after conversion. The auto sampling rate is set using the *SAMC* (Auto Sample Time Bits) and the *ADCS* (ADC Conversion Clock Select Bits). An analogue clock ($T_{AD}$) controls the conversion timing ($\geq 14T_{AD}$) and the acquisition time ($T_{AQU}$). The *SAMP* bit is cleared automatically once conversion begins (Adapted from Microchip Technology Incorporated Reference Manual, 2003) **[63]**.

The results of an ADC conversion are stored in a read-only 16-word buffer called ADCBUF0 – ADCBUF15. The buffer can be split into two 8-word buffers. An interrupt will occur once first 8-words in the buffer are full; data are then loaded into the second 8-word buffer. The user can access data from the first buffer while the second buffer is filled, providing more processing time between interrupts.

The buffer is 12-bits wide and data are represented in four different 16-bit data formats. These are: Integer; Signed Integer; Fractional; and Signed Fractional (Q1.15).

*InitADC12* was used to configure the 12-bit analogue to digital converter module for use on the vibration data logging board; setup the analogue to digital converter interrupts and to start the ADC module. An *InitADC12.c* function was written for the different analogue sensors as each requires different configuration settings. A general description of initialising functions for each of the analogue sensors is given below:

1. *Accelerometer* – The ADXL202E has two outputs ACCX (AN9) and ACCY (AN10), which must each be sampled at 16 kHz. All other ADC pins are set as digital outputs. The *ALTS* bit in *ADCON2* register selects the alternate input sample mode. This allows the ADC to alternate inputs to the sample-hold amplifier after every sample-hold sequence. The *ALTS* bit is set in this case and the ADC alternates between the ACCX and ACCY

inputs. The incoming signals must be sampled at twice the required rate to ensure a sampling rate of 16 kHz. Only half the samples from a particular signal will be available when both signals are collectively sampled at 16 kHz. The sample rate is therefore set to 32kHz.

Automatic sampling is enabled and the sample rate is set by the *SAMC<4:0>* (Auto Sample Time Bits) and *ADSC<5:0>* (A/C Conversion Clock Select Bits). These values are set to

$$ADCS<5:0>=57=111001_b$$
$$SAMC<4:0>=8=01000_b$$

The auto sampling calculations can be found in Appendix J.

The output data format from the ADC is signed fractional. This produces numbers in the range 0.9995 to -1 which ensures that no overflow can occur in multiplication operations such as in RMS calculations.

| S | d10 | d9 | d8 | d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**Figure 6.5: Signed Fractional (Q1.15) Data Format of the 12-bit ADC Module**

The 12-bit ADC result is represented in one of four different 16-bit number formats. The Q1.15 data format has one sign bit and fifteen data bits, but in this case only 11-bits are used for data.

Interrupts occur after the sixteenth sample-hold sequence. This allows eight samples to be collected from ACCX and then from ACCY. The result buffer is configured as one 16-word buffer.

**Table 6.3: Function Registers for the ADC with Accelerometer Inputs**

The values of the function registers used to configure the ADC module which is connected to the accelerometer. They are given in hexadecimal and binary.

| REGISTER NAME | HEXADECIMAL VALUE | BINARY VALUE |
|---|---|---|
| *ADCON1* | 0x03E0 | 0000 0011 1110 0000 |
| *ADCON2* | 0x003D | 0000 0000 0011 1101 |
| *ADCON3* | 0x0A39 | 0000 1010 0011 1001 |
| *ADCHS* | 0x0A09 | 0000 1001 0000 1000 |
| *ADPCFG* | 0xF9FF | 1111 1001 1111 1111 |
| *ADCSSL* | 0x0000 | 0000 0000 0000 0000 |

2. *Electret Condenser Microphone* – The output of the Electret Condenser Microphone is connected to AN11. All other ADC pins are set as digital outputs. Automatic sampling is enabled and set to 20 kHz using *SAMC<4:0>* and *ADSC<5:0>* bits. They are:

$$ADCS<5:0> = 57 = 11100\mathrm{I}_b$$
$$SAMC<4:0> = 21 \quad 10101_b$$

Data are output as a signed fractional and the ADC result buffer *ADCBUF* is split into two 8-word buffers. Interrupts occur after the eighth sample hold sequence.

**Table 6.4: Function Registers for the ADC with a Microphone Input**

The values of the function registers used to configure the ADC module which is connected to the microphone. They are given in hexadecimal and binary.

| REGISTER NAME | HEXADECIMAL VALUE | BINARY VALUE |
|---|---|---|
| *ADCON1* | 0x03E0 | 0000 0011 1110 0000 |
| *ADCON2* | 0x0022 | 0000 0000 0010 0010 |
| *ADCON3* | 0x1539 | 0001 0101 0011 1001 |
| *ADCHS* | 0x000B | 0000 0000 0000 1011 |
| *ADPCFG* | 0xF7FF | 1111 0111 1111 1111 |
| *ADCSSL* | 0x0000 | 0000 0000 0000 0000 |

3. *Analogue Temperature Sensor* – The output of the MAX6608 analogue temperature sensor chip is connected to analogue to digital converter via pin AN8. All of the other ADC pins are set as digital outputs.

Data from the temperature sensor must be updated every ten second. The ADC module's sample/conversion sequence is manually set in this case as the sampling period is relatively high. This is achieved by placing a suitable delay between the setting and clearing of the *SAMP* bit in ADCON1.

Data are outputted as integers as there are no large multiplications in the temperature analysis. Interrupts are not used in this case.

| 0 | 0 | 0 | 0 | d11 | d10 | d9 | d8 | d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**Figure 6.6: Unsigned Integer Data Format of the 12-bit ADC Module**

The 12-bit ADC result is represented in one of four different 16-bit number formats. The unsigned integer data format has no sign bit and 12-bits of data.

**Table 6.5: Function Registers for the ADC with a Temperature Sensor Input**

The values of the function registers used to configure the ADC module which is connected to the temperature sensor. They are given in hexadecimal and binary.

| REGISTER NAME | HEXADECIMAL VALUE | BINARY VALUE |
|---|---|---|
| *ADCON1* | 0x0000 | 0000 0000 0000 0000 |
| *ADCON2* | 0x0000 | 0000 0000 0000 0000 |
| *ADCON3* | 0x0002 | 0000 0000 0000 0010 |
| *ADCHS* | 0x0008 | 0000 0000 0000 1000 |
| *ADPCFG* | 0xFFEF | 1111 1111 1110 1111 |
| *ADCSSL* | 0x0000 | 0000 0000 0000 0000 |

### 6.3.3 *Setup Universal Asynchronous Receiver Transmitter Module – InitUART1*

The dsPIC30F6014A contains two universal asynchronous receiver transmitter modules. UART1 was used in this study to transmit the collected data from the vibration data logging board to a personal computer. The module is made up of an asynchronous transmitter, and asynchronous receiver and a baud rate generator.

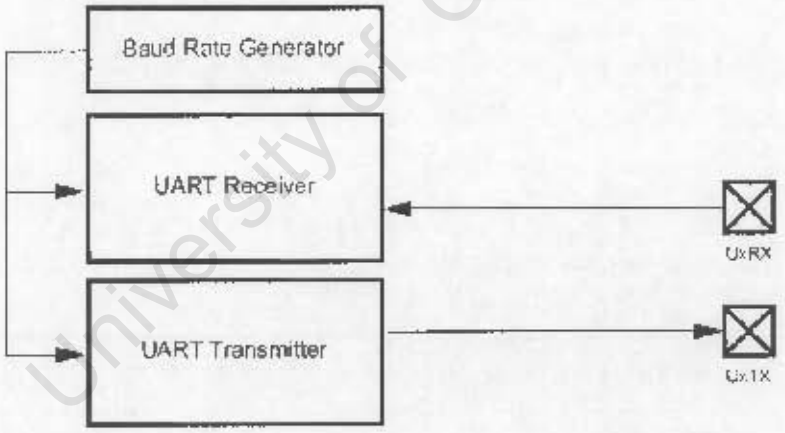



**Figure 6.7: Structure Block Diagram of the UART1 Module**

The module is made up of an asynchronous transmitter, and asynchronous receiver and a baud rate generator. Data are shifted in and out of the UART module at a baud rate set by the baud rate generator (Adapted from Microchip Technology Incorporated Reference Manual, 2003) **[63]**.

The UART modules are controlled with the following registers:

1. *UxMODE* (UARTx Mode Register) is used to enable the module; set the size, parity data and the number of stop bits of the data.

2. *UxSTA* (UARTx Status and Control Register) is used to enable interrupts and monitor the condition of the UART module.

3. *UxBRG* (UARTx Baud Rate Generator Control Register) is used to control the period of a free running 16-bit timer. The baud rate can be calculated with Equation 6.3 **[63]**:

$$Baud\ Rate = \frac{F_{cy}}{16 \times (UxBRG < 15:0 > +1)} \tag{6.3}$$

Where $F_{cy}$ is the instruction cycle clock frequency and *UxBRG<15:0>* is the decimal value of the UARTx Baud Rate Generator Control Register.

Each UART module contains a 4-word deep First-In-First-Out (FIFO) transmit buffer (*UxTXREG*) and a 4-word deep FIFO receive buffer (*UxRXREG*). Data are loaded in the transmit buffer by the user. This is then shifted into a transmit shift register (*UxTSR*) where it will be transmitted at the baud rate set by the baud rate generator. The UART receiver module is not used in this project.

*InitUART1* is used to initialise the UART1 module onboard the dsPIC30F6014A. The values of the function registers are given in Table 6.6.

**Table 6.6: Function Registers for the UART1 Module**

The values of the function registers used to configure the UART1 module in the dsPIC30F6014A digital signal processor are given in hexadecimal and binary.

| REGISTER NAME | HEXADECIMAL VALUE | BINARY VALUE |
|---|---|---|
| *U1MODE* | 0x8000 | 1000 0000 0000 0000 |
| *U1STAT* | 0x0000 | 0000 0000 0000 0000 |
| *U1BRG* | 0x0015 | 0000 0000 0001 0101 |

The data size is set to 8-bits with no parity and no stop bit. The module can continue operation in idle mode and no interrupts are enabled. The baud rate is set to 57600 bits/second and the value of the *UxBRG* register is calculated to be:

$$UxBRG<15:0>=\frac{F_{cy}}{16\cdot Baud\ Rate}-1$$
$$=\frac{20\times10^{6}}{16\times57600}-1$$
$$=20.7$$
$$\approx 21$$
$$=0x0015_{h}$$

This gives a baud rate of 56818.18 bits/second. The error between the desired baud rate and the calculated board rate is determined with Equation 6.4:

$$Error(\%)=\frac{(Calculated\ Baud\ Rate-Desired\ Baud\ Rate)}{Desired\ Baud\ Rate}\times100 \qquad (6.4)$$

$$=\frac{(56818.18-57600)}{57600}\times100$$
$$=-1.36\%$$

### 6.3.4 *Setup Serial Peripheral Interface – InitSPI1*

The dsPIC30F6014A contains two serial peripheral interface modules which can be used to connect to other devices. SPI1 is used onboard the vibration data logging board to communicate with the AT45DB321C Dataflash® Chip. The SPI modules are setup using two function registers:

1. *SPIxCON* (SPI Control Register) is used to set up the functioning of the SPI module. It can enable framed SPI; disable the SDO pin; select whether the word length is 8 or 16-bits long; select whether the dsPIC30F6014A is a master or a slave; set the SPI mode and set the serial clock speed.

2. *SPIxSTAT* (SPI Status Register) is used to give the status of the SPI device.

The SPI module contains a buffer called *SPIxBUF* which is internally comprised of two separate unidirectional registers *SPIxTXB* (SPI Transmit Buffer) and *SPIxRXB* (SPI Receive Buffer). These two registers are connected to the *SPIxSR* (SPI Shift Register) which shifts data into and out of the device at the serial clock speed. When data are written to the

*SPIxBUF* it is written internally to the *SPIxTXB* register and when data are read from *SPIxBUF* it is read from the *SPIxRXB* register. For every byte shifted out of a serial peripheral interface, one byte is shifted in.

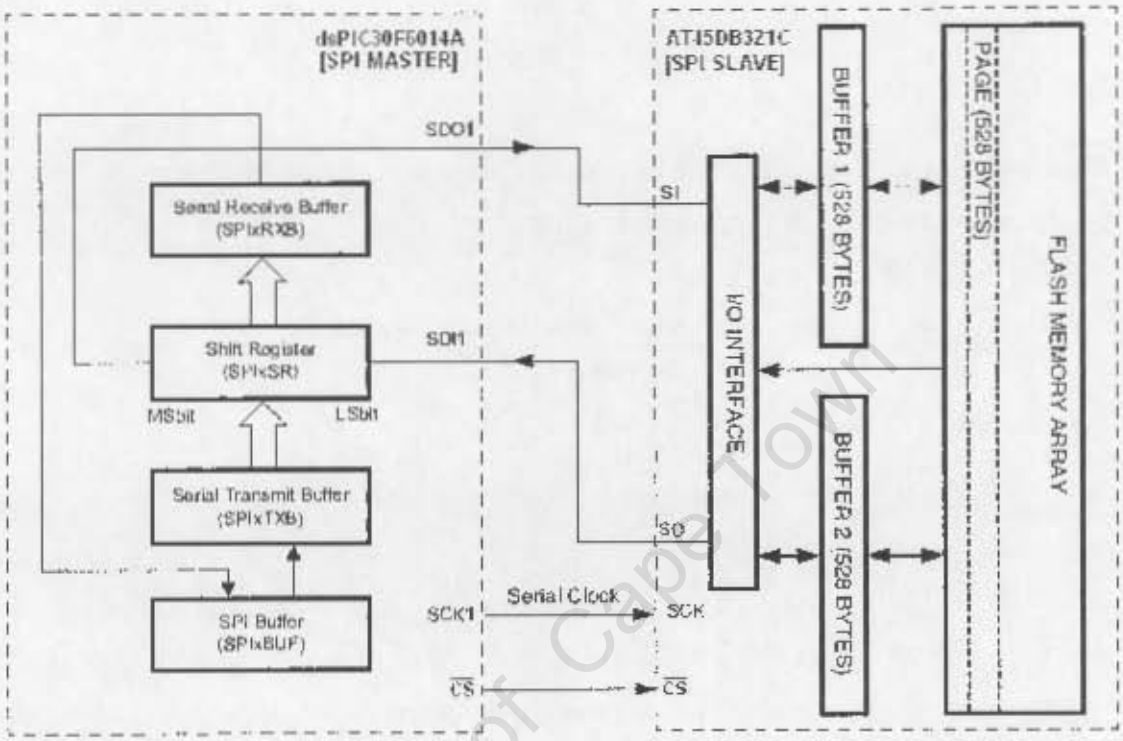



**Figure 6.8: Structure Block Diagram of the SPI Connections between the dsPIC30F6014A and the AT45DB321C Dataflash® Chip**

The SPI module contains a buffer called *SPIxBUF* which is internally comprised of two separate unidirectional registers *SPIxTXB* (SPI Transmit Buffer) and *SPIxRXB* (SPI Receive Buffer). These two registers are connected to the *SPIxSR* (SPI Shift Register) which shifts data into and out of the device at the serial clock speed. The AT45DB321C 32-Mbit Dataflash® chip is a serial flash chip which can be interfaced to a processing device using a 3-wire SPI. The AT45DB321C's memory is organised into 8192 pages of 528 bytes each. It also contains two 528 byte SRAM buffers which can be used to store data while the main memory is programmed (Adapted from Microchip Technology Incorporated Reference Manual, 2003) [63] and (Atmel Corporation Data Sheet, 2003) [48].



The SPI module supports four SPI clock modes which are defined as SPI Mode 0; SPI Mode 1; SPI Mode 2 and SPI Mode3. The timing diagrams for each of these modes are given in Figure 6.9.

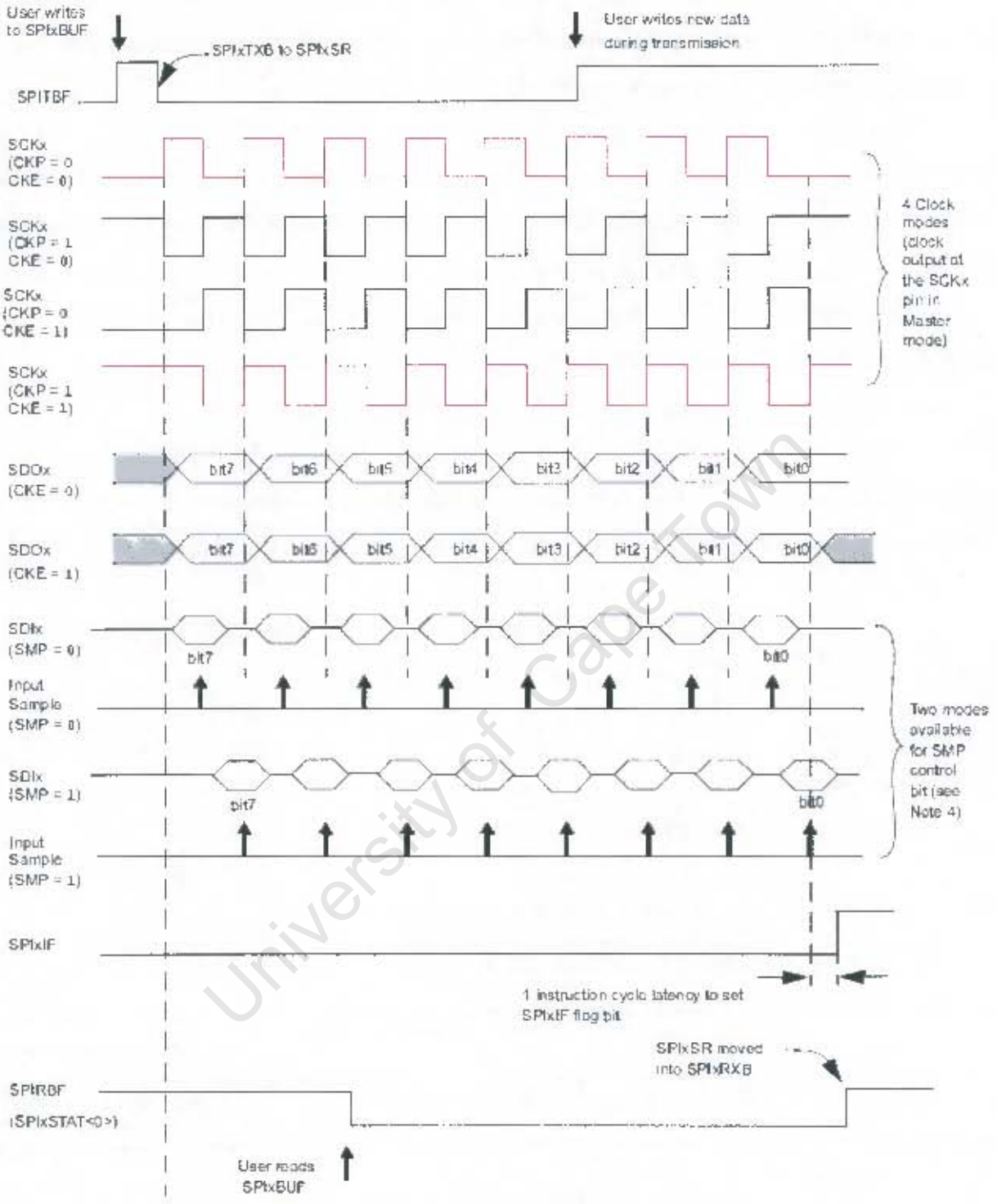



**Figure 6.9: Timing Diagrams for the Four SPI Modes**

The SPI Mode depends on the clock polarity (CKP), the edge of the clock signal (CKE) on which data are outputted and the data input sample phase. The AT45DB321C only supports SPI modes 0 and 3 which are indicated in red in the above diagram (Taken from Microchip Technology Incorporated Reference Manual, 2003) [63]:

The SPI Mode depends on the polarity of the clock (CKP) in an idle state; the edge of the clock signal (CKE) on which data are outputted and the whether the input data are sampled at the middle or end of the data output time.

*InitSPI1* is used to initialise the SPI1 module onboard the dsPIC30F6014A for communication with an AT45DB321C Dataflash® chip. The dsPIC30F6014A is selected as the master device with framed SPI support disabled as another general I/O pin RD8 ($\overline{CS}$) is used as the chip select pin. Communication is 8-bits wide and SPI Mode 0 is used. This implies that the polarity of the idle clock state is low; data are shifted in on the rising edge of the clock and that data are sampled at in the middle of the data output time.

The value of the serial clock is generated by scaling the instruction clock frequency with a primary and secondary prescaler. The clock, in this case, is set to 1.67 MHz.

$$SCLK = \frac{F_{cy}}{(Primary\ Prescaler \times Secondary\ Prescaler)} \tag{6.5}$$

$$= \frac{20 \times 10^6}{(3 \times 4)}$$

$$= 1.67\ MHz$$

**Table 6.7: Function Registers for the SPI1 Module**

The values of the function registers used to configure the SPI1 module in the dsPIC30F6014A digital signal processor are given in hexadecimal and binary.

| REGISTER NAME | HEXADECIMAL VALUE | BINARY VALUE |
|---|---|---|
| *SPI1CON* | 0x0036 | 0000 0000 0011 0110 |
| *SPI1STAT* | 0x8000 | 1000 0000 0000 0000 |

# 6.4 General Functions

## 6.4.1 *The Send and Receive Function for the Serial Peripheral Interface Module – SendReceiveByte*

The *SendReceiveByte* function is used to send an 8-bit data byte to the AT45DB321C Dataflash® chip via serial peripheral interface 1 on the dsPIC30F6014A and it returns the received byte in *SPI1BUF*. This function utilises the SPI property that for every byte shifted out of a master device via a serial peripheral interface, one byte is shifted in from the slave device.

The input data byte is written to *SPI1BUF* and the SPI1 module is enabled. This will transmit the data via the SPI output data line to the slave device. The function must then wait until the transfer is complete and a byte has been received in *SPI1BUF* via the SPI input data line. The received byte is then returned by the function. This program was adapted from code by Stowe (2006) **[65]**.

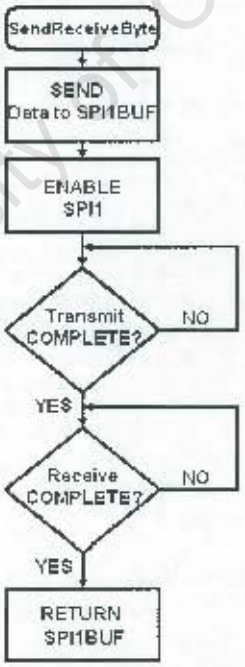



**Figure 6.10: The System Diagram of the *SendReceiveByte* Function**

The *SendReceiveByte* function is used to send an 8-bit data byte to the AT45DB321C Dataflash® chip via serial peripheral interface 1 on the dsPIC30F6014A and it returns the received byte found in SPI1BUF.

### 6.4.2 *RMS Calculation Function – RMSCalculation*

The *RMSCalculation* function is used to calculate the RMS value of an array of numbers, N values long. Its input arguments are a pointer to the beginning of an array of integers and the length of the array N. The RMS value is calculated using Equation 3.2. The RMS value is then converted from a Q1.15 signed fractional number to an unsigned integer using the Q15toInteger function described in Section 6.4.3. This value is then returned by the function.

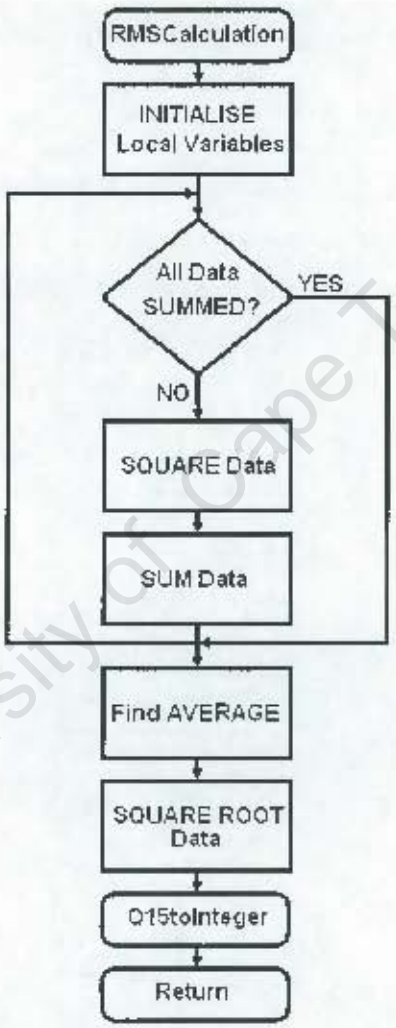



**Figure 6.11: The System Diagram of the *RMSCalculation* Function**

The *RMSCalculation* Function is used to calculate the RMS value for an array of length N. All data points are squared and summed together. The average of the sum is then calculated. The square root of the average is converted to an integer and returned by the function.

### 6.4.3 *Q1.15 Signed Integer to Unsigned Integer Conversion Function – Q15toInteger*

This function converts the 12-bit result in Q1.15 signed fractional format from the ADC module to an unsigned integer. To convert a Q1.15 number to and unsigned integer the following steps were taken as illustrated in Figure 6.12. The converted value is then returned by the function.

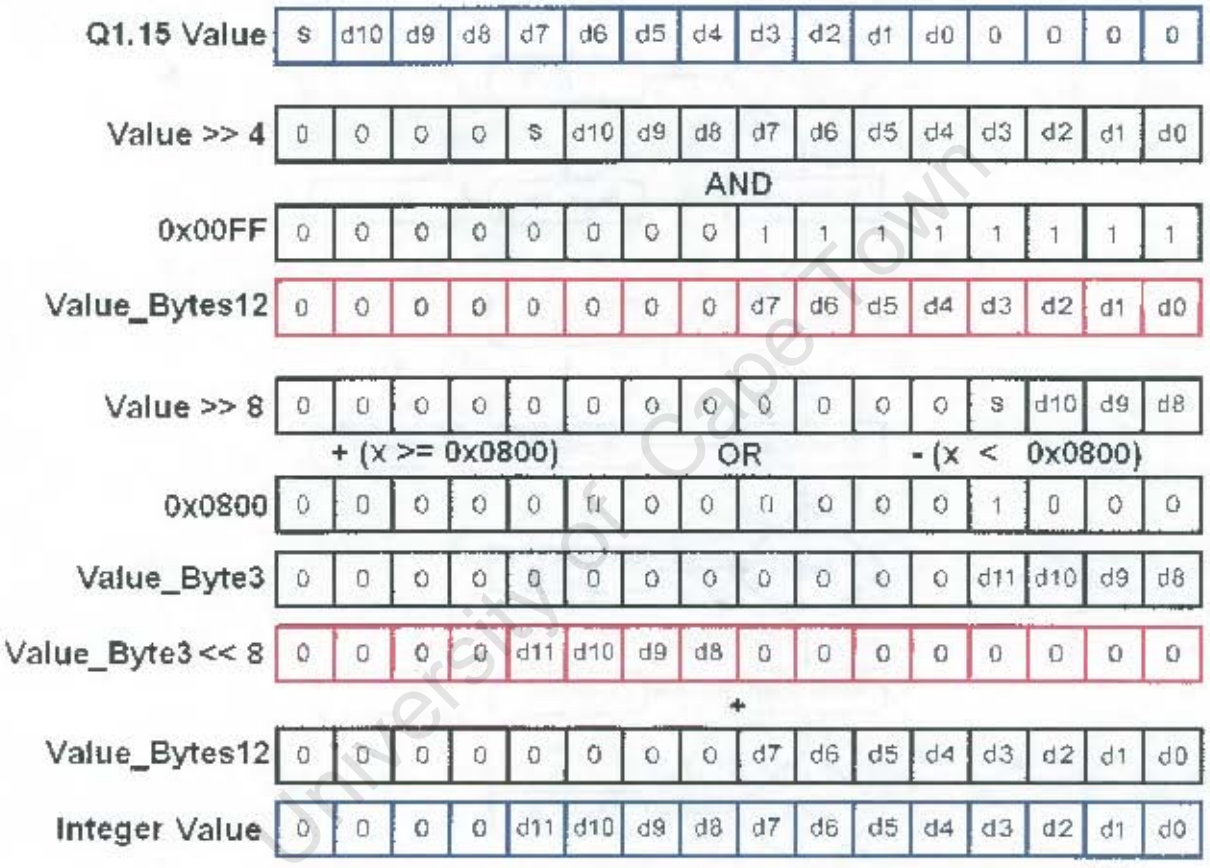



**Figure 6.12: Q1.15 Fractional Format to an Unsigned Integer Conversion Function**

A Q1.15 fractional value can be converted to an unsigned integer using the above procedure.

### 6.4.4 *Output from the ADC Module to Volts in ASCII Format Function – asciiVoltsConv*

The output from the ADC is a 12-bit value which ranges in value from 0 – 4096 ($2^{12}$). It is therefore necessary to convert this value into a voltage value which can be analysed more easily. The *asciiVoltsConv* function converts an unsigned integer into four ASCII characters which can be easily sent to a PC via the UART1 module.

If the 12-bit value is greater than 1V, it is divided by 0x04D9 (1V → output from the ADC is $1241_d$ = $0x04D9_h$). The quotient is equal to the number of voltages. This value is then converted to an ASCII character by the *asciiConv* function described in Section 6.4.5. The remainder is passed on to the next step. This is repeated for 100mVs, 10mVs and 1mVs.

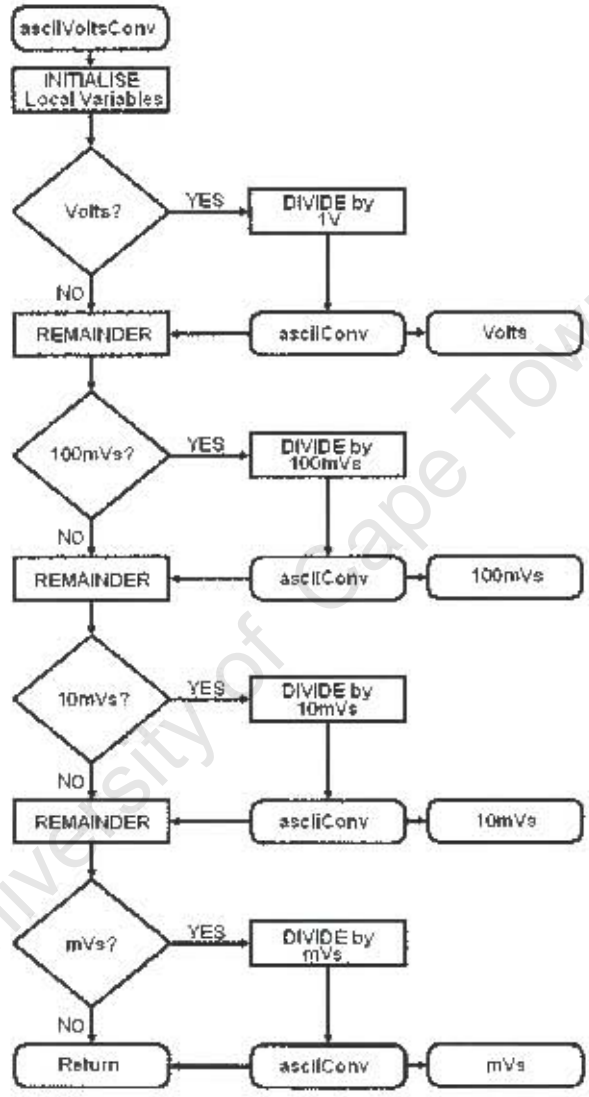



**Figure 6.13: ADC Output to a ASCII Voltage Value Conversion Function**

The output from the ADC can be converted to the equivalent voltage value using the algorithm illustrated above. The original value is separated into four ASCII voltage values which can be displayed easily on a PC.

### *6.4.5 Integer to ASCII Conversion Function – asciiConv*

This function converts a 16-bit value to an 8-bit ASCII character which can be viewed easily on a PC. This is achieved by selecting the first four bits of the value and adding them to a suitable offset of either 0x37 for numbers greater than 9 or 0x30 for numbers less than 9.

## 6.5 Erase

The erase function is activated by enabling switch 3 on the vibration data logging board. It is used to erase the entire contents of the AT45DB321C Dataflash® chip. It does this by erasing blocks of eight pages at a time. The default reset state of all the values is 0xFF. This code was adapted from code written for the AVR335 Digital Sound Recorder [66].

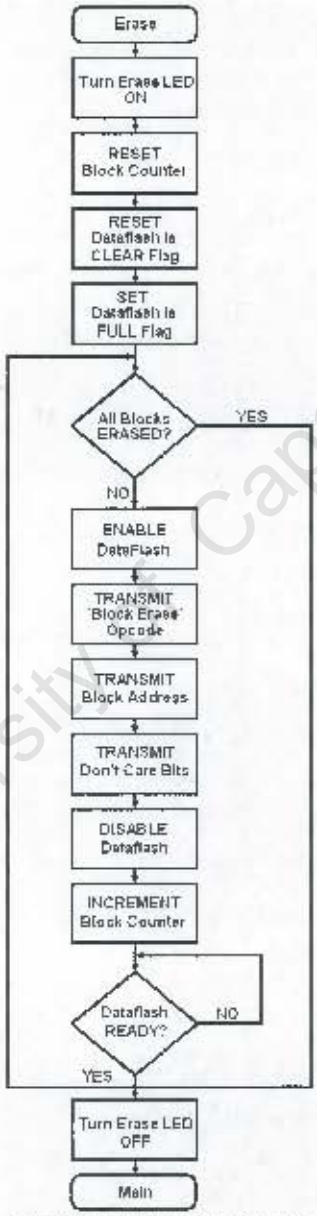



**Figure 6.14: The System Diagram of the *Erase* Function**

The *Erase* function is used to erase the entire contents of the AT45DB321C Dataflash® chip by erasing blocks of eight pages at a time. The default reset state of all the values is 0xFF.

The erase function does not have any input parameters and it does not return a value. The Erase LED is used to indicate when the erase function is in operation. The Erase function should be activated before any data logging takes place as it set all the bits in the Dataflash® chip to a known state.

A block erase of the AT45DB321C Dataflash® is achieved by enabling the Dataflash® chip ($\overline{CS} = 0$), transmitting the block erase opcode command $0x50_h$ followed by two reserved bits, nine address bits, and thirteen don't care bits via the serial peripheral interface to the Dataflash® chip, and then disabling the chip ($\overline{CS} = 1$).


**Figure 6.15: Block Erase Instruction Format**

A block erase instruction for the AT45DB321C Dataflash® consists of a four bit opcode (C), followed by two reserved bits (R), nine address bits (A) and thirteen don't care bits (X).

This process must be repeated 1024 times in order for the entire Dataflash® chip to be erased. A static block counter is initialised to store the address of the block to be erased and this is incremented every loop. After each loop the Dataflash's® *Ready / Busy* line is checked to see if the current block erase operation is complete.

## 6.6 Record

The record function is selected by enabling switch 1 on the vibration data logging board. It is used to record data from one of the sensor devices until the AT45DB321C Dataflash® chips memory is full. A record LED is used to indicate when recording takes place.

This function enables the ADC to collect 8 samples from one of the onboard sensors. These values are stored in the ADC buffer. The voltage offset of 1.67V is subtracted from each value in the buffer and then the resulting buffer is converted into a single RMS value. This value is then written to the Dataflash® chip using the *WritetoDataFlash* function described in Section 6.6.1.

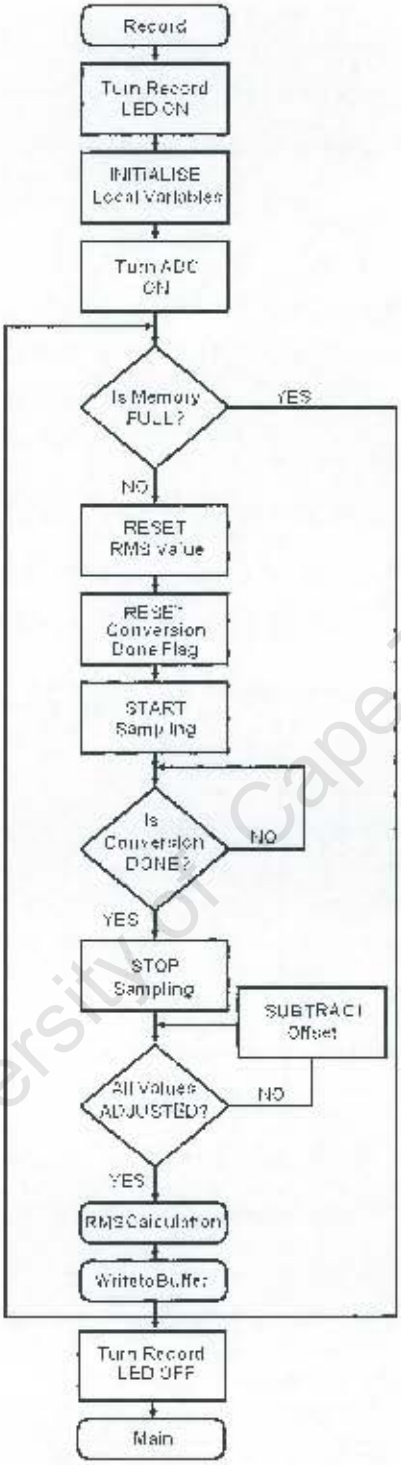



**Figure 6.16: The System Diagram of the *Record* Function**

The *Record* function collects eight samples from the ADC module and calculates the RMS value of these samples. The RMS values are then stored in the AT45DB321C Dataflash® until it is full. This process is then repeated.

### 6.6.1 *The Write to the AT45DB321C Dataflash® Chip Function – WritetoDataflash*

This function is used by the *Record* function to store all the calculated RMS values in the AT45DB321C chip. Data are written to the Dataflash® device's main memory via SRAM buffer 1. This program was adapted from Atmel Corporation Application Note (2005) **[66]**.

This function first checks whether the memory is full before it commences with a write operation. Data are first transferred to SRAM buffer 1. Data are only written to the main memory when Buffer 1 is full.

A transfer to buffer 1 of the AT45DB321C Dataflash® is achieved by enabling the Dataflash® chip ($\overline{CS}=0$), transmitting the write to buffer 1 opcode command $0x84_h$ followed fourteen don't care bits, ten address bits, and eight bits of data via the serial peripheral interface to the Dataflash® chip, and then disabling the chip ($\overline{CS}=1$). The position of the data in the buffer is set by a buffer counter.

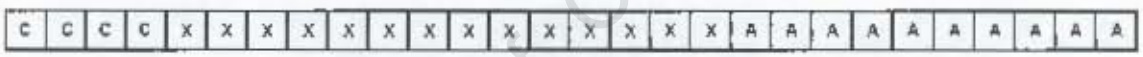



**Figure 6.17: Write to Buffer 1 Instruction Format**

A write to buffer1 instruction for the AT45DB321C Dataflash® consists of a four bit opcode (C), followed by fourteen don't care bits (X) and ten address bits (A).

Once buffer 1 contains 528 bytes of data, a write data from buffer 1 to main memory without erase command is issued. This is achieved by taking $\overline{CS}$ low, transmitting a program main memory without erase opcode $0x88_h$ followed by one reserved bit, thirteen address bits and ten don't care bits. The chip is then disabled. This process is repeated 8192 times until all the pages in the Dataflash® are full. A memory is full flag is then set indicating that data must be transferred or erase before another write operation can take place.

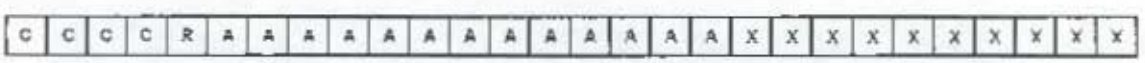



**Figure 6.18: Write Buffer 1 to Main Memory Instruction Format**

A write buffer 1 to main memory instruction for the AT45DB321C Dataflash® consists of a four bit opcode (C), followed by one reserved bits (R), thirteen address bits (A) and ten don't care bits (X).

The function then returns to the record function.

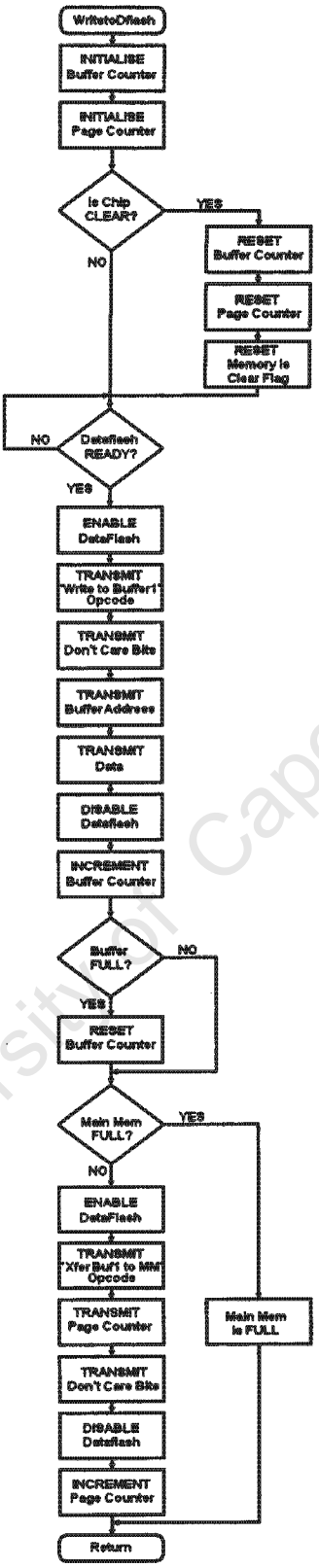



**Figure 6.19: The System Diagram of the *WritetoDataflash* Function**

The *WritetoDataflash* function is used to write a character of data to main memory through buffer 1 until the AT45DB321C Dataflash® chip is full.

## 6.7 Send

The *Send* function is selected by turning Switch 2 on. The send LED then turns on to indicate that the board is sending data. This function reads data from the Dataflash® and then transfers it to a PC running Microsoft® HyperTerminal via UART1. To ensure a seemingly continuous read operation, data are transferred into one SRAM buffer until it is full and then transferred into the next buffer while the first buffer is sending data out of the device.

A page of memory to buffer transfer is accomplished by the *PagetoBuffer* function described in Section 6.7.1. Once the data are in the buffer it is sent to the dsPIC30F6014A by the *ReadfromBuffer* function described in Section 6.7.2. These values are then sent to a PC using the SendtoUART1 function described in Section 6.7.3.

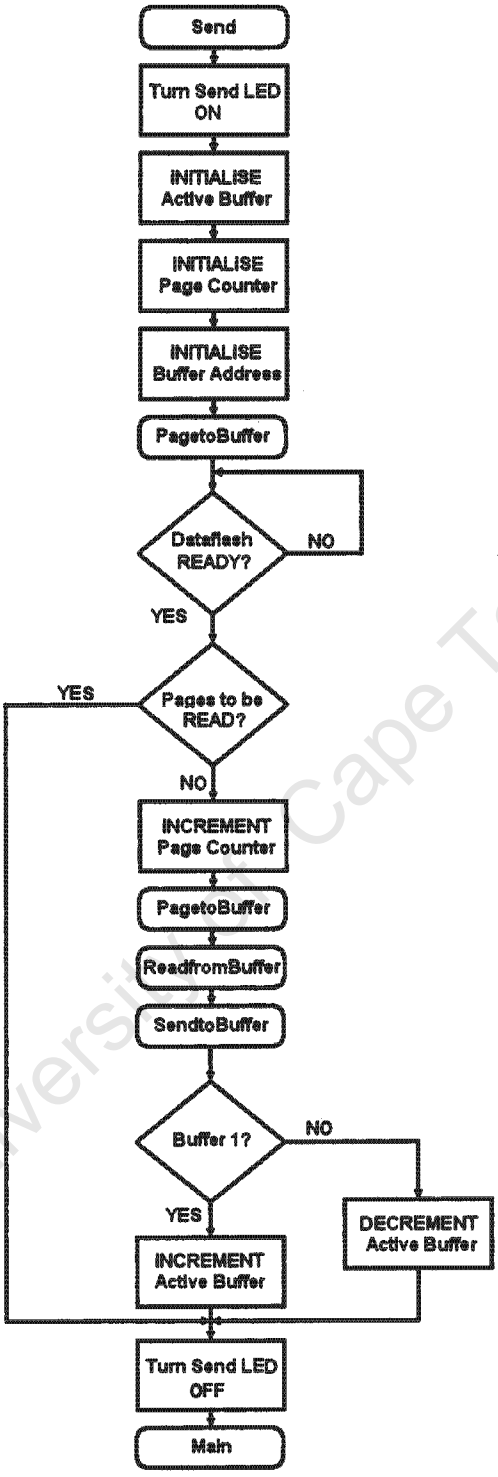



**Figure 6.20: The System Diagram of the *Send* Function**

The *Send* function is used to upload all the data stored in the AT45DB321C Dataflash® chip, to a PC computer via the UART1 module.

### *6.7.1 Main Memory Page to Buffer Transfer Function – PagetoBuffer*

This function was adapted from Atmel Corporation Application Note (2005) **[66]** and is used to transfer one page of data from the active buffer of the AT45DB321C Dataflash® chip.

| C | C | C | C | R | A | A | A | A | A | A | A | A | A | A | A | A | X | X | X | X | X | X | X | X | X | X |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**Figure 6.21: Page to Buffer Transfer Instruction Format**

A page to buffer transfer instruction for the AT45DB321C Dataflash® consists of a four bit opcode (C), followed by one reserved bits (R), thirteen address bits (A) and ten don't care bits (X).

This is achieved by enabling the Dataflash® chip, sending a transfer from main memory to the appropriate buffer opcode command (Buffer 1 – $0x53_h$; Buffer 2 – $0x55_h$), followed by one reserved bit, thirteen address bits and ten don't care bits. The chip is then disabled.

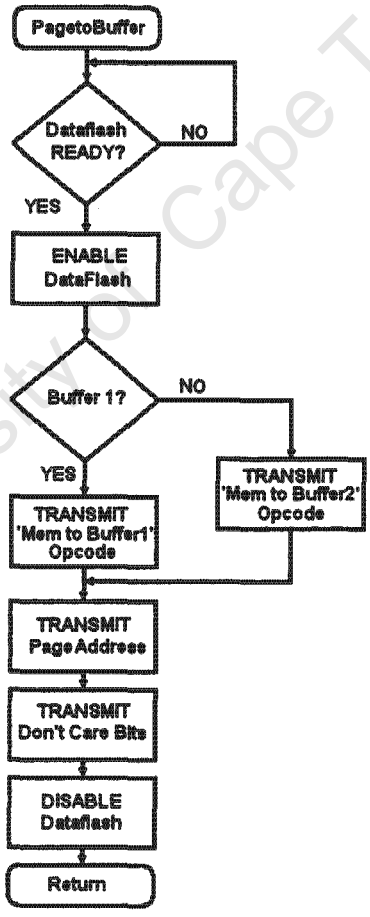



**Figure 6.22: The System Diagram of the *PagetoBuffer* Function**

The *PagetoBuffer* function is used to transfer one page of data from the main memory to the active buffer of the AT45DB321C Dataflash® chip.

### 6.7.2 *Read Data from AT45DB321C SRAM Buffer Function – ReadfromBuffer*

The *ReadfromBuffer* function was adapted from Stowe (2006) **[65]** and it is used to read data of a specified length from a particular address in the chosen buffer.


**Figure 6.23: Read from Buffer Instruction Format**

A read from buffer instruction for the AT45DB321C Dataflash® consists of a four bit opcode (C), followed by fourteen don't care bits (X) and ten address bits (A).

This is achieved by setting $\overline{CS}$ low, transmitting a read from buffer opcode command (Buffer 1 – $0xD4_h$; Buffer 2 – $0xD6_h$), followed by fourteen don't care bits, ten address bits and eight don't care bits. A following don't care bit starts the transfer. A byte of data is transferred out of the device by sending a don't care bit. This is repeated until the buffer is empty. The transferred data are stored in a buffer array. The chip is then disabled.

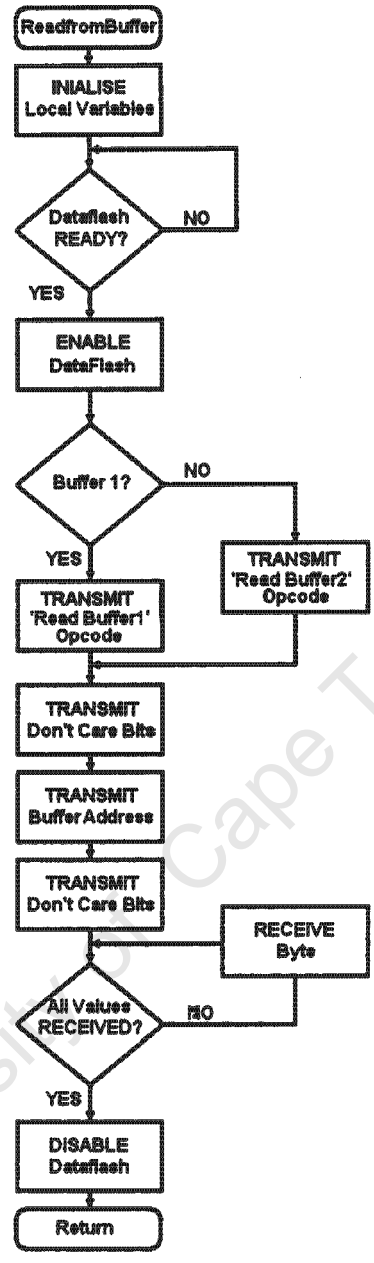





**Figure 6.24: The System Diagram of the *ReadfromBuffer* Function**

The *ReadfromBuffer* function is used to read data of a specified length from a particular address in the active buffer.

### *6.7.3 Send Data to a PC via the UART1 Module Function – SendtoUART1*

The collected data are transferred to a PC via the UART1 module. A function called *SendtoUART1* transfers data from an array to a PC via a serial port. This function takes two consecutive characters from the buffer array and combines them into one 16-bit integer. This

value is then converted into volts and ASCII using *asciiVoltsConv* (Section 6.4.4). Data values are outputted as below:

x.xxx V

The UART1 module is then enabled. An interrupt will occur when the module has transmitted one word of data. The interrupt service routine then clears the interrupt has occurred flag and transmits another word of data. This process is repeated until all the data are sent to the PC.

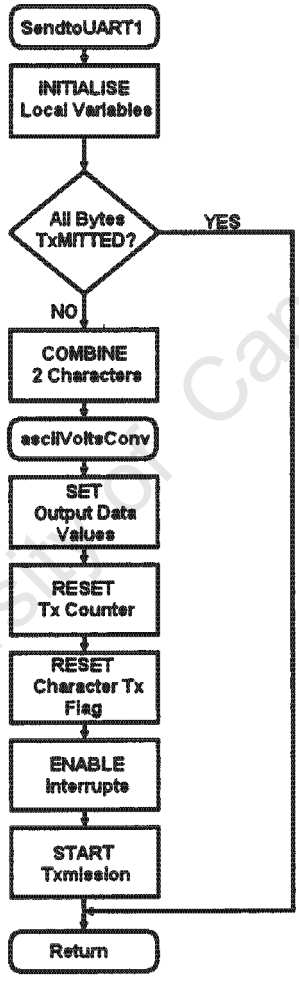



**Figure 6.25: The System Diagram of the *SendtoUART1* Function**

The *SendtoUART1* function is used to transfer collected data to a PC via the UART1 module.

# 7 Hardware Testing

The Vibration Data Logging Board hardware subsystem's functionality was tested to monitor performance under different conditions. The following subsystems were tested the power supply system; the anti-aliasing filtering circuits; the accelerometer board and the microphone circuit. They are discussed in more detail in the next sections.

## 7.1 Vibration Data Logging Board Version 1.0

### *7.1.1 A General Description of the System*

Two vibration data logging prototype boards were designed and constructed during this project. The first vibration data logging board was similar to the Vibration Data Logging Board described in Chapter 5, in all respects except for the following:

1. The board was powered by two AA 1.2V Nickel Metal Hydride (NiMH) Batteries. This input voltage of +2.4V was directly stepped up to +3.3V using an LM2623 based switch mode power supply circuit. No low voltage dropout regulators were used on the output of the SMPS.

2. A ground plane was only included on the bottom side of the Printed Circuit Board as opposed to both sides as in the Vibration Data Logging (VDL) Board Version 2.0.

3. There were no jumpers to separate modules and signals. No test points were provided on the signals.

4. Microchip MCP604 Quad 2.7V to 5.5V Single-Supply CMOS Op Amps [67] were used instead of Microchip MCP6024 operational amplifiers [56] in the anti-aliasing filters.

5. No anti-aliasing filter was included on the output signal from the electret condenser microphone.

6. The output signals from the accelerometer and microphone circuits were not amplified.

7. A dsPIC30F6014 digital signal controller was used as the processor chip. A dsPIC30F6014A digital signal controller, a revised version of the original chip, was used on the later board.

8. A 7.372 MHz crystal was used to provide the system clock for the dsPIC30F6014 processor as opposed to a 10 MHz crystal for the dsPIC30F6014A processor.

9. An external power on reset circuit was included on the $\overline{MCLR}$ line for slow $V_{dd}$ power up (Figure 7.1).

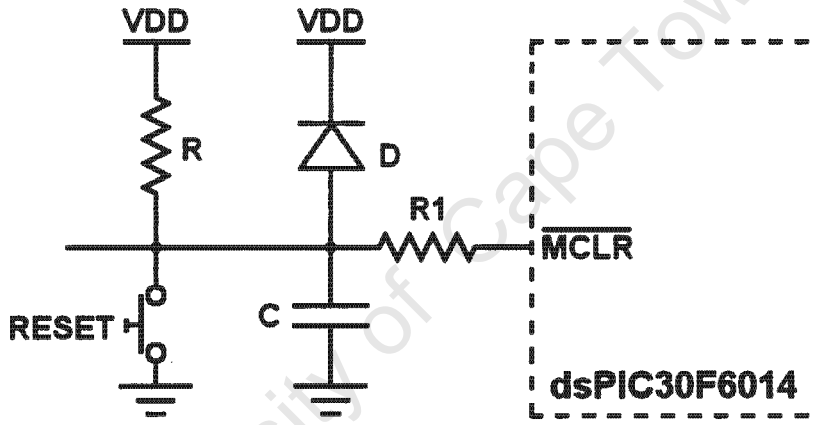



**Figure 7.1: External Power on Reset Circuit used on VDL Board ver. 1.0**

This cicuit is required when the VDD power –slope is too slow (Adapted fromMicrochip Technology Incorporated Data Sheet, 2005) [47].

10. An Atmel AT45DB321B 32-Mbit Serial Dataflash® chip was used as the memory storage device. A newer version of the device the AT45DB321C Dataflash® chip was used on the later board.

11. An external reset push button was included on the Serial Dataflash® chip $\overline{RESET}$ Line.

### *7.1.2 Board Testing and Troubleshooting*

A Hewlett Packard E3630A (0-6V, 2.5A/ 0-±20V, 0.5A) Triple DC Power Supply Unit (PSU) was used to simulate the 2.4V produced by two 1.2V AA NiMH cells. The output range of the PSU was set to 0-6V and the output voltage was set to +2.4V. A Hewlett Packard 54600A 100MHz 2 Channel Oscilloscope was used to monitor voltages and signals through the board.

When power was connected to the Vibration Data Logging Board ver. 1.0 for the first time the PSU indicated that the PSU was sourcing too much current. The output voltage of the switch mode power supply circuit was not reaching the expected 3.3V no matter what input voltage was supplied. This implied there was a problem and possible short circuit on the vibration data logging board.

The following steps were taken to find the source of the problem:

1. All power supply tracks to the digital and analogue systems were cut, therefore isolating the power supply from the rest of the board. The SMPS then appeared to be functional and produced the required output voltage of +3.3V.

2. Each analogue and digital system was then examined and compared to the data sheets and circuit diagrams. It was then discovered that there was an error in the PCB layout and due to this the MCP604 chips were wired incorrectly. The positive supply lines were connected to Pin 11 and the ground connections were connected to Pin 4. Ground should have been connected to Pin 11 and the positive supply to Pin 4.

This problem was solved by cutting the power supply tracks to the MCP604 chips. External wires were then used to connect the correct supply lines to the chips.

An oscilloscope probe was connected to the output of the SMPS and it was discovered that there was a large output voltage ripple of approximately 1V.

When the ICD2 is connected to a device it runs a self test which checks the targets board's voltage levels to ensure correct operation. If the voltage levels are not correct the self test fails. The ICD2 module's self test function failed when it was connected to the Vibration Data

Logging Board ver. 1.0 for the first time. After careful examination it was found that the external power on reset circuit was interfering with the voltage levels on the $\overline{MCLR}$ line. After removing this circuit the self test passed.

Programming of the dsPIC30F6014 processor was now attempted. The ICD2 Module could not connect to the processor in debugging mode. An oscilloscope probe was connected to OSC1 and OSC2 to monitor the clock signal. It was discovered that the crystal oscillator circuit was not oscillating. A range of capacitors with different values were inserted to start the oscillator but it did not oscillate. The crystal was disconnected from the circuit and an external clock circuit was constructed and used as the clock signal. This allowed the dsPIC30F6014 to be tested. It is probable that the clock did not oscillate due to the length of the clock signal lines (approximately 30 mm), reflections at the acute corners in the clock tracks and the ripple on the supply voltage line.

Due to the design flaws in the VDL ver.1.0, a new board was constructed. The design of the new board took all of these problems into account. All issues in the VDL ver1.0 were corrected in VDL ver. 2.0.

## 7.2 The Power Supply

Before the new power supply was designed for the Vibration Data Logging Board ver. 2.0 the following tests were carried out:

1. The LM2623 Switch Mode Power Supply's output RMS voltage was monitored under varying input voltages. The SMPS was configured for a +3.3V output.

2. The LM2623 Switch Mode Power Supply's output RMS voltage was monitored under varying input voltages. The SMPS was configured for a +5V output.

3. Low Dropout Voltage regulators (LDO) TC1264-3.3V and TC2185-3.3V were connected to the output of as power supply unit. The input voltage was varied and the RMS output voltage was monitored.

4. Each of the subsystem's signal to noise ratios were calculated to give an indication of the noise rejection properties of each system.

A Hewlett Packard E3630A (0-6V, 2.5A/ 0-±20V, 0.5A) Triple DC Power Supply Unit (PSU) was used to generate the different input voltages and an Agilent Technologies DSO3152A (100 MHz, 1 GSa/s) Digital Storage Oscilloscope was used to monitor the input and output voltages. At the beginning of testing the Agilent Technologies DSO3152A Digital Storage Oscilloscope was calibrated using the self calibrate function.

### *7.2.1 Switch Mode Power Supply 3.3V Testing*

The SMPS onboard the Vibration Data Logging Board ver. 1.0 was used as the SMPS in this test. The circuit is illustrated in Figure 7.2. The voltage supply ($V_{dd}$) lines which connect the SMPS to the rest of the board were cut, isolating the SMPS.

The switch mode power supply's input was designed to be connected to two NiMH cells (2.4V) and the output voltage was set to 3.3V using resistors R1 and R2. These resistors were selected from E24 resistor values using Equation 5.7.

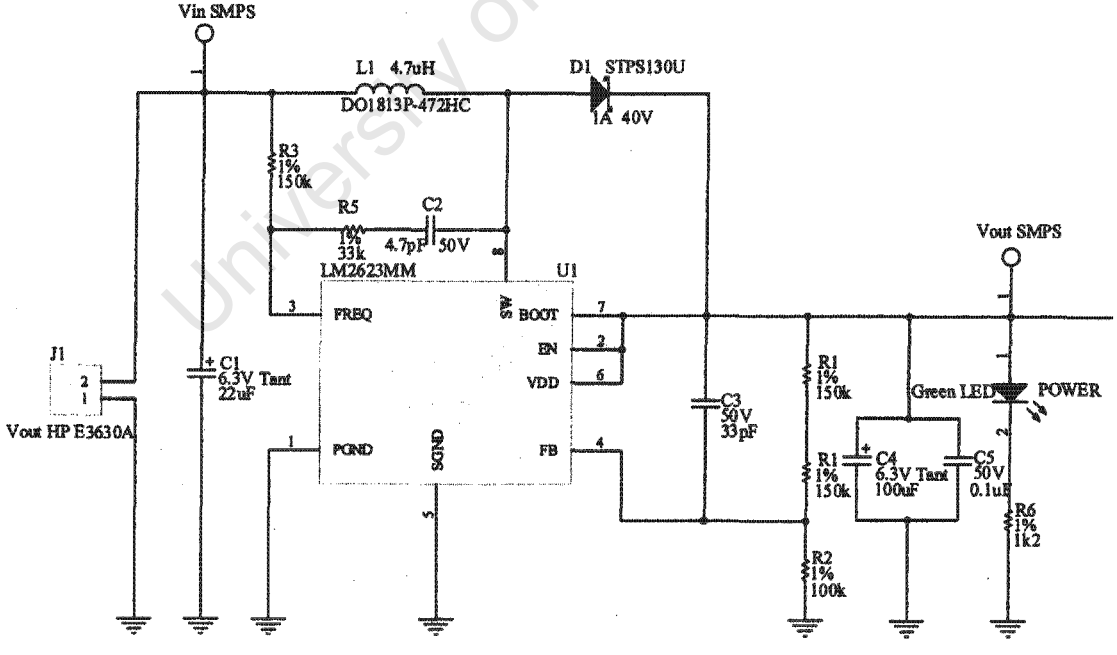



**Figure 7.2: Switch Mode Power Supply (3.3V)**

The Switch Mode Power Supply (SMPS) is found on the Vibration Data Logging Board ver. 1.0. It was isolated from the rest of the circuit by cutting through the lines which connected the supply voltage ($V_{dd}$) to the rest of the board.

The input of the SMPS was connected to the Hewlett Packard E3630A DC Power Supply Unit. The 0-6V, 2.5A output was used. The input ($V_{inSMPS}$) and output ($V_{outSMPS}$) RMS voltages ($V_{rms}$) of the SMPS were monitored using the Agilent Technologies DSO3152A Digital Storage Oscilloscope. The oscilloscope was given 30 minutes to warm up as suggested in the DSO3000 operating manual in order to provide the best results. The ambient temperature was approximately 18°C.

The input was monitored on channel 1 (CH1) and the output was monitored on channel 2 (CH2). Both probes were x10 probes and the acquisition mode of the oscilloscope was set to NORMAL. The scopes automatic measuring mode was used to find the RMS voltage of both channels and this was displayed on the oscilloscope screen.

The input to the switch mode power supply was swept from 0–5V in approximately 100mV intervals and the input and output RMS voltages were recorded at these different values. A graph could then be plotted of the input versus the output RMS voltage.

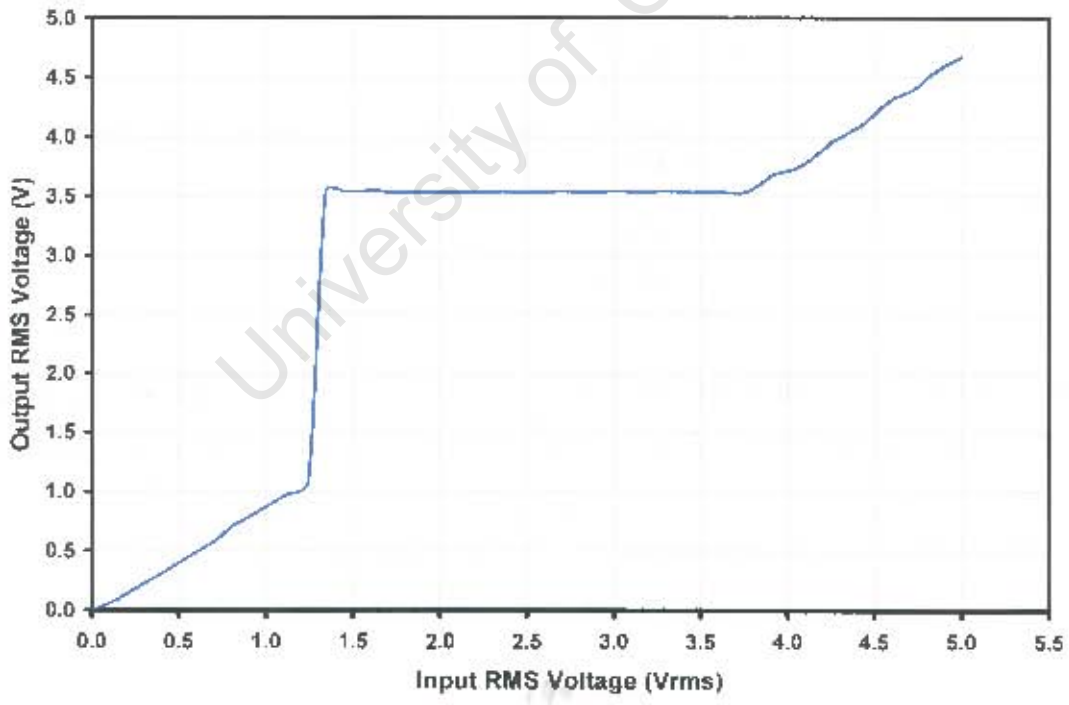



**Figure 7.3: Input RMS Voltage vs. Output RMS Voltage for the SMPS (3.3V)**

The input voltage to the Switch Mode Power Supply (SMPS) was varied between 0 and 5V in 100mV intervals and the output RMS voltages were recorded. The device operates correctly in the 1.2V to 3.5V voltage range.

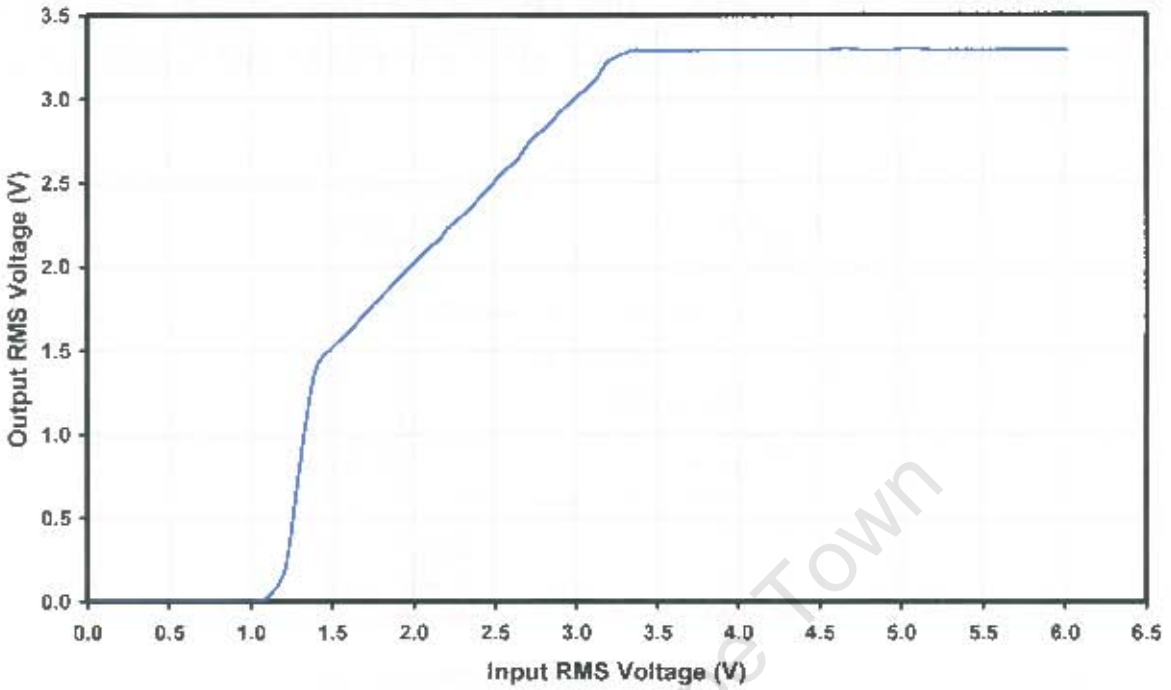



**Figure 7.9: Input RMS Voltage vs. Output RMS Voltage for the TC2185-3.3V**

The TC2185-3.3V turns on at 1.5V. It regulates the supply to 3.3V when the input voltage reaches approximately 3.4V.

The device produces the desired 3.3V output at an input of approximately 3.3V in the above figure. This agrees with the calculated value of 3.302V ($I_L$ = 100$\mu$A) and proves that the device functions as expected.

### *7.2.5 Signal to Noise Ratio Testing*

The signal to noise ratio (SNR) of the SMPS (5V) and the LDOs was tested. The signal to noise ratio gives an indication of the power of a signal to the power of background noise at a point in a circuit and can be calculated with Equation 7.2.

$$SNR = 20\log_{10}\left(\frac{V_s}{V_n}\right) \tag{7.2}$$

Where $SNR$ is the signal to noise ratio in decibels (dB), $V_s$ is the voltage of the signal and $V_n$ is the peak to peak voltage of the noise.

An Agilent Technologies DSO3152A Digital Storage Oscilloscope was used to monitor the signal and noise strength. The signal was monitored on channel 1 (CH1). The probe was in x10 mode and the acquisition mode of the oscilloscope was set to NORMAL. The scopes automatic measuring mode was used to find the RMS voltage of the signal and the peak to peak value of the noise. This was displayed on the oscilloscope screen.

The SMPS (5V) was powered by a +3V input voltage generated by the 0-6V, 2.5A output of the HP E3630A DC Power Supply Unit. The 5V output from the SMPS then provided the input for both low voltage dropout regulators. The RMS voltage of the signal and the peak to peak value of the noise were measured at the input and output of the SMPS (5V), and at the outputs of the LDOs.

**Table 7.1: The Signal to Noise Ratios for Each of the Power Supply Modules**

The signal voltage and peak to peak noise level were measured for each of the power supply modules. The SNRs were then calculated using Eq.7.2. A higher signal to noise ratio is indicates a stronger signal to noise level at that part of the circuit.

| SIGNAL | SIGNAL VOLTAGE (V) | NOISE VOLTAGE (mVp-p) | SIGNAL TO NOISE RATIO (dB) |
|---|---|---|---|
| *Input from HP E3630A* | 3.05 | 60 | 34 |
| *Output from SMPS(5V)* | 5.01 | 50 | 40 |
| *Output from TC1264-3.3V* | 3.34 | 15 | 46 |
| *Output form TC2185-3.3V* | 3.32 | 5.66 | 55 |

The SNR value increases through the power supply modules with the TC2185-3.3V low voltage dropout regulator providing the best value. This test shows that the LDOs provide an enhancement on the SNR of the SMPS which will improve ripple rejection on the power supply line. This will create more stability in the boards functioning.

The remodelling of the vibration data logging board took all of the test results of the VDL ver 1.0 into account and low voltage dropout regulators were included on the output of the SMPS to decrease the amount of output ripple on the supply line. They decreased the output ripple value from approximately 1Vp-p to 60mV which is a more acceptable value.

## 7.3 The Anti-Aliasing Filtering Circuits

The functioning of the anti-aliasing filters onboard the vibration data logging board ver. 2.0 was tested. The input and output voltages were monitored over a wide frequency range in order to plot a gain versus input frequency graph.

### *7.3.1 The 5-pole Low-pass Bessel Filter Testing*

The 5-pole low-pass Bessel anti-aliasing filter used to filter the ACCX output from the accelerometer was utilised in the function test. The accelerometer board was disconnected and the jumper (JP4) linking the anti-aliasing filter circuit to the dsPIC30F6014A processor was removed. This isolated the anti-aliasing filter from the rest of the board.

The 0-6V, 2.5A output of a Hewlett Packard E3630A DC Power Supply Unit was connected to the SMPS onboard the vibration data logging board. It was set at +3V. This powered the power supply module of the board which in turn powered the anti-aliasing filter with +3.3V single supply voltage.

A Hewlett Packard E3312A, 15 MHz Function/Arbitrary Waveform Generator was used to produce the input signal to circuit. The input was set as a 1Vp-p sine wave with a DC offset of +1.61V. This was then connected to test pin ACCX as the input to the filter and test pin ACCX_ADC was the output of the filter. The frequency of the sine wave was swept from 0-25 kHz in 200Hz intervals. The corresponding input and output voltages were recorded. The gain of the filter was then calculated using Equation 7.3 and plotted against input frequency (Figure 7.10).

$$Gain(dB) = 20\log_{10}\left(\frac{V_{out}}{2.2V_{in}}\right) \tag{7.3}$$

An Agilent Technologies DSO3152A Digital Storage Oscilloscope was used to monitor the input and output voltages. The input voltage was monitored on channel 1 (CH1) and the output voltage was monitored on channel 2 (CH2). The probe was in x10 mode and the acquisition mode of the oscilloscope was set to NORMAL. The scopes automatic measuring mode was used to find the voltage of the input and output voltage signals.

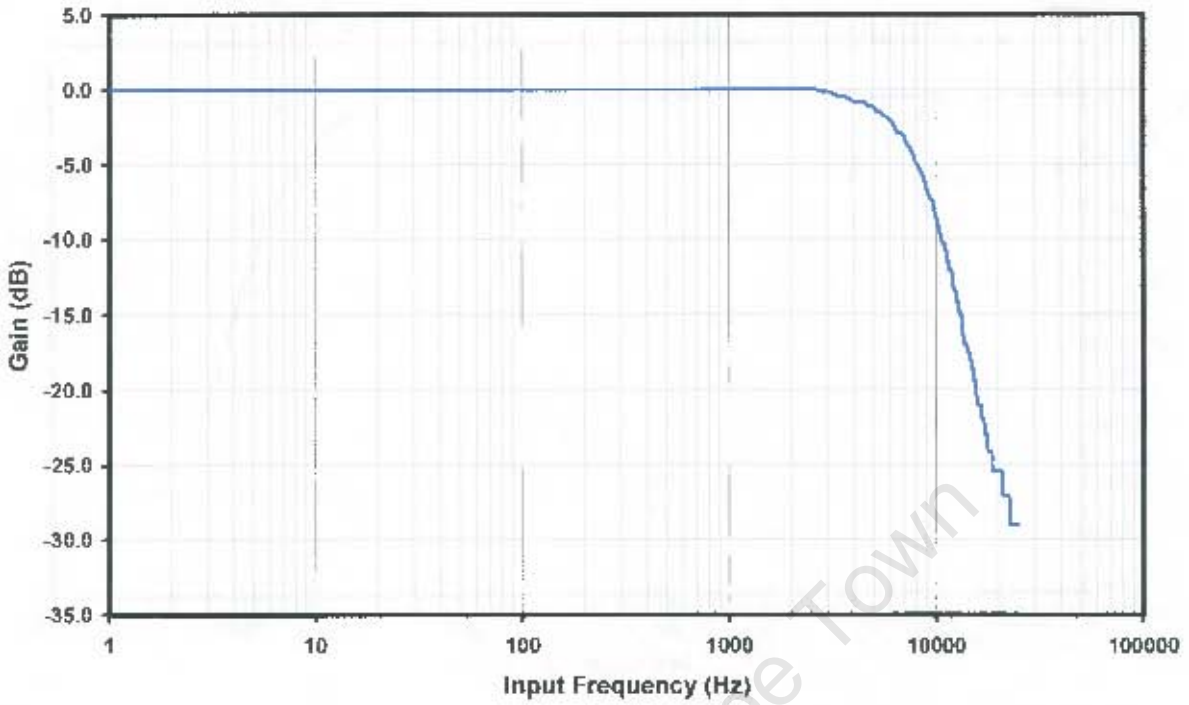



**Figure 7.10: Bode Plot for the 5-Pole Low-pass Bessel Filter**

One of the 5-pole low-pass Bessel anti-aliasing filters onboard the vibration data logging board was used to test filter functioning. The input of the filter was connected to a signal generator outputting a 1Vp-p sine wave offset at 1.61V. The input frequency was sweep from 0-25 kHz in 200 Hz intervals. The gain of the circuit was then compared to input frequency.

The filter circuit has a gain of 2.2V set by an inverting amplifier. The output voltage was normalised by dividing it by 2.2. The gain of the circuit was then plotted against frequency using a logarithmic scale. The graph illustrated in Figure 7.10, follows the typical low-pass Bessel filter curve and the -3dB value is at approximately 7 kHz.

### *7.3.2 The RC Low-pass Filter Testing*

The RC low-pass anti-aliasing filter used to filter the output of the electret condenser microphone was tested using the same procedure as the 5-pole low-pass Bessel filter. The gain was calculated using the different input and output voltages and was plotted against frequency using a logarithmic scale Figure 7.11.

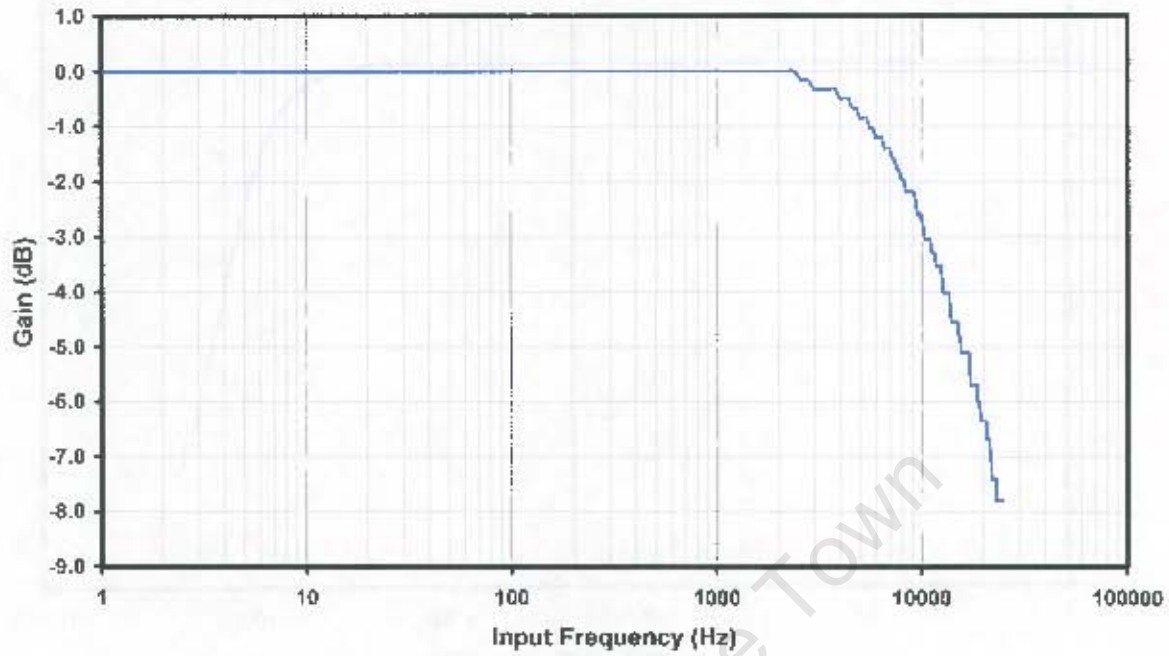



**Figure 7.11: Bode Plot for the RC Low-pass Filter**

The RC low-pass anti-aliasing filters onboard the vibration data logging board was used to test filter functioning. The input of the filter was connected to a signal generator outputting a 1Vp-p sine wave offset at +1.61V. The input frequency was sweep from 0-25 kHz in 200 Hz intervals. The gain of the circuit was then compared to input frequency.

The filter circuit has a gain of 2.2V set by an inverting amplifier. The output voltage was normalised by dividing it by 2.2. The voltage gain values were calculated using Equation 7.3 and then plotted against frequency using a logarithmic scale. The graph illustrated in Figure 7.11, follows the typical low-pass RC filter curve with a very slow drop-off rate. The -3dB value is at approximately 10 kHz.

# 8 Software Testing

The software modules and subsystems used onboard the vibration data logging board were each tested separately. The functionality of every module could then be verified. A project called Initial Testing was created in MPLAB® 7.30. Each test was written in a different file and the code can be found in Appendix L. The following header files were included in the project:

1. p30f6014a.h [64]
2. math.h
3. dsPIC30F6014APins.h
4. AT45DB321C.h
5. common.h

The dsPIC30F6014A was configured using macros defined in the p30f6014a.h header file. The oscillator was set to XT w/PLL 8x; the watchdog timer was disabled; master clear on reset was enabled and general code segment protect was disabled.



## 8.1 The Light Emitting Diodes

A program called *TestLEDs.c* was written to test the indicator LEDs onboard the vibration data logging board ver.2.0. This program sets the appropriate pins in Port C and Port G as outputs and sets their initial value to zero. All the indicator lights will then be turned on for the duration of the program by setting the values of the output port pins to one.

All of the LED indicator lights functioned correctly.

## 8.2 The Universal Asynchronous Receiver Transmitter

The UART1 module onboard the vibration data logging board was tested using a program called *TestUART.c*. This module is the primary serial communication link, between the

Vibration Data Logging Prototype Board and to a personal computer. Microsoft® Hyper Terminal is used to view the data. All the jumpers (JP8) on the transmit and receive signal lines must be connected.

The UART1 module is initialised with an *InitUART1* function. This function enables the UART1 module; enables interrupts to occur when the transmit buffer is empty; and enables 8-bit data with no parity and one stop bit. Data are transferred at 9600 baud.

A variable containing the 'A' character is initialised and written to the *U1TXREG*, where it is transmitted continuously to a PC running Microsoft® Hyper Terminal.

The UART1 module onboard the dsPIC30F6014A was able to transmit 'A' character to a PC running Microsoft® Hyper Terminal, correctly. This proves that the UART1 module is operational.



## 8.3 The Analogue to Digital Converter

### *8.3.1 Variable DC Voltage Input Testing*

*TestADC12Potentiometer.c* was written to test the basic functioning of the 12-bit analogue to digital converter module onboard the dsPIC30F6014A.

A DC voltage (0-3.3V) was produced by a potentiometer connected across the analogue power supply. Jumper (JP6) was removed and the output from the potentiometer connected to test pin MIC_ADC which is linked to the ADC module in the dsPIC30F6014A.

The voltage value of potentiometer is sampled manually by the ADC module every 20ms. This is achieved by inserting a delay loop between set and clearing of the *SAMP* bit in *ADCON1*. This value is then sent to a PC running Microsoft® Hyper Terminal via UART1. The 12-bit integer value of the potentiometer is first converted to binary coded decimal and then to a suitable ASCII character before transmission.

The ADC module converted the potentiometer to a number with two decimal places and this was received by a PC running Microsoft® Hyper Terminal correctly. This shows that the ADC module is functional.

### 8.3.2 *MAX6608 Temperature Sensor Testing*

The code used to test the MAX6608 analogue temperature sensor (*TestMAX6608.c*) is very similar to that of *TestADC12Potentiometer.c*. This program tests the functioning of both the 12-bit ADC module of the dsPIC30F6014A and the low voltage analogue temperature sensor (MAX6608) onboard the Vibration Data Logging Prototype Board. Jumper JP7 must be connected to link the temperature sensor to the processor.

The voltage output from the MAX6608 is sampled manually by the ADC module every 20ms. This value is then converted to °C using Equation 5.6. This value is then converted to binary coded decimal and the suitable ASCII character and sent to a PC running Microsoft® Hyper Terminal via UART1.

The ADC module converted the analogue temperature from the MAX6608 to the expected value and this was received by a PC running Microsoft® Hyper Terminal correctly.

### 8.3.3 *Electret Microphone Testing*

This program (*TestMIC.c*) tests the functioning of both the 12-bit Analogue to Digital Converter Module and the Electret Condenser Microphone which can be connected to the socket J2 on the Vibration Data Logging Prototype Board.

The output from the microphone is automatically sampled at 20 kHz by setting the *ADCS<5:0>* and *SAMC<4:0>* bits of the *ADCON3* register. An interrupt occurs once eight samples have been collected. These values are then put into an array which forms the input to the RMS conversion function. This value is then converted to a binary coded decimal number and in the suitable ASCII format. It is then outputted to a PC running Microsoft® Hyper Terminal via the UART1 module.

### 8.3.4 *ADXL202E Accelerometer Testing*

The *TestACC.c* program works very similarly to *TestMIC.c* except that data are collected alternatively from two different sources.

The outputs from the accelerometer are automatically sampled at 30kHz by setting the *ADCS<5:0>* and *SAMC<4:0>* bits of the *ADCON3* register. An interrupt occurs once every sixteen samples and these values are separated into two different arrays which form the input to the RMS conversion function. These values are then converted to binary coded decimal and he suitable ASCII format. It is then outputted to a PC running Microsoft® Hyper Terminal via the UART1 module.

## 8.4 The Memory Circuit

### 8.4.1 *Memory Status Register Testing*

This program (*TestMEMORYSTATUS.c*) tests the Atmel 32-Mbit Dataflash®, AT45DB321C (U9) onboard the Vibration Data Logging Prototype Board. The Dataflash® chip is connected to the dsPIC30F6014A via one of the Serial Peripheral Interface (SPI1) modules on board the MCU.

The AT45DB321C has an onboard status register. The contents of this register are sent to the dsPIC30F6014A and are compared to the expected value of 0xB4. If the Dataflash® chip is intact LED2 is switched ON if it is not intact LED4 is switched ON.

The correct value of the status register was sent back to the dsPIC30F6014A and LED2 was switched on. It therefore appears that the memory chip works correctly as the correct value of the status is register is outputted from the AT45DB321C.

### 8.4.2 *Memory Erase/Write/Read Testing*

This program (*TestMEMORY.c*) tests the erase, write to and read from the SRAM buffers onboard the Atmel 32-Mbit Dataflash®, AT45DB321C chip.

An array of 20 characters is created. These are then written to buffer 1 onboard the AT45DB321C chip. Once the write is complete, the data are then read back to the dsPIC30F6014A from the memory chip. A character from the transmitted array and the corresponding character from the read array are compared. If these values are the same the Dataflash® chip is intact and LED2 is switched ON, if it is not LED4 is switched ON.

The Erase operation of the AT45DB321C was tested, the Erase LED switched on for approximately 20 seconds. This means that this operation appears to work. However, when the Write/Read operation was tested, an incorrect value was outputted back to the dsPIC30F6014A SPI1BUF register. This value was often 0xFF which is the reset state of the chip.

Another AT45DB321C was tested and produced the same results. The complete memory functioning could therefore not be tested.

# 9 Conclusions

## 9.1 Restatement of Project Objectives

Overland conveyor belt systems form a vital part of modern transportation systems in the mining and mineral processing industries. It is vital that the system is well maintained in order to minimise system downtime and maximise profit. The conveyor belt is the single most expensive item in the system. It must be monitored to pick up potential problems before they cause belt failure. The majority of conveyor belt condition monitoring methods identify belt failure rather than belt failure causes.

The purpose of this project was to research and design a belt condition monitoring board which could be embedded in a conveyor belt. This would then be used to monitor the condition of the conveyor idlers whose failure can result in major system damage. The venture was split into two areas of research: the design of a vibration data logging board, and the design of a power generation system. This thesis focused on the design and construction of a DSP vibration data logging prototype board, while S.A. Williams investigated the design of a power generation system.

The objectives of this project were to: conduct an extensive literature review of different maintenance methods with a specific focus on rolling element bearing monitoring; to understand the design of an overland conveyor system; to take the knowledge learnt from the literature and design and construct a digital signal processing vibration data logging system; and finally to test the system in simulation.

## 9.2 Context of Design

Many different aspects of overall maintenance were examined in this thesis, from general maintenance theory to signal processing techniques used in the condition monitoring equipment.

Condition-based monitoring systems are the best long term solution in conveyor belt maintenance. They give maintenance managers advanced warning of potential problems and provide vital data which can be used in improving future designs. A parameter monitoring data logging sensor is therefore a good way to monitor the condition of system.

Rolling element failure is one of the largest causes of rotating machinery breakdowns. Conveyor belt idlers and pulleys are supported by rolling elements and therefore these need to be monitored.

Various monitoring methods were investigated and it was found that mechanical vibration of the bearing and bearing housing is a good indicator of bearing defects. However, there were few studies on remote vibration monitoring. This area of research needs to be investigated further. As the sensor would be located in the belt, signal damping and surrounding environmental vibrations will hide small vibration signals emanating from failing bearings. Therefore late damage will only be successfully detected by time and frequency domain methods as these techniques are unable to distinguish the small vibration signals that occur during early damage from environmental noise.

RMS vibrational acceleration was selected as the main method which was used to measure mechanical vibration. This technique was selected as it is simple to implement; not affected to the same extent by conveyor movement as frequency measurements; and has been shown to be successful in detecting late damage in bearings [19]. Vibrational acceleration must be measured in the direction of the belt travel, and vertical to the belt, as a faulty idler bearing will cause excessive idler vibration resulting in increased horizontal and vertical acceleration of the belt.

Sound can also be used as an indicator of advanced bearing wear as a badly worn bearing may create an audible sound. Temperature is another parameter which can be used as a wear indicator. A belt's temperature increases due to increased fiction on badly worn idlers.

## 9.3 Fulfilment of Requirements

A vibration data logging board with digital signal processing board capability was designed and constructed. It contained three onboard condition monitoring sensors: a dual axis accelerometer which was used to monitor mechanical vibration, an electret condenser microphone which picked up audible sound and an analogue temperature sensor which measured the temperature of the board. Signals from the accelerometers were filtered with 5-pole low-pass Bessel filters and the signal from the electret microphone was filtered with a simple RC filter.

The vibration data logger was battery powered making the device portable. A switch mode power supply was used to efficiently regulate the batteries supply. Low voltage dropout regulators provided post regulation and filtered out ripple on the supply lines.

The board contains a dsPIC30F6014A digital signal controller with many different peripheral modules. The 12-bit analogue to digital controller, the universal asynchronous receiver transmitter and the serial peripheral interface modules were used. This processor had enough onboard memory to store all the software that was written for the board and enough processing power to perform all the required mathematical operations.

Three switches are used to select the whether the board records, sends or erases data. Indicator LEDs are used to show which process is in operation.

Data are sampled at different rates according to the device. RMS calculations are preformed on samples from the accelerometer and electret condenser microphone. Up to sixteen minutes of data can be recorded. Data are stored in an Atmel 32-Mbit AT45DB321C serial Dataflash® device.

The vibration data logger can upload data to a personal computer via a serial link. Data can be viewed using Microsoft® HyperTerminal.

The main circuit was constructed on a 150 x120 x 3 mm double sided plated through hole board. Large separate ground planes on both sides of the board simplified circuit layout and reduces noise on the power supplies.

## 9.4 Hardware Performance

Two vibration data logging boards were constructed. The design of vibration data logging board ver. 2.0 corrected all the faults found in ver 1.0 therefore improving on the overall design of the system. All hardware sections could be isolated from the rest of the board by disconnecting the relevant jumpers. This simplified testing and trouble shooting. Test pins were provided on all the major signals.

The power supply system and anti aliasing filters were thoroughly tested. The low voltage dropout regulators significantly reduced noise output from the switch mode power supply. Therefore a switch mode power supply with low voltage dropout regulators forms low input current, efficient, low noise power supply system.

The anti-aliasing filters functioned as required and successfully filtered out high frequency signal components.



## 9.5 Software Performance

The software modules and subsystems used onboard the vibration data logging board were each tested separately. The functionality of every module could then be verified. A project called Initial Testing was created in MPLAB® 7.30. Each test was written in a different file. The following components were tested:

1. Light Emitting Diodes – These LEDs functioned correctly.

2. Universal Asynchronous Receiver Transmitter Module – This module was able to successfully transmit a character to a PC running Microsoft® HyperTerminal.

3. Analogue to Digital Converter – Four programs were written to test the ADC module when connected to different inputs. The module was able to sample each source correctly.

4. AT45DB321C Dataflash® Chip –Two programs were written to test this device. The device's integrity was verified by polling the status register and comparing it to the expected value. It was found that this value was correct. The second program was used to

5. write and read an array of data to Dataflash® Chip. The program could also erase the chip. The erase function seemed to work, however the data that was received back from the Dataflash® Chip was not the same data that had been written to the chip. This operation failed and prevented further programming of the vibration data logging device.

Software was written for the integration of the all the functions used to control the modules on the vibration data logging board. Due to the failure of the Write/Read operation of the AT454DB321C Dataflash® this integration program could not be implemented.

# 10 Recommendations

## 10.1 Recommendations for Current Study

### *10.1.1 Conduct Further Testing on AT45DB321C Dataflash® Chip*

The AT45DB321C Dataflash® chip must be further tested to determine the cause of the write/read fault. The write/read problem must be solved in order for the Vibration Data Logging Board to be fully functional. If the problem cannot be found, or if there is a conflict between the SPI timing of the dsPIC30F6014A and the AT45DB321C, an alternative memory storage device must be used.

### *10.1.2 Mechanical Vibration as a Primary Input Source*

Mechanical vibration must be used as the primary input to the vibration data logging device, as sound is redundant in this case as it lies in the same frequency range and is more difficult to detect. Sound is very hard to detect in noisy industrial environments.

### *10.1.3 Convert Serial Data Link to a Wireless Data Link*

The standard serial link between the vibration data logging board and the PC must be changed to a wireless link. This would increase the portability of the device as it would not have to be close to or brought back to a PC for data upload.

### *10.1.4 Create a User-friendly Software Interface for a PC*

A simple to use software package to view and analyse collected data must be written for a personal computer. This would provide the user with a simpler way to access and monitor bearing condition.

### *10.1.5 Provide Battery Recharging Function*

A battery recharging function should be incorporated into the device. This would save on the cost of batteries and would make embedding the device in the conveyor belt more feasible. This would mean that a suitable recharge system be developed.

### *10.1.6 Provide an Onboard Power Management System*

An onboard power management system could be implemented that would put idle chips in sleep mode when they are not in use. This would improve power saving and increase battery life of the device.

### *10.1.7 Test the System on a Variety of Bearings*

The system must be tested on many different types of bearings which are run from new to failure. This would provide a large data set which can be used to improve on the design and the monitoring system in general.

### *10.1.8 Construct a Smaller and More Flexible Board*

Once all of the above recommendations have been taken into account, a smaller, more flexible board must be constructed which could be embedded in a belt and then tested on the actual system.

## 10.2 Recommendations for Further Studies

### *10.2.1 Investigate Remote Vibration Sensing on Rolling Element Bearings*

Not many studies have been carried out on remote vibration sensing of rolling element bearings. All studies focus on measuring the vibration of the bearing in close proximity through its housing. A further study should be under taken which investigates the possibility of measuring vibration of a bearing at a distance. These results would lead to an improved design for the vibration data logging board.

### *10.2.2 Investigate Wavelet Transforms as a Possible Monitoring Technique*

Wavelet transforms should be investigated as an alternative to traditional time and frequency domain methods for vibrational data analysis. Current studies seem to be moving towards this technique and further investigation into the feasibility of using wavelets in remote sensing should be investigated.

# 11 List of References


[1] **NORDELL, L.K.** ZISCO installs worlds longest troughed belt 15.6 km horizontally curved overland conveyor. *CKit - The Bulk Materials Handling Knowledge Base Paper.* www.ckit.co.za

[2] **SMITH, W.A.C. & SPRIGGS, G.H.** (1981). Long overland conveyors. *Proceedings of the BELTCON 1 Conference.* International Materials Handling Conference: Johannesburg, South Africa

[3] **OWEN, P.** (1997). Condition monitoring for conveyors. *Proceedings of the BELTCON 9 Conference.* International Materials Handling Conference: Johannesburg, South Africa

[4] **BROWN, B.** (1983). State of the art usage of the Harrison conveyor belt monitor. *Proceedings of the BELTCON 2 Conference.* International Materials Handling Conference: Johannesburg, South Africa

[5] **DAVIES, G.J.** (1987). Modern concepts in belt rip detection for steel-cord reinforced conveyor belts. *Proceedings of the BELTCON 4 Conference.* International Materials Handling Conference: Johannesburg, South Africa

[6] **TAPSON, J.** (2004). *Personal Communication*

[7] **DING, J.J. & AL-JUMAILY, A.** (2000). A linear regression model for the identification of unbalance changes in rotating machines. *Journal of Sound and Vibration.* **231**(1): 125-144

[8] **DEKKER, R.** (1996). Applications of maintenance optimization models: a review and analysis. *Reliability Engineering and Safety Systems.* **51**(3): 229-240

[9] **KHAN, F.I. & HADDARA, M.M.** (2003). Risk-based maintenance (RBM): A quantitative approach for maintenance/inspection scheduling and planning. *Journal of Loss Prevention in the Process Industries.* **16**(6): 561-573

[10] **RAO, B.K.N.** (1996). *Hand book of Condition Monitoring.* 1st Ed. Oxford, U.K.: Elsevier Advanced Technology

[11] **HORNER, R.M.W., EL-HARAM, M.A. & MUNNS, A.K.** (1997). Building maintenance strategy: A new management approach. *Journal of Quality in Maintenance Engineering.* **3**(4): 273-280

[12] **SURTEES, A.J.** (1995). Conveyor system commissioning, maintenance and failure analysis using black box techniques. *Proceedings of the BELTCON 8 Conference.* International Materials Handling Conference: Johannesburg, South Africa

[13] **REIMCHE, W., SÜDMERSEN, U., PIETSCH, O., SCHEER, C. & BACH, F-W.** (2003). Basics of vibration monitoring for fault detection and process control. *Proceedings of the 3rd Pan-American Conference for NDT:* Rio de Janeiro, Brazil

[14] **TUCKEY, K., WOMACK, R. & STOLZ, H.** (1985). Maintenance on belt conveyors - A practical approach to this vital link in continuous production. *Proceedings of the BELTCON 3 Conference.* International Materials Handling Conference: Johannesburg, South Africa.

[15] **MUTOU, Y., FUJIU, K., MATSUSHIMA, T., OGAWA, T. & NODA, B.** (2000). Schematics for the cage for a ball bearing. *U.S. Patent #6074099*

[16] **LI, Y., BILLINGTON, S., ZHANG, C., KURFESS, T., DANYLUK, S. & LAING, S.** (1999). Adaptive prognostics for rolling element bearing condition. *Mechanical Systems and Signal Processing.* **13**(1): 103-113

[17] **HEEMSKERK, R.S. & ALLENSPACH, E.** (1987). Rolling bearings in bulk conveyors. *Proceedings of the BELTCON 4 Conference.* International Materials Handling Conference: Johannesburg, South Africa

[18] **ESCHMANN, P, HASBARGEN, L. & WEIGAND, K.** (1958). *Ball and roller bearings: Their theory, design and application.* München: Oldenbourg

[19] **BIRCH, D.** (1994). A review of vibration signal processing techniques for use in a real time condition monitoring system. *M.Sc Thesis.* Department of Electrical Engineering. University of Cape Town

[20] **SKF RELIABILITY SYSTEMS.** (1981). Bearings for bulk conveyors. *SKF Reliability Systems "S-Range" Handbook Series.* **SKF 3209**

[21] **FRITTELLA, A. & COHEN, M.G.** (1991). Conveyor Idler Standards. *Proceedings of the BELTCON 6 Conference.* International Materials Handling Conference: Johannesburg, South Africa

[22] **TANDON, N. & CHOUDHURY, A.** (1999). A review of vibration and acoustic measurement methods for the detection of defects in rolling element bearings. *Tribology International.* **32**(8): 469-480

[23] **TANDON, N. & CHOUDHURY, A.** (1997). An analytical model for the prediction of the vibration response of rolling element bearings due to a localized defect. *Journal of Sound and Vibration.* **205**(3): 275-292

[24] **LI, C.J. & MA, J.** (1997). Wavelet decomposition of vibrations for detection of bearing-localized defects. *NDT&E International.* **30**(3): 143-149

[25] **KIRAL, Z. & KARAGÜLLE, H.** (2003). Simulation and analysis of vibration signals generated by rolling element bearing with defects. *Tribology International.* **36**(9): 667-678

[26] **MCFADDEN, P.D. & SMITH, J.D.** (1984). Model for the vibration produced by a single point defect in a rolling element bearing. *Journal of Sound and Vibration.* **96**(1): 69-82

[27] **WHITE, M.F.** (1984). Simulation and analysis of machinery fault signals. *Journal of Sound and Vibration.* **93**(1): 95-116

[28] **JANTUNEN, E.** (2002). A summary of methods applied to tool condition monitoring in drilling. *International Journal of Machine Tools & Manufacture.* **42**(9): 997-1010

[29] **GUSTAFSSON, O.G. & TALLIAN, T.** (1961). Detection of damage in assembled rolling element bearings. *ASLE Preprint 61-AM 3B-1.* 16th ASLE, Philadelphia, PA

[30] **DYER, D. & STEWART, R.M.** (1978). Detection of rolling element bearing damage by statistical vibration analysis. *Transactions of the ASME, Journal of Mechanical Design.* **100**(2): 229-235

[31] **MCFADDEN, P.D. & SMITH, J.D.** (1984). Vibration monitoring of rolling element bearings by the high frequency resonance technique - a review. *Tribology International.* **17**(1): 3-10

[32] **PENG, Z.K. & CHU, F.L.** (2004). Application of the wavelet transform in machine condition monitoring and fault diagnosis: a review with bibliography. *Mechanical Systems and Signal Processing.* **18**(2): 199-221

[33] **TANDON, N. & NAKRA, B.C.** (1990). The application of the sound intensity technique to defect detection in rolling-element bearings. *Applied Acoustics.* **29**(3): 207-217

[34] **CHOUDHURY, A. & TANDON, N.** (2000). Application of acoustic emission technique for the detection of defects in rolling element bearings. *Tribology International.* **33**(1): 39-45

[35] **DAVIES, G.J.** (1981). Aspects of conveyor belting. *Proceedings of the BELTCON 1 Conference.* International Materials Handling Conference: Johannesburg, South Africa

[36] **HARRISON, A.** (1987). Future design of belt conveyors using dynamic analysis. *Bulk Solids Handling.* 7(3): 375-379

[37] **LODEWIJKS, G.** (2003). Report of the static and dynamic analyses of the belt conveyor CV-08 for Savmore Colliery Maquesa Shafts Overland project. *Report for Bateman Materials Handling Ltd.* **DAR-060103-01**

[38] **HARRISON, A. & ROBERTS, A.W.** (1984). Technical requirements for operating conveyor belts at high speed. *Bulk Solids Handling.* 4(1): 99-104

[39] **PAGE, T.P.T.** (1987). Large conveyors - The case for total system design. *Proceedings of the BELTCON 4 Conference.* International Materials Handling Conference: Johannesburg, South Africa

[40] **HARRISON, A.** (1988). A new development in textile-ply belt monitoring. *Bulk Solids Handling.* 8(2): 231-233

[41] **GOODYEAR INDUSTRIAL PRODUCTS.** www.goodyearindustrialproducts.com/conveyorbelts/products/flexsteel.html#fspec

[42] **OSBORN ENGINEERED PRODUCTS SA (PTY) LTD.** www.osborn.co.za/products/specs/osb_conv_idler_specs.htm

[43] **MÖLLER, J.J.** (1981). Protect your conveyor investment. *Proceedings of the BELTCON 1 Conference.* International Materials Handling Conference: Johannesburg, South Africa

[44] **GOODYEAR INDUSTRIAL PRODUCTS.** Conveyor belt maintenance manual. *Goodyear Industrial Products Manuals.*

[45] **ANALOG DEVICES INCORPORATED.** (2002). Low-cost ±2 g dual-axis accelerometer with duty cycle, ADXL202E. *Analog Devices Incorporated Data Sheet.* Rev. A

[46] **MAXIM INTEGRATED PRODUCTS.** (2001). MAX6607/MAX6608 low-voltage analog temperature sensors in SC70 and SOT23 packages. *Maxim Integrated Products Data Sheet.* **19-2040**; Rev. 1; 6/01

[47] **MICROCHIP TECHNOLOGY INCORPORATED.** (2005). dsPIC30F6011A/ 6012A/6013A/6014A data sheet, high-performance digital signal controllers. *Microchip Technology Incorporated Data Sheet.* **DS70143B**

[48] **ATMEL CORPORATION.** (2003). AT45DB321B, 32-megabit 2.7-volt only DataFlash®. *Atmel Corporation Data Sheet.* Rev. 2223E-DFLASH-11/03

[49] **SIPEX CORPORATION.** (2005). SP3222E, SP3232E true +3.0 to +5.0V RS-232 transceivers. *Sipex Corporation Data Sheet.*

[50] **NATIONAL SEMICONDUCTOR CORPORATION.** (2004). LM2623 general purpose, gated oscillator based, DC/DC boost converter. *National Semiconductor Corporation Data Sheet.* **DS200388**

[51] **MICROCHIP TECHNOLOGY INCORPORATED.** (2002). TC1264 800mA fixed output CMOS LDO. *Microchip Technology Incorporated Data Sheet.* **DS21375B**

[52] **MICROCHIP TECHNOLOGY INCORPORATED.** (2004). TC2014/2015/ 2185 50 mA, 100 mA, 150 mA CMOS LDOs with shutdown and reference bypass. *Microchip Technology Incorporated Data Sheet.* **DS21662D**

[53] **WILSON, J.** (1999). A practical approach to vibration detection and measurement - Part 1: Physical principles and detection techniques. *Sensors.* **16**(2): 12-27

[54] **HOROWITZ, P. & HILL, W.** (1995). *The art of electronics.* 2nd Ed. Cambridge: Cambridge University Press

[55] **KARKI, J.** (2000). Active low-pass filter design. *Texas Instruments Incorporated Application Note.* **SLOA049A**

[56] **MICROCHIP TECHNOLOGY INCORPORATED.** (2003). MCP6021/2/3/4 rail-to-rail input/output, 10 MHz op amp. *Microchip Technology Incorporated Data Sheet.* **DS21685B**

[57] **MICROCHIP TECHNOLOGY INCORPORATED.** (2003a). dsPICDEM™ 1.1 development board user's guide. *Microchip Technology Incorporated User's Guide.* **DS70099B**

[58] **MICROCHIP TECHNOLOGY INCORPORATED.** (2003b). MPLAB® ICD2 in-circuit debugger user's guide. *Microchip Technology Incorporated User's Guide.* **DS51331A**

[59] **FAIRBANKS, J.** (2002). LM2623 ratio adaptive gated oscillator cookbook. *National Semiconductor Corporation Application Note.* **AN1221**

[60] **MANCINI, R.** (2002). Op amps for everyone. *Texas Instruments Incorporated Design Reference.* **SLOD006B**

[61] **JENSEN, C.** (1999). Layout guidelines for switching power supplies. *National Semiconductor Corporation Application Note.* **AN1149**

[62] **MICROCHIP TECHNOLOGY INCORPORATED.** (2003c). MPLAB® C30 C compiler user's guide. *Microchip Technology Incorporated User's Guide.* **DS51284B**

[63] **MICROCHIP TECHNOLOGY INCORPORATED.** (2003). dsPIC30F family reference manual, high performance digital signal controllers. *Microchip Technology Incorporated Reference Manual.* **DS70046B**

[64] **SINHA, P.** (2005). MPLAB-C30 dsPIC30F6014A processor header. *Microchip Technology Incorporated Program.* Rev. 4.3

[65] **STOWE, G.** (2006). Dataflash® integrity testing SPI code.

[66] **ATMEL CORPORATION.** (2005). AVR335: Digital sound recorder with AVR® and DataFlash®. *Atmel Corporation Application Note.* Rev. 1456C-AVR-04/05

[67] **MICROCHIP TECHNOLOGY INCORPORATED.** (2004). MCP601/2/3/4 2.7V to 5.5V Single -Supply CMOS Op Amps. *Microchip Technology Incorporated Data Sheet.* **DS21314F**

[68] **D'SOUZA, S.** (2003). dsPICDEM™ low cost starter board demonstration code. *Microchip Technology Incorporated Program.* **ADC12_UART.c**; Rev. 1.0.

[69] **VASUKI, H.** (2005). CE001_ADC_DSP_lib_FILTER example source code. *Microchip Technology Incorporated Program.* **ADC.c**; Rev. 2.0.

# Appendix A

## Rolling Element Bearing Fundamental Frequencies

The vibrational motion of a rotating rolling element bearing can be expressed in terms of the fundamental frequencies of the bearing parts. It is assumed that the outer race is stationary and the inner race rotates at the shaft frequency $\omega_s$.

- ***Fundamental Train Frequency*** **(FTF)** – The rate of rotation of the bearing cage.

$$\omega_c = \frac{\omega_s}{2}\left(1 - \frac{d}{D}\cos\alpha\right) \quad \text{[rad/s]}$$

- ***Ball Spin Frequency*** **(BSF)** – The rate at which a point on a rolling element makes contact with the races.

$$\omega_b = \frac{D\omega_s}{2d}\left(1 - \frac{d^2}{D^2}\cos^2\alpha\right) \quad \text{[rad/s]}$$

- ***Ball Pass Frequency Outer*** **(BPFO)** – The rate at which the rolling elements pass over a fixed point on the outer race.

$$\omega_{od} = Z\omega_c \quad \text{[rad/s]}$$

- ***Ball Pass Frequency Inner*** **(BPFI)** – The rate at which the rolling elements pass over a fixed point on the inner race.

$$\omega_{id} = Z(\omega_s - \omega_c) \quad \text{[rad/s]}$$

where $\omega_s$ is the shaft rotation frequency in rad/s, $d$ is the rolling element diameter, $D$ is the distance between centres of two rolling elements on opposite sides of the bearing also known as the pitch diameter, $Z$ is the number of rolling elements and $\alpha$ is the contact angle.

# Appendix B

## Selected Design Specifications for the Savmore Overland Belt Conveyor CV-08 (Lodewijks, G., 2003) [36].

***Project Details:***

| | | |
|---|---|---|
| 1. Project | - | Maquesa Shafts Overland |
| 2. Owner | - | Savmore Colliery |
| 3. Conveyor | - | Overland Belt Conveyor System CV-08 |
| 4. Length | - | 6541 m |
| 5. Elevation | - | -60 m |

***Material Specification:***

| | | |
|---|---|---|
| 1. Material Conveyed | - | ROM Coal |
| 2. Design Tonnage | - | 1000 tons/hour |
| 3. Bulk Density | - | 850 $kg.m^{-3}$ |
| 4. Maximum Lump Size | - | 100 mm |
| 5. Surcharge Angle | - | 20° |

***Belt Specification:***

| | | |
|---|---|---|
| 1. Manufacturer | - | Goodyear |
| 2. Width | - | 900 mm |
| 3. Strength | - | 1000 N/mm |
| 4. Design Velocity | - | 5.75 m/s |
| 5. Belt Type | - | Steel Cord |
| 6. Class | - | ST 1000 |
| 7. Cover Thickness (Top) | - | 6.0 mm |
| 8. Cover Thickness (Bottom) | - | 5.0 mm |
| 9. Weight | - | 19.0 kg/m |
| 10. Tape Length (approx) | - | 13125 m |
| 11. Belt Edge Clearance to Ore | - | 137 mm |

***Idler Specification:***

1. Type:
   a. Carry - 3 Equal Width Rolls, 45° trough angle
   b. Return - 2 Equal Width Rolls, 10° trough angle
2. Roll Dimensions
   a. Carry - 152 mm diameter x 350 mm face
   b. Return - 152 mm diameter x 350 mm face
3. Applied Load
   a. Carry - 3050 N
   b. Return - 1855 N
4. Spacing
   a. Carry (straight) - 4.5 m
   b. Carry (convex/concave) - 4.5 m
   c. Carry (horizontal curve) - 2.25 m
   d. Return (straight) - 9.0 m
   e. Return (convex/concave) - 9.0 m
   f. Return (horizontal curve) - 4.5 m
5. Banking Angles
   a. Carry - 3.4 m
   b. Return - 9.0 m
6. Quantity (approx)
   a. Carry - 1458
   b. Return - 730

***Pulley Specifications:***

1. Drive Pulley
   a. Diameter - 800 mm
   b. Face Width - 1150 mm
   c. Lagging (ceramic) - 12 mm
   d. Shaft Diameter - 240 mm
   e. Bearing Diameter - 200 mm
   f. Bearing Centres - 1550 mm

2. High Tension Pulley
    a. Diameter - 800 mm
    b. Face Width - 1150 mm
    c. Lagging (ceramic) - 12 mm
    d. Shaft Diameter - 240 mm
    e. Bearing Diameter - 200 mm
    f. Bearing Centres - 1550 mm
3. Low Tension Pulley
    a. Diameter - 630 mm
    b. Face Width - 1150 mm
    c. Lagging (ceramic) - 0 mm
    d. Shaft Diameter - 140 mm
    e. Bearing Diameter - 115 mm
    f. Bearing Centres - 1550 mm
4. Turnover Pulley
    a. Diameter - 150 mm
    b. Face Width - 1150 mm
    c. Lagging (ceramic) - 0 mm
    d. Shaft Diameter - 75 mm
    e. Bearing Diameter - 75 mm
    f. Bearing Centres - 1550 mm

***Drive System Specification***

1. Motor Types - VSD/VFD
2. Nameplate Rating of Motors - 375 kW
3. Motor Quantity Primary Pulley - 1
4. Motor Quantity Secondary Pulley- 1
5. Total Installed Power - 750 kW
6. Motor Synchronous Speed - 1500 RPM

# Appendix C

## Troubleshooting Manual for Problems Encountered in Overland Conveyor Belts (Taken from Goodyear Industrial Products) [44].

| PROBLEM | CAUSE *in order of probable* | | | | | |
|---|---|---|---|---|---|---|
| Conveyor runs to one side at given point on structure | 5 | 4 | 1 | 2 | 3 | 44 |
| Particular section of belt runs to one side at all points on conveyor | 6 | 7 | - | - | - | - |
| Belt runs to one side for long distance or entire length of conveyor | 39 | 8 | 5 | 1 | 2 | 3 |
| Belt runs off at tail pulley | 39 | 10 | 1 | - | - | - |
| Belt runs off head pulley | 33 | 10 | 1 | 3 | - | - |
| Belt slip | 34 | 33 | 31 | 10 | 4 | - |
| Belt slip on starting | 34 | 31 | 33 | - | - | - |
| Excessive belt stretch | 41 | 42 | 43 | 12 | 32 | 35 |
| Grooving, gouging or stripping of top cover | 13 | 14 | 15 | 16 | - | - |
| Excessive top cover wear, uniform around belt | 19 | 20 | 10 | 8 | 36 | - |
| Severe pulley cover wear | 4 | 9 | 10 | 17 | 11 | 27 |
| Longitudinal grooving or cracking of bottom cover | 4 | 10 | 9 | 33 | - | - |
| Covers harden or crack | 23 | 37 | - | - | - | - |
| Cover swells in spots or streaks | 21 | - | - | - | - | - |
| Belt breaks at or behind fasteners; fasteners pull out | 24 | 22 | 12 | 23 | - | - |
| Vulcanized splice separation | 38 | 30 | 12 | 17 | 25 | - |
| Excessive edge wear, broken edges | 8 | 10 | 40 | 7 | - | - |
| Transverse breaks at belt edge | 18 | 25 | 26 | - | - | - |
| Short breaks in carcass parallel to belt edge, star breaks in carcass | 16 | 17 | - | - | - | - |
| Ply separation | 29 | 30 | 23 | - | - | - |
| Carcass fatigue at idler junction | 25 | 26 | 27 | 28 | 29 | 36 |
| Cover blisters or sand blisters | 45 | 21 | - | - | - | - |
| Belt cupping – new belt | 46 | - | - | - | - | - |
| Belt cupping – old belt (was OK when new) | 21 | 23 | - | - | - | - |

**1. Idlers or pulleys out-of-square with center line of belt:**
Readjust idlers in affected area.
**2. Conveyor frame or structure crooked:**
Straighten in affected area.
**3. Idler stands not centered on belt:**
Readjust idlers in affected area.
**4. Sticking idlers:**
Free idlers and improve maintenance and lubrication.
**5. Buildup of material on idlers:**
Remove accumulation; improve maintenance, install scrapers or other cleaning devices.
**6. Belt not joined squarely:**
Remove affected splice and re-splice.
**7. Bowed belt:**
For new belt this condition should disappear during break-in; in rare instances belt must be straightened or replaced; check storage and handling of belt rolls.
**8. Off-center loading or poor loading:**
Adjust chute to place load on center of belt; discharge material in direction of belt travel at or near belt speed.
**9. Slippage on drive pulley:**
Increase tension thru screw take up or add counterweight; lag drive pulley; increase arc of contact.
**10. Material spillage and buildup:**
Improve loading and transfer conditions; install cleaning devices; improve maintenance.
**11. Bolt heads protruding above lagging:**
Tighten bolts; replace lagging; use vulcanized-on lagging.
**12. Tension too high:**
Increase speed, same tonnage; reduce tonnage, same speed; reduce friction with better maintenance and replacement of damaged idlers; decrease tension by increasing arc of contact Or go to lagged pulley; reduce CWT to minimum amount.
**13. Skirt boards improperly adjusted or of wrong material:**
Adjust skirt board supports to minimum 1" between metal and belt with gap increasing in direction of belt travel; use skirt board rubber (not old belt).
**14. Belt spanking down under load impact:**
Install cushion idlers.
**15. Material hanging up in or under chute:**
Improve loading to reduce spillage; install baffles; widen chute.
**16. Impact of material on belt:**
Reduce impact by improving chute design; install impact idlers.
**17. Material trapped between belt and pulley:**
Install plows or scrapers on return run ahead of tail pulley.
**18. Belt edges folding up on structure:**
Same corrections as for 1, 2, 3; install limit switches; provide more clearance.
**19. Dirty, stuck, or misaligned return rolls:**
Remove accumulations; install cleaning devices, use self cleaning
Return rolls, improve maintenance and lubrication.
**20. Cover quality too low:**
Replace with belt of heavier cover gauge or higher quality rubber.
**21. Spilled oil or grease, over-lubrication of idlers:**
Improve housekeeping; reduce quantity of grease used; check grease seals.
**22. Wrong type of fastener, fasteners too tight or too loose:**
Use proper fasteners and splice technique; set up schedule for regular fastener inspection.
**23. Heat or chemical damage:**
Use belt designed for specific condition.
**24. Fastener plates too long for pulley size:**
Replace with smaller fasteners; increase pulley size.
**25. Improper transition between troughed belt and terminal pulleys:**
Adjust transition in accordance with Goodyear Handbook of Belting [44]
**26. Severe convex (hump) vertical curve:**
Decrease idler spacing in curve; increase curve radius; consult Goodyear Handbook of Belting [44] for assistance.
**27. Excessive forward tilt of trough rolls:**
Reduce forward tilt of idlers to no more than 2° from vertical.
**28. Excess gap between idler rolls:**
Replace idlers; replace with heavier belt.
**29. Insufficient transverse stiffness:**
Replace with the proper belt.
**30. Pulleys too small:**
Use larger diameter pulleys.
**31. Counterweight too light:**
Add counterweight or increase screw take-up tension to value determined from calculations.
**32. Counterweight too heavy:**
Lighten counterweight to value required by calculations.
**33. Pulley lagging worn:**
Replace pulley lagging.
**34. Insufficient traction between belt and pulley:**
Lag drive pulley; increase belt wrap; install belt cleaning devices.
**35. System under belted:**
Recalculate belt tensions and select proper belt.
**36. Excessive sag between idlers causing load to work and shuffle on belt as it passes over idlers:**
Increase tension if unnecessarily low; reduce idler spacing.
**37. Improper storage or handling:**
Refer to Goodyear [44] for proper storage and handling instructions.
**38. Belt improperly spliced:**
Re-splice using proper method as recommended by Goodyear [44].
**39. Belt running off-center around the tail pulley and through the loading area:**
Install training idlers on the return run prior to tail pulley.
**40. Belt hitting structure:**
Install training idlers on carrying and return run.
**41. Improper belt installation causing apparent excessive belt stretch:**
Pull belt through counterweight with a tension equal to at least
Empty running tension; run belt in with mechanical fasteners.
**42. Improper initial positioning of counterweight in its carriage causing apparent excessive belt stretch:**
Check Goodyear Handbook of Belting [44] for recommended initial position.
**43. Insufficient counterweight travel:**
Consult Goodyear Conveyor and Elevator Belt Selection Manual for recommended minimum distances.
**44. Structure not level:**
Level structure in affected area.
**45. Cover cuts or very small cover away from carcass:**
Make spot repair with vulcanizer or self-curing repair material.
**46. Excessive cover gauge ratio:**
Use a belt with a lower gauge ratio and/or a thicker carcass.

# Appendix D

## External Memory Calculations for the Vibration Data Logging Board

An Atmel AT45DB321C 32-Mbit Dataflash® Chip was selected as the memory storage device on the vibration data logging board and this size of storage capacity was selected due to the following calculations:


**Figure D: Savmore Overland Belt Conveyor CV-08**

The parameters from the Savmore Overland Conveyor Belt CV-08 were used to calculate the size of the memory storage device.

The speed of the belt is assumed to be 5.75 m/s, the distance between idlers is 4.5m and the diameter of an idler is 152 mm. The maximum required sampling rate of any of the sensors is 16 kHz.

The RMS calculation is a form of averaging and is taken over 8 samples of data as this is half the size of the ADC Buffer. It gives and indication of the magnitude of the acceleration and therefore vibration of the system. It is important to take a large number of samples over the idlers as these are of interest. The maximum number of samples per metre is:

$$\begin{aligned} Number\ of\ Samples/Metre &= \frac{Samples/Second}{BeltSpeed} \\ &= \frac{16000 samples/s}{5.75 m/s} \\ &= 2782.601\ samples/m \\ &\approx 2783\ samples/m \end{aligned}$$

Therefore the number of samples per idler is:

$$\begin{aligned} Number\ of\ Samples/Idler &= (samples/m)\times(Idler\phi) \\ &= 2783\ samples\cdot m^{-1}\times 0.152\ m \\ &= 422.96 samples/idler \end{aligned}$$

There will be a maximum number of 256 kbits produced every second for a sampling rate of 16 kHz as every sample is 16 bits long.

$$\begin{aligned} Number\ of\ bits/s &= (samples/s)\times(word\ size) \\ &= 16000kHz\times 16bits \\ &= 256kbits/s \end{aligned}$$

As there are 8 samples per RMS measurement this number is reduced to 32 kbits/second.

$$\begin{aligned} RMSRate &= \frac{Number\ of\ bits/s}{N} = \frac{256\ kbits}{8} \\ &= 32\ kbits/s \end{aligned}$$

This means that a 32Mbit memory chip will store approximately 16minutes and 40 seconds of data.

$$\begin{aligned} StorageTime &= \frac{MemoryCapacity}{RMSRate} = \frac{32Mbits}{32kbits/s} \\ &= 1000s \\ &= 16\min 40s \end{aligned}$$

# Appendix E

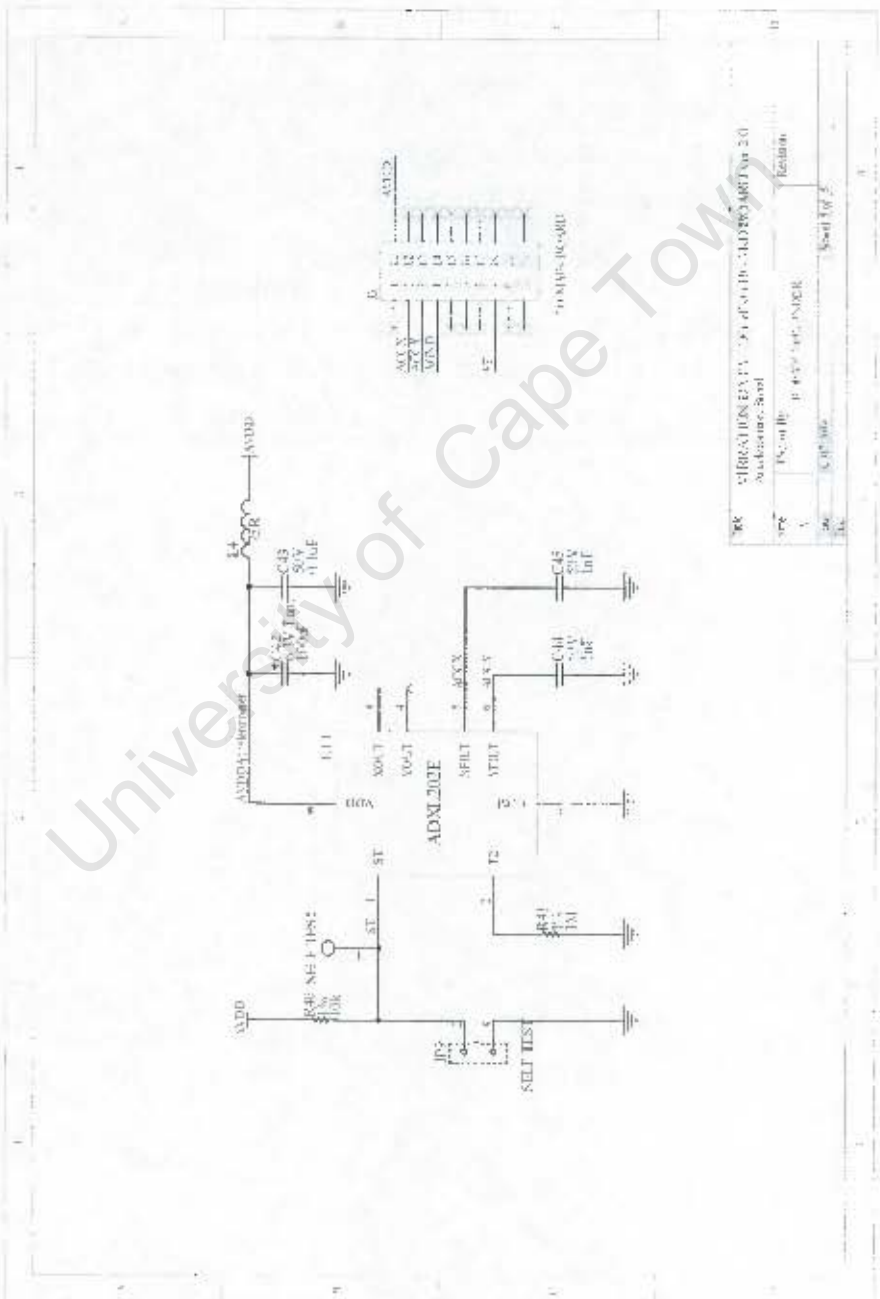
## Bessel Filter Table (Taken from Karki, J., 2000) [55]

| FILTER ORDER | STAGE 1 | | STAGE 2 | | STAGE 3 | | STAGE 4 | | STAGE 5 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | FSF | Q | FSF | Q | FSF | Q | FSF | Q | FSF | Q |
| 2 | 1.2736 | 0.5773 | | | | | | | | |
| 3 | 1.4524 | 0.6910 | 1.3270 | | | | | | | |
| 4 | 1.4192 | 0.5219 | 1.5912 | 0.8055 | | | | | | |
| 5 | 1.5611 | 0.5635 | 1.7607 | 0.9165 | 1.5069 | | | | | |
| 6 | 1.6060 | 0.5103 | 1.6913 | 0.6112 | 1.9071 | 1.0234 | | | | |
| 7 | 1.7174 | 0.5324 | 1.8235 | 0.6608 | 2.0507 | 1.1262 | 1.6853 | | | |
| 8 | 1.7837 | 0.5060 | 2.1953 | 1.2258 | 1.9591 | 0.7109 | 1.8376 | 0.5596 | | |
| 9 | 1.8794 | 0.5197 | 1.9488 | 0.5894 | 2.0815 | 0.7606 | 2.3235 | 1.3220 | 1.8575 | |
| 10 | 1.9490 | 0.5040 | 1.9870 | 0.5380 | 2.0680 | 0.6200 | 2.2110 | 0.8100 | 2.4850 | 1.4150 |

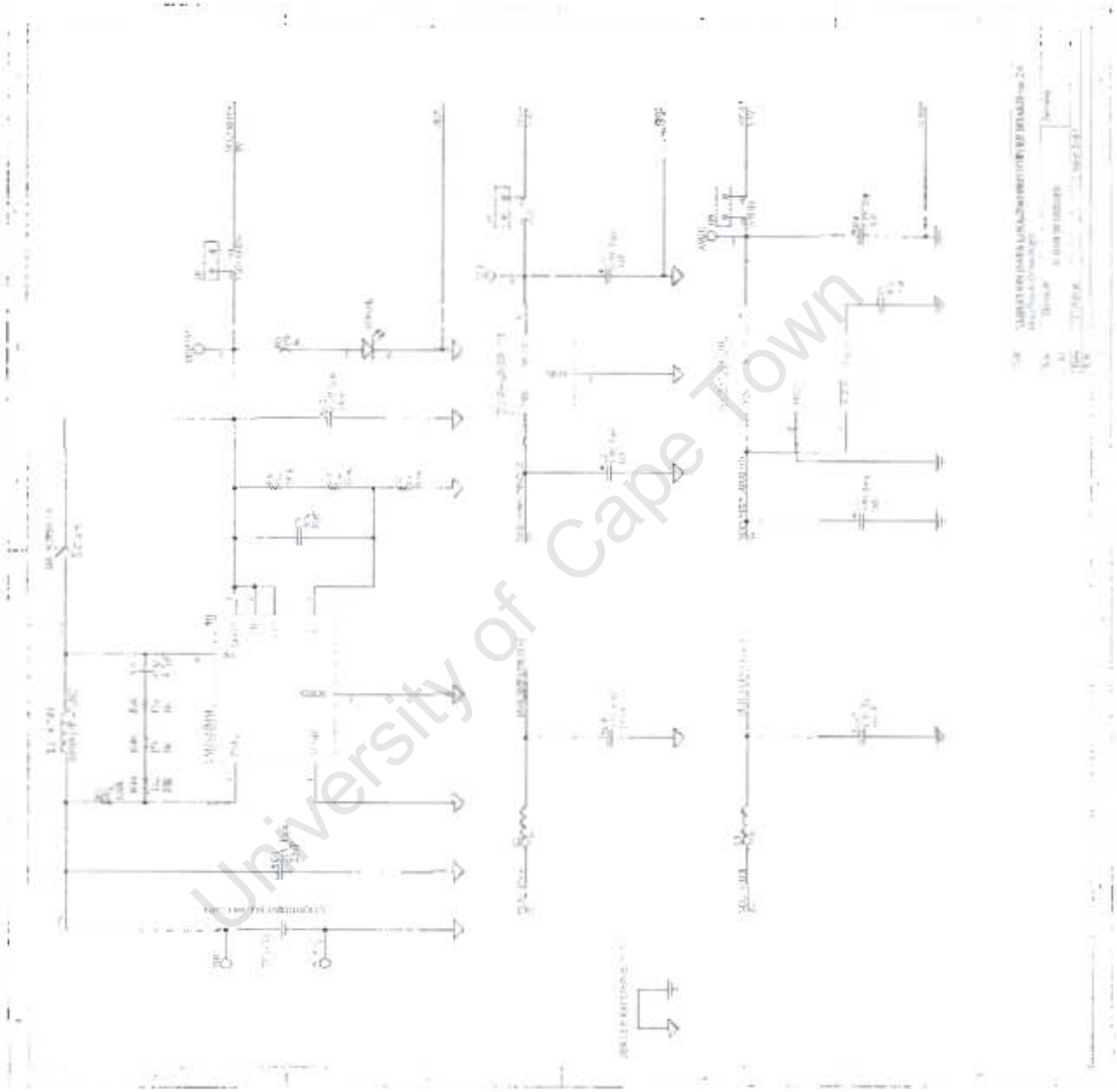
# Appendix F

## Circuit Diagrams for the Vibration Data Logging Prototype Board

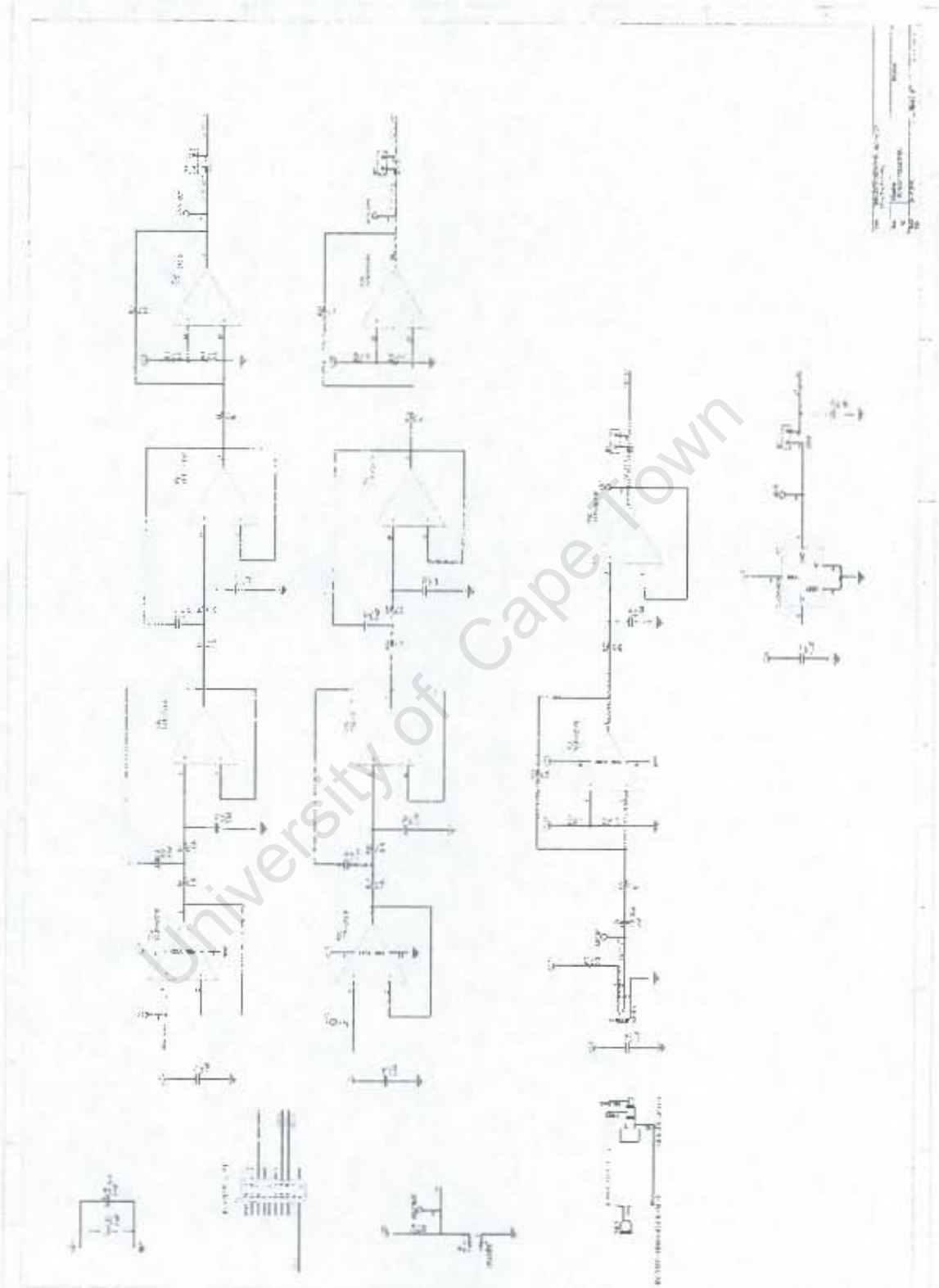
*Accelerometer Board*



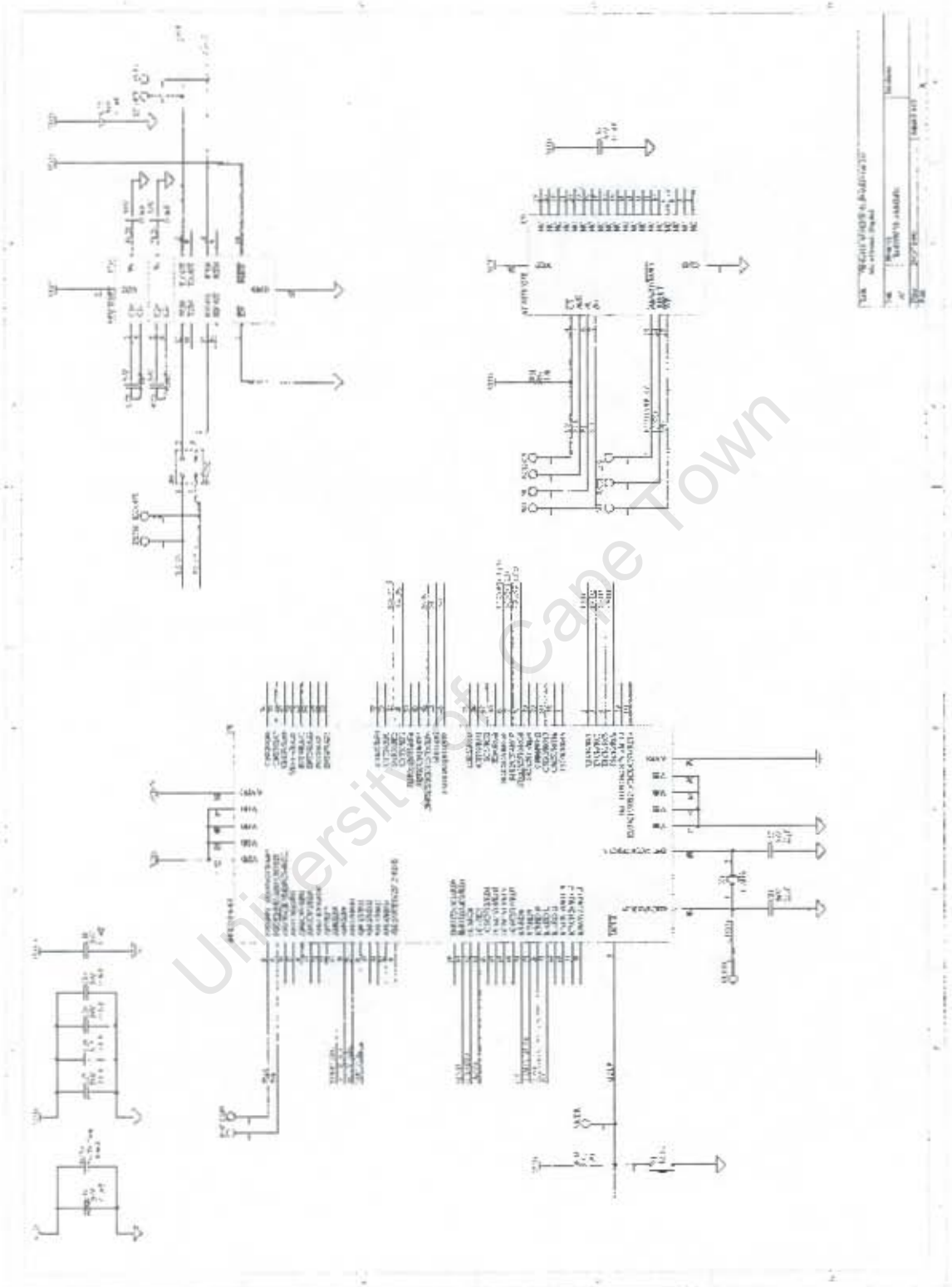
*Main Board – Power Supply*



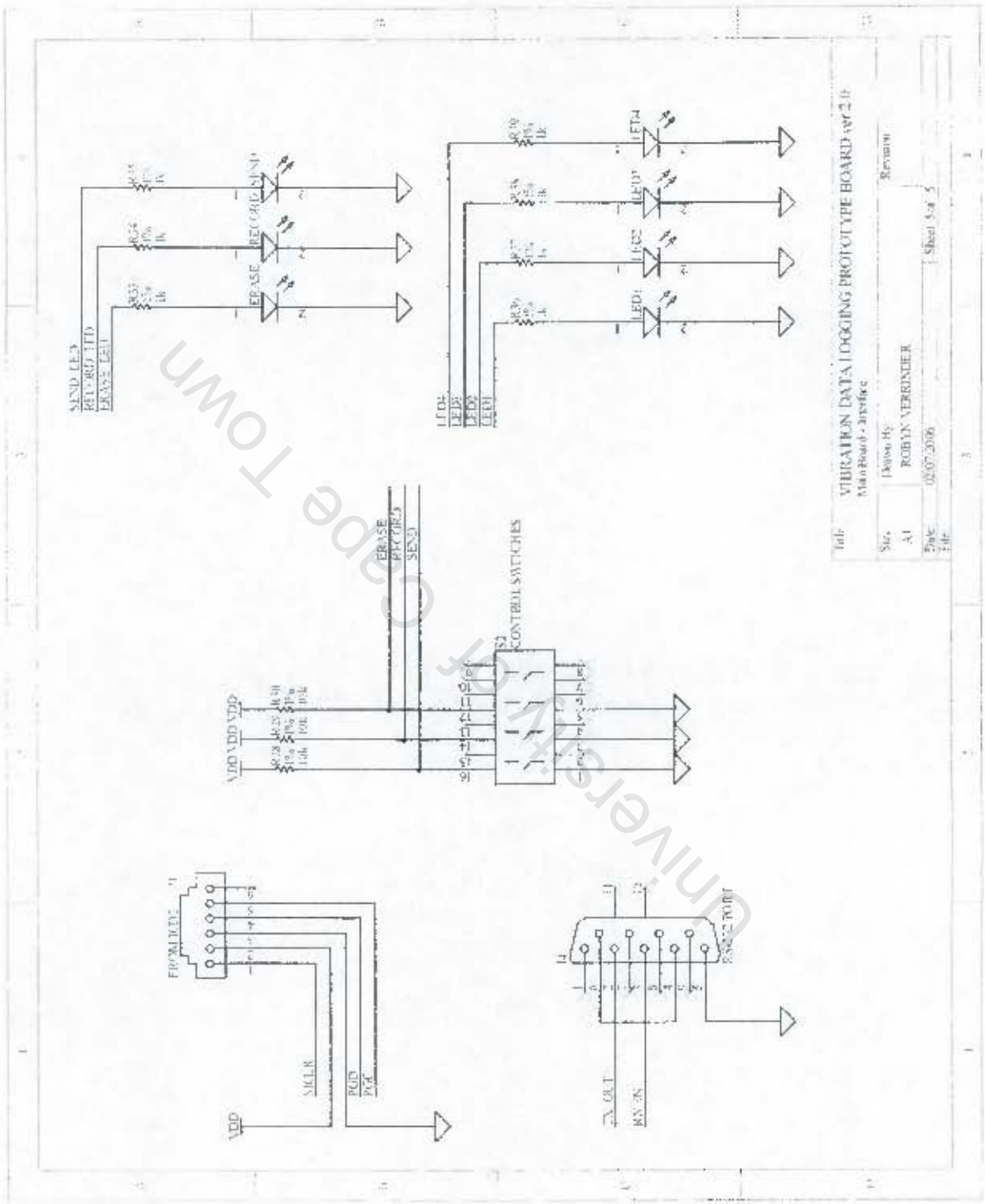
*Main Board – Analogue Circuits*



## *Main Board – Digital Circuits*



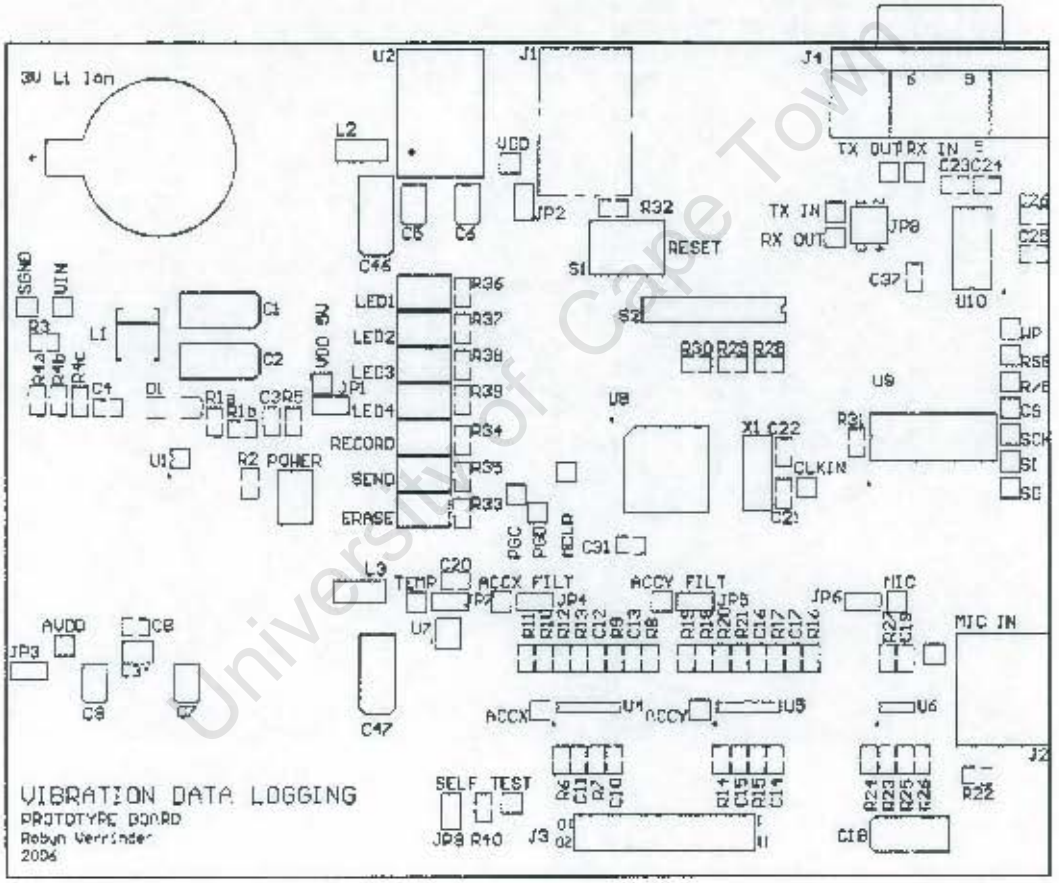
*Main Board – Interfacing Circuits*



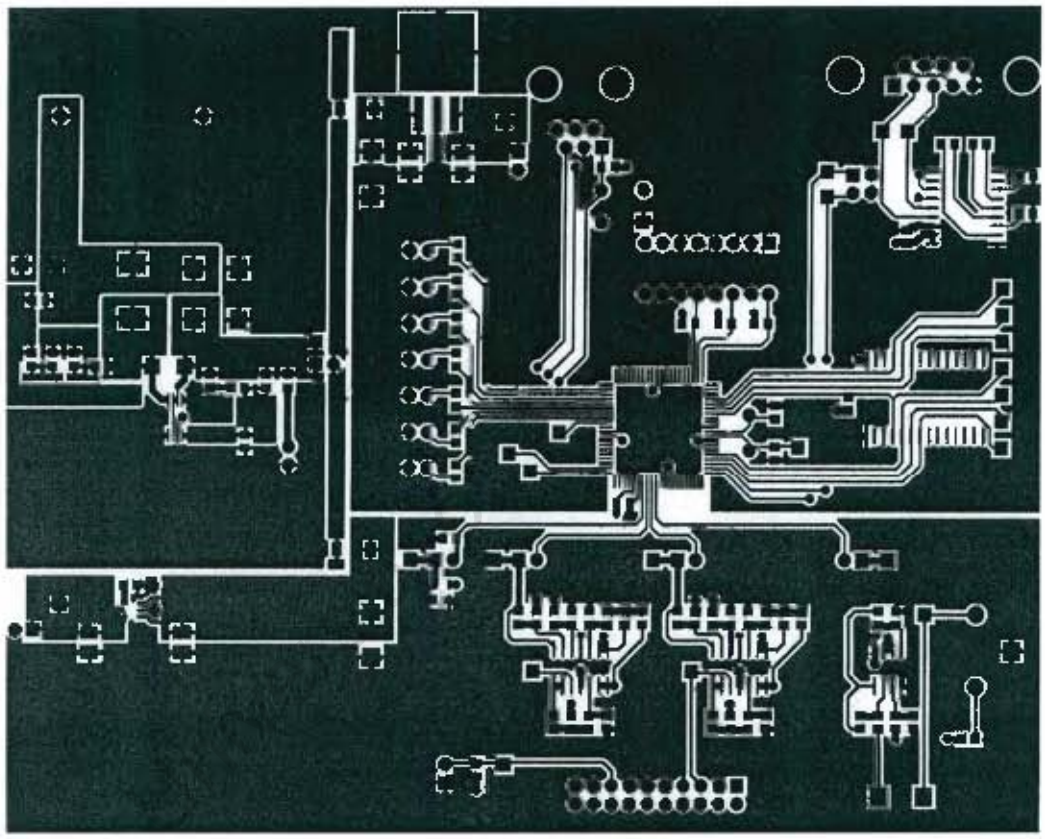
# Appendix G

## Main Vibration Data Logging Prototype Printed Circuit Board Layout

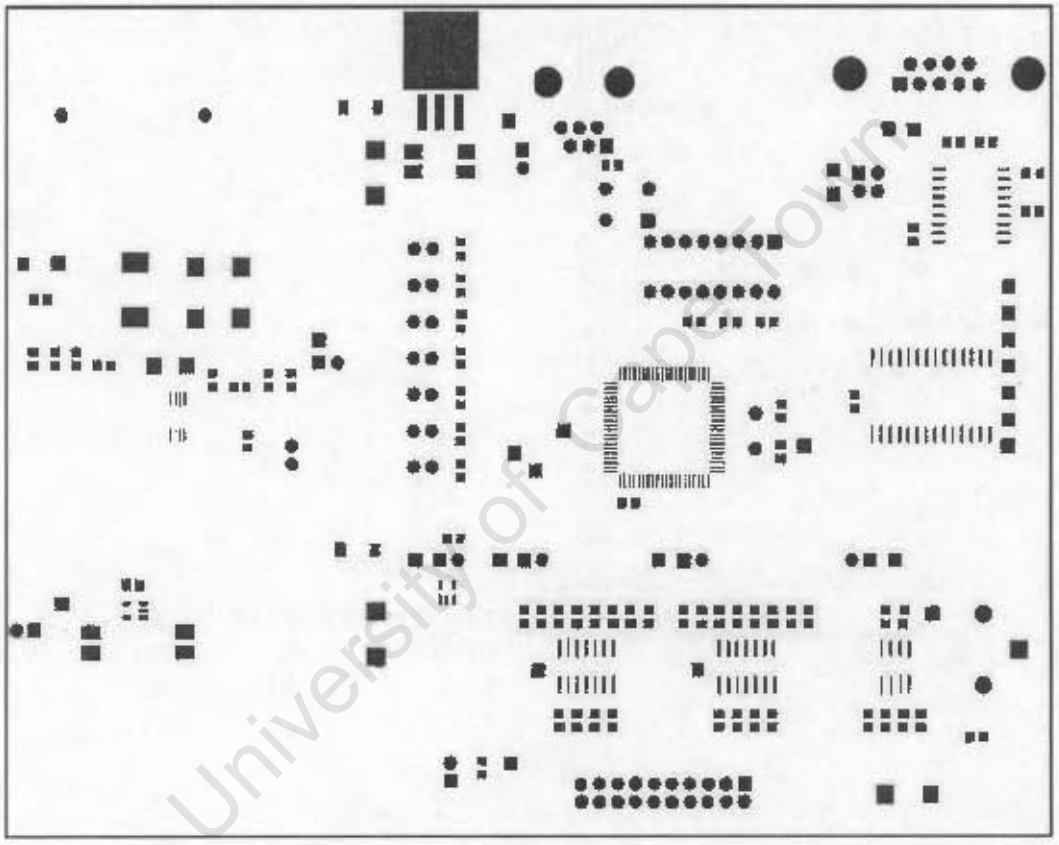
*Top Layer Silkscreen*



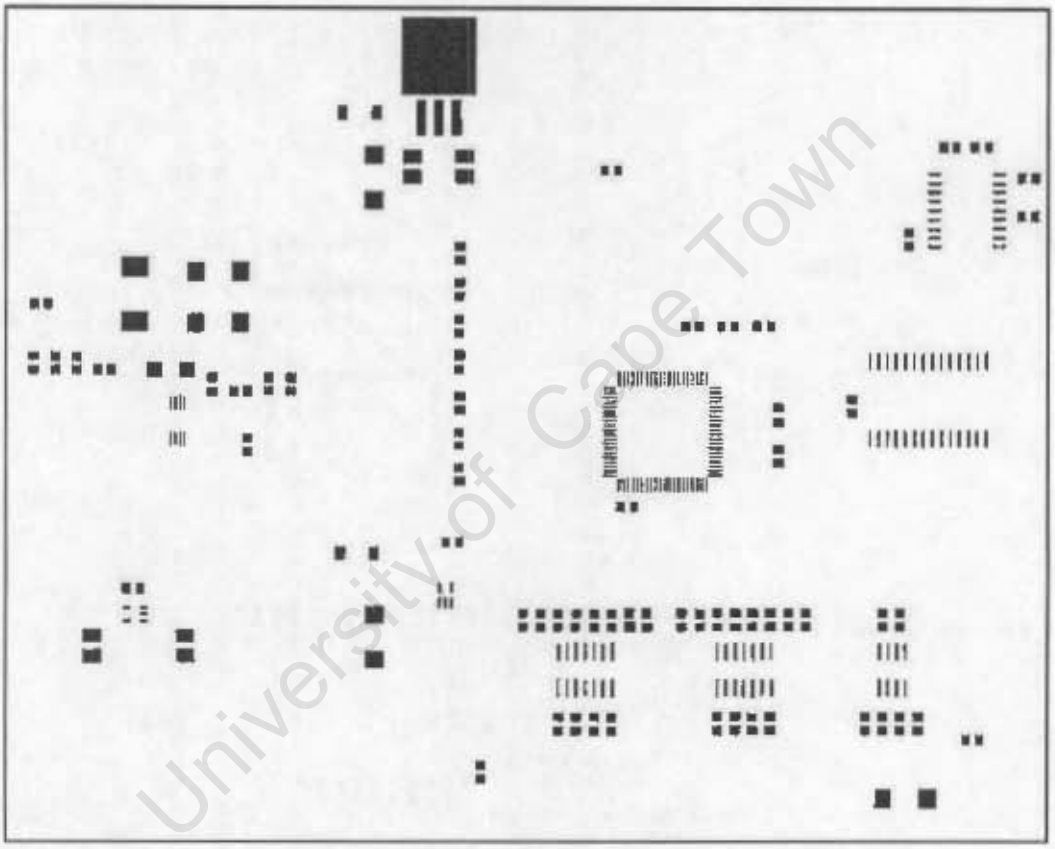
*Top Layer*



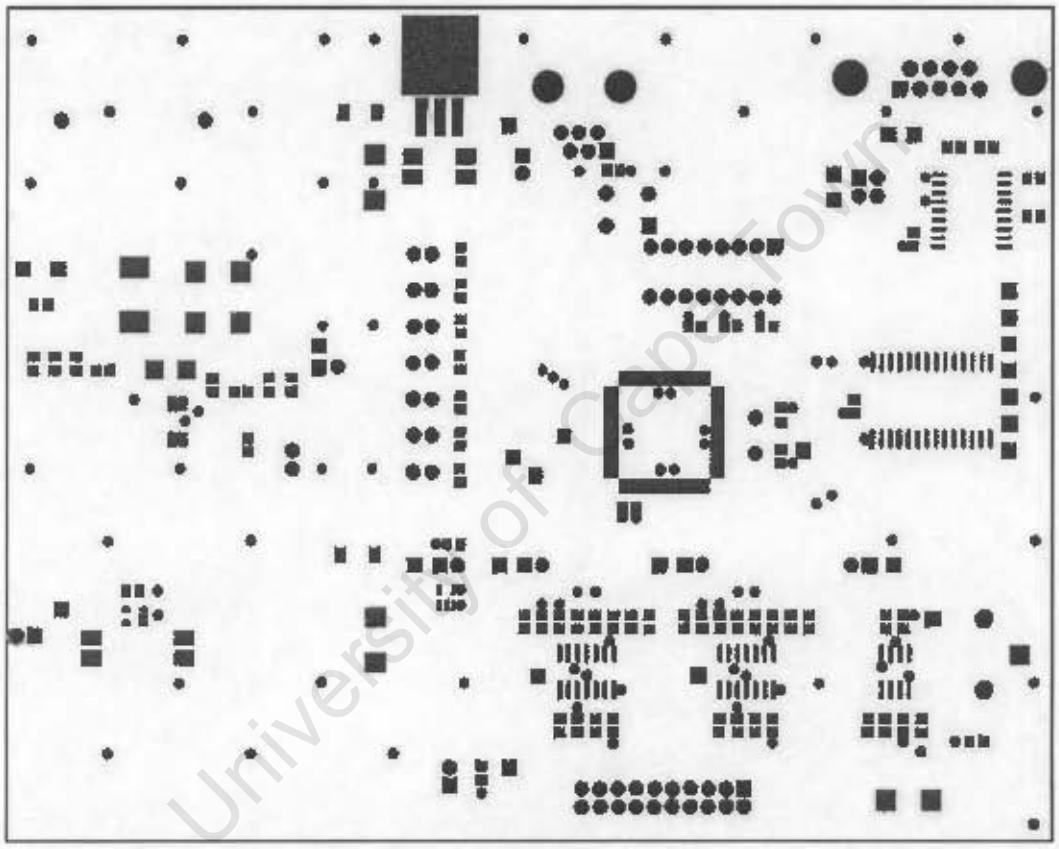
*Top Layer Pad Master*



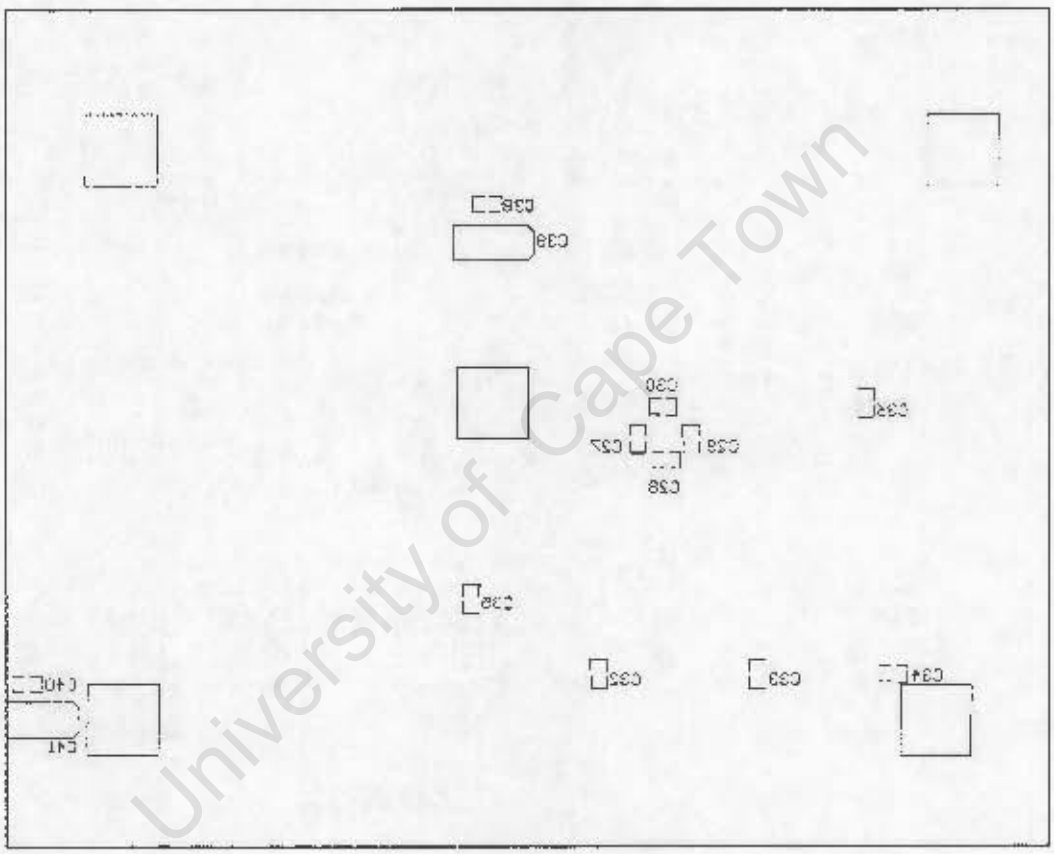
*Top Layer Paste Mask*



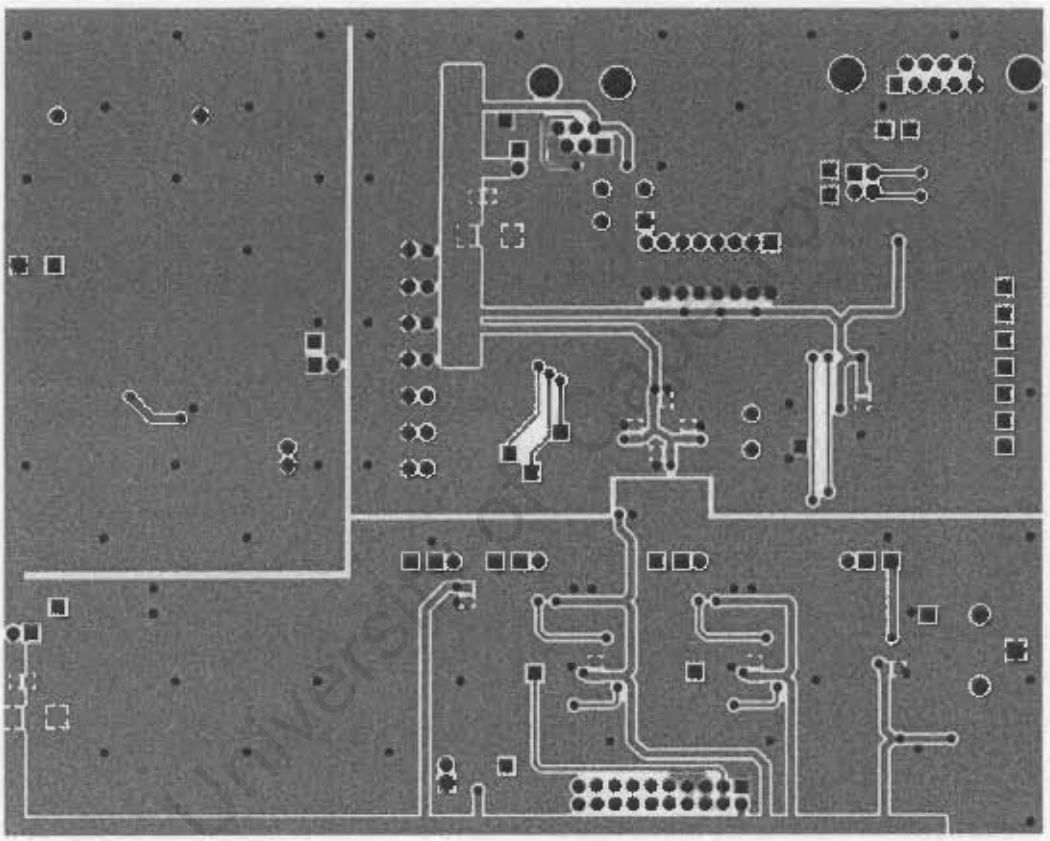
*Top Layer Solder Mask*



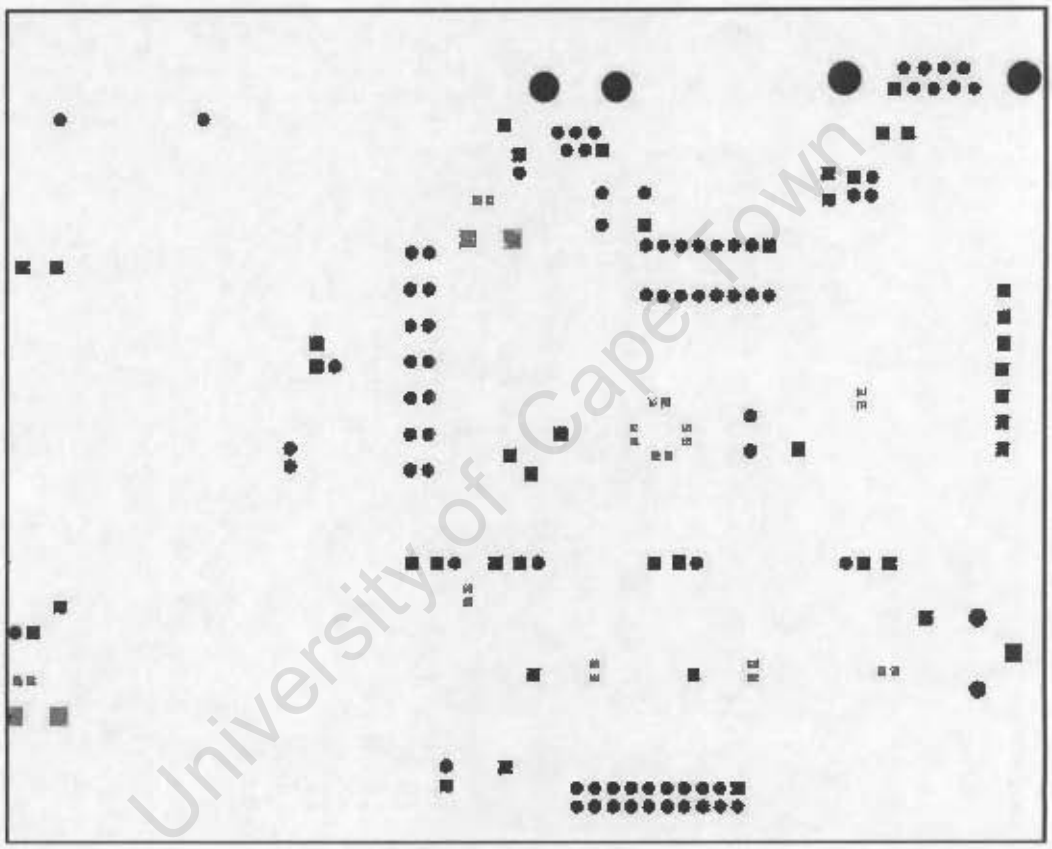
*Bottom Layer Silkscreen*



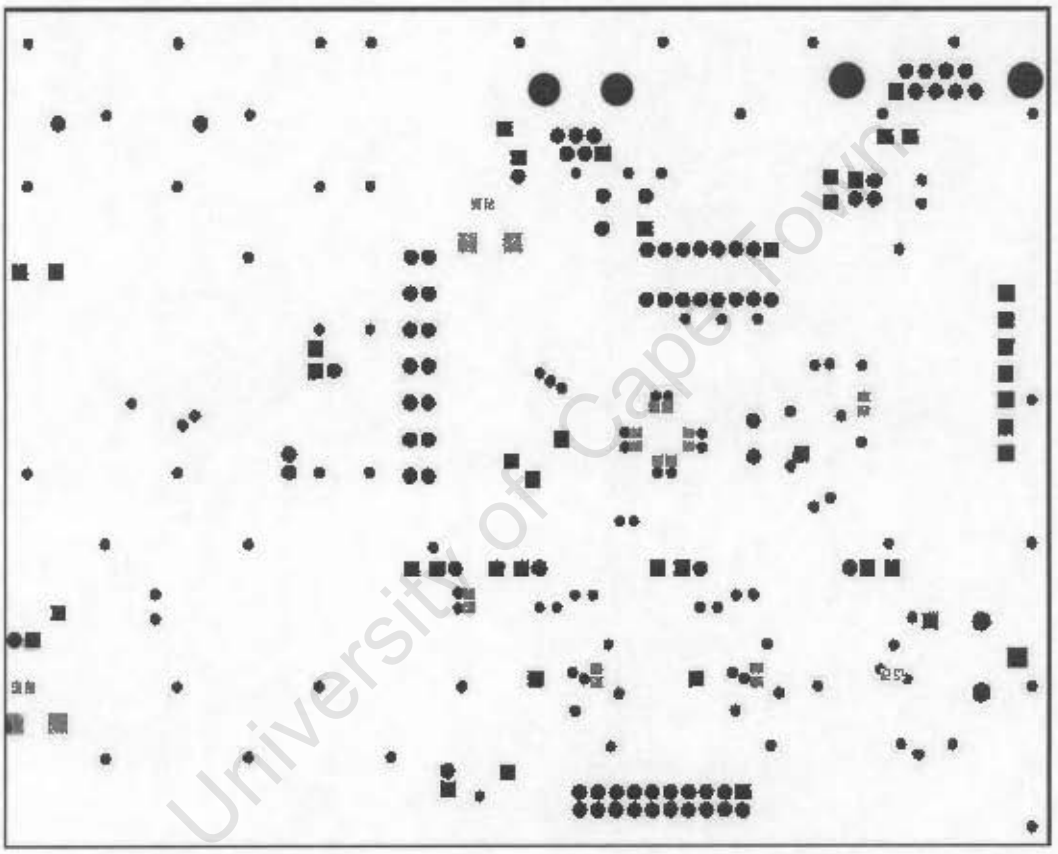
*Bottom Layer*



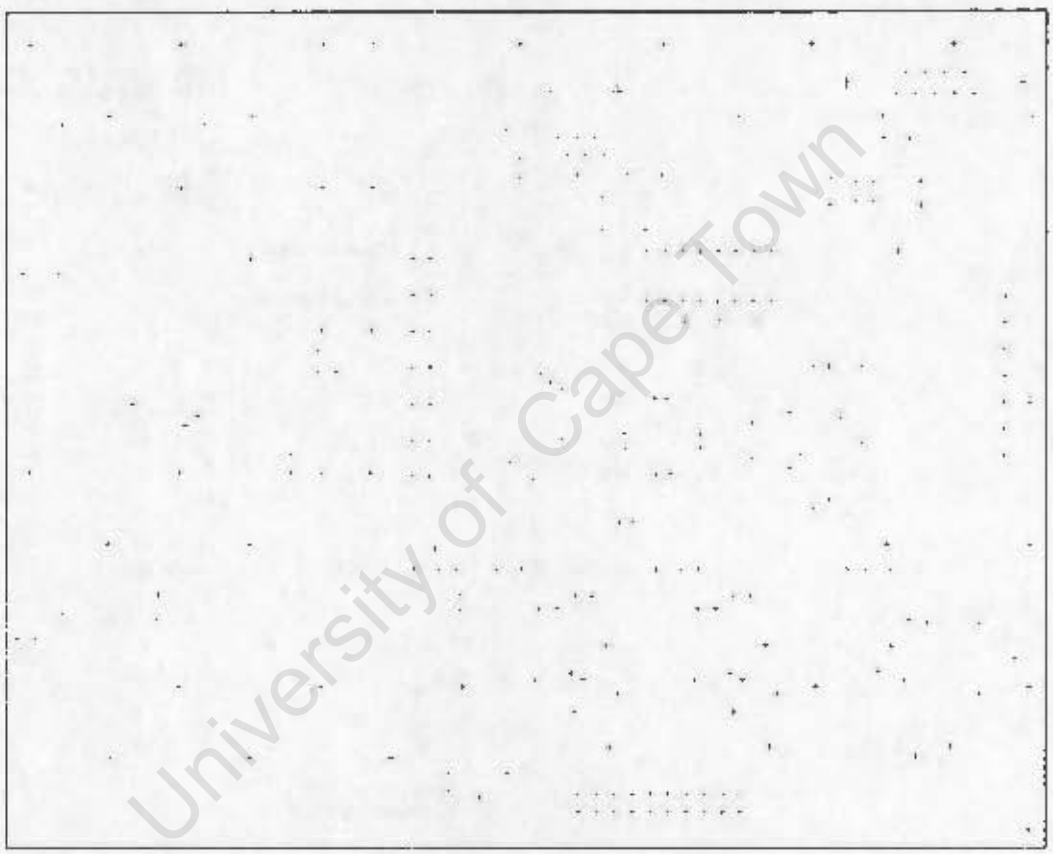
*Bottom Layer Pad Master*



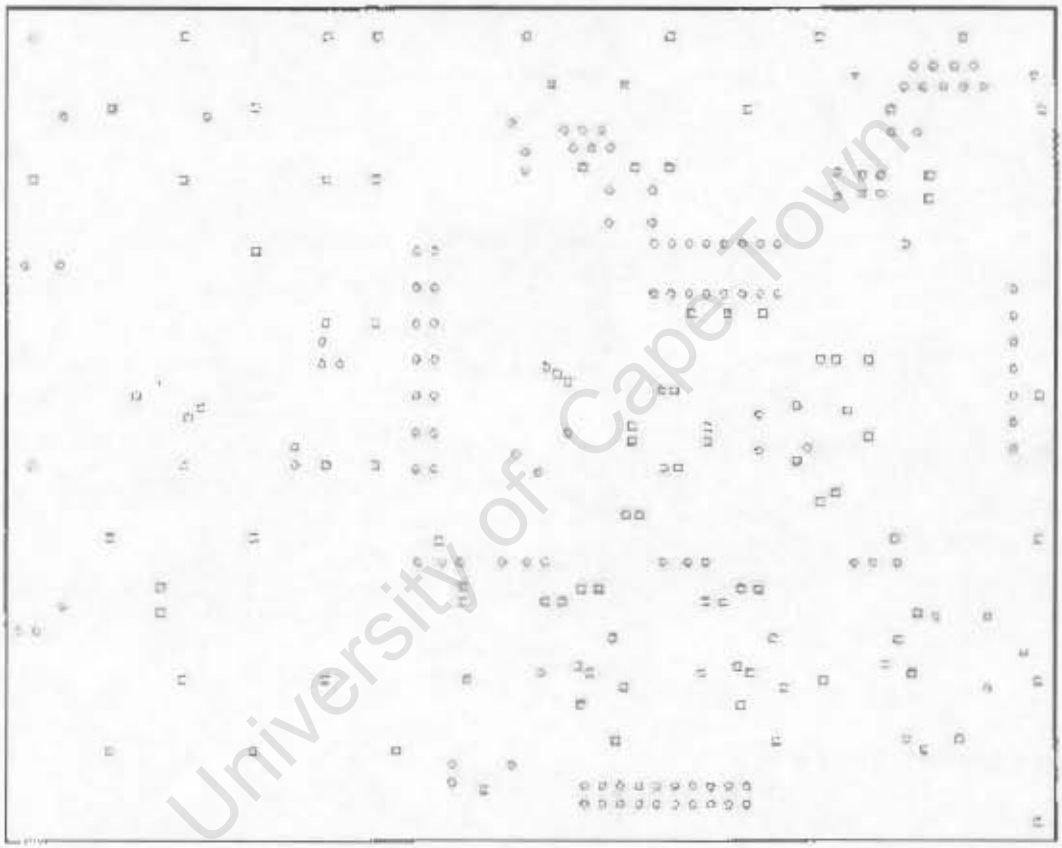
*Bottom Layer Solder Mask*



*Drill Guide for the Top and Bottom Layers*



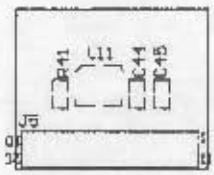
*Drill Drawing for Top and Bottom Layers*



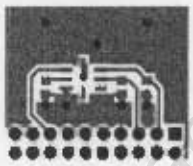
# Appendix H

## Accelerometer Printed Circuit Board Layout

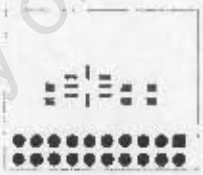
*Top Layer Silkscreen*



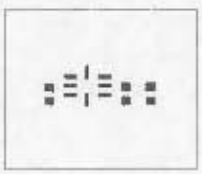
*Top Layer*



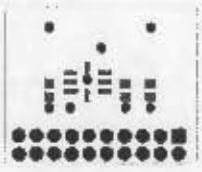
*Top Layer Pad Master*



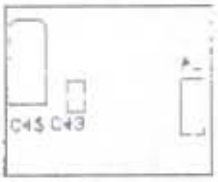
*Top Layer Paste Mask*



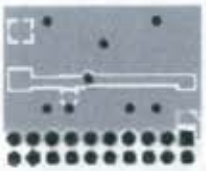
*Top Layer Solder Mask*



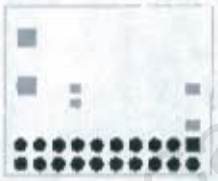
*Bottom Layer Silkscreen Overlay*



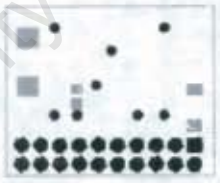

*Bottom Layer*



*Bottom Layer Pad Master*



*Bottom Layer Solder Mask*



*Drill Guide for the Top and Bottom Layers*

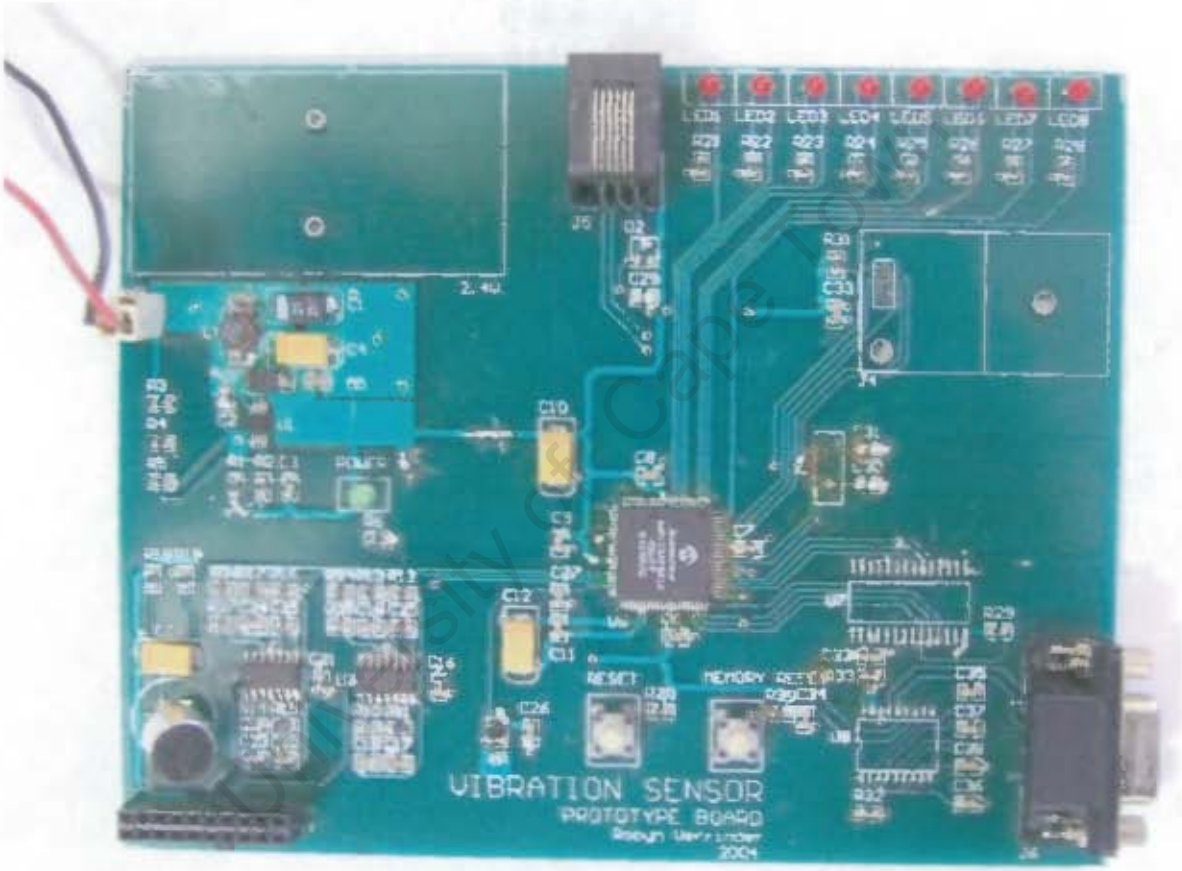


*Drill Drawings for the Top and Bottom Layers*

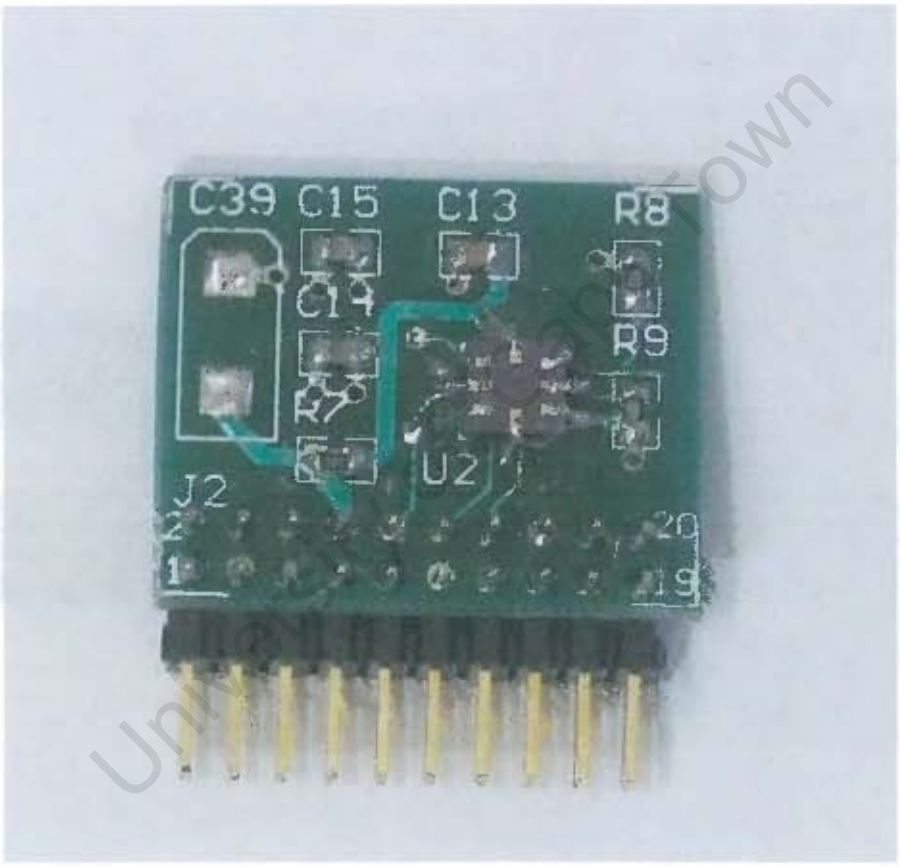
# Appendix I

## Photographs of the Printed Circuit Board

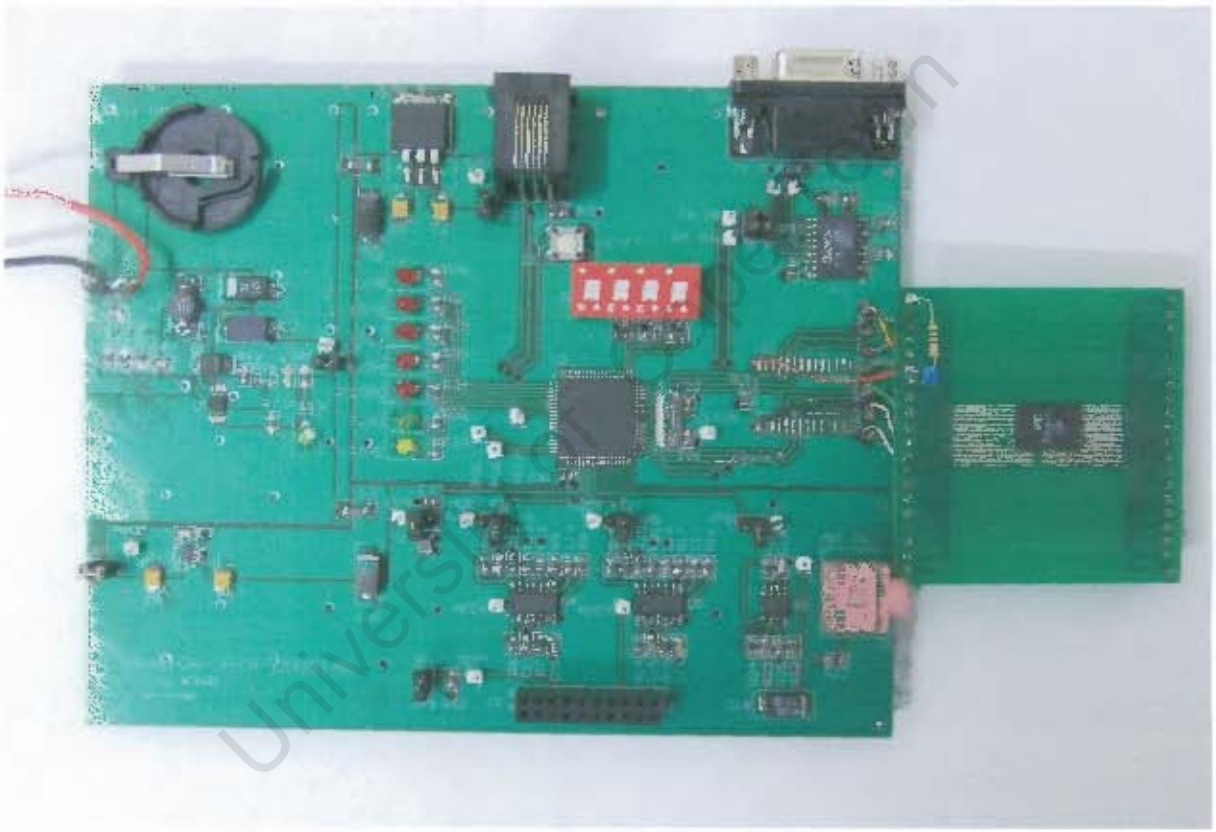
*Vibration Data Logging Board ver. 1.0 – Main Board*



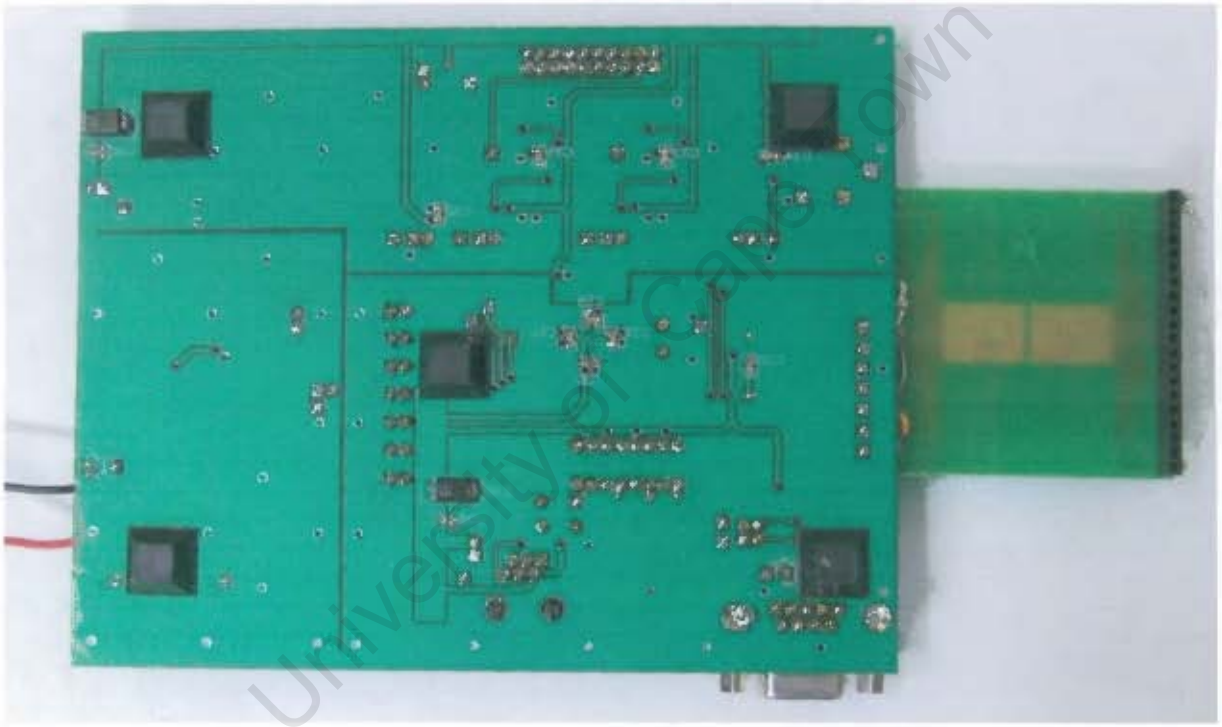
*Vibration Data Logging Board ver. 1.0 – Accelerometer Board*



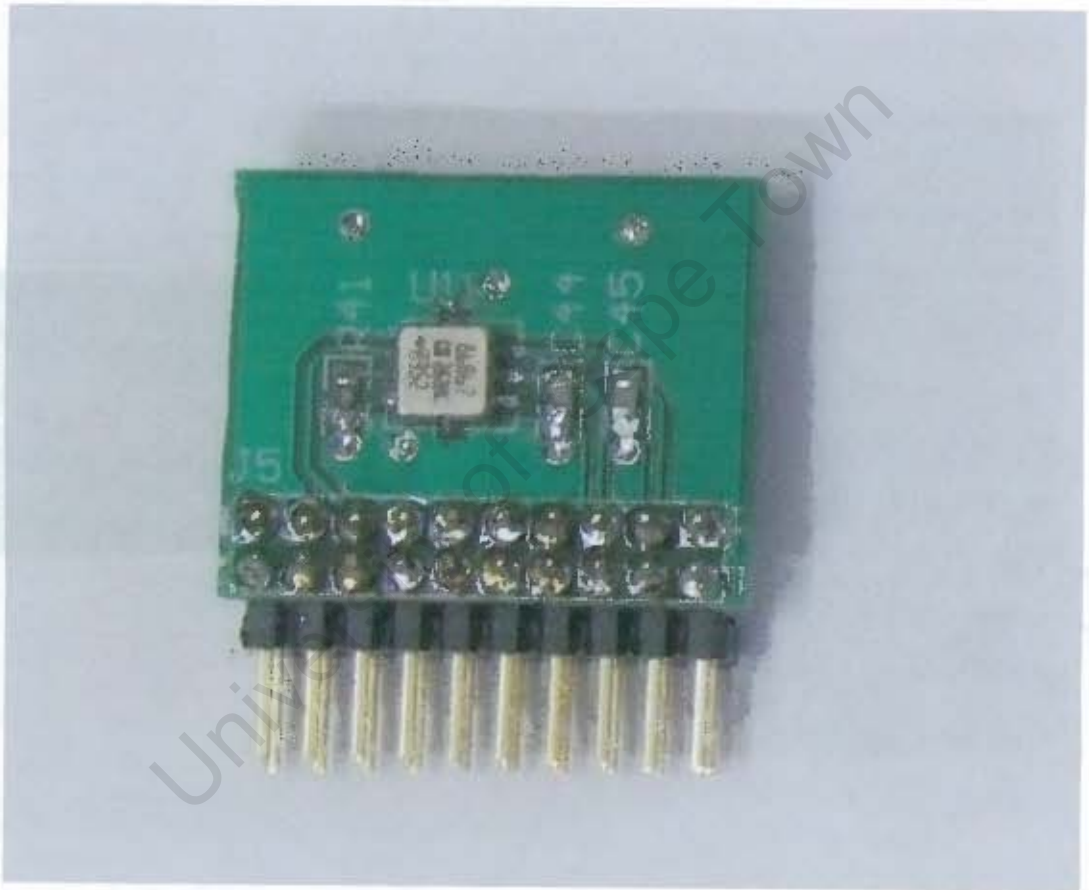
*Vibration Data Logging Board ver. 2.0 – Main Board (Top)*



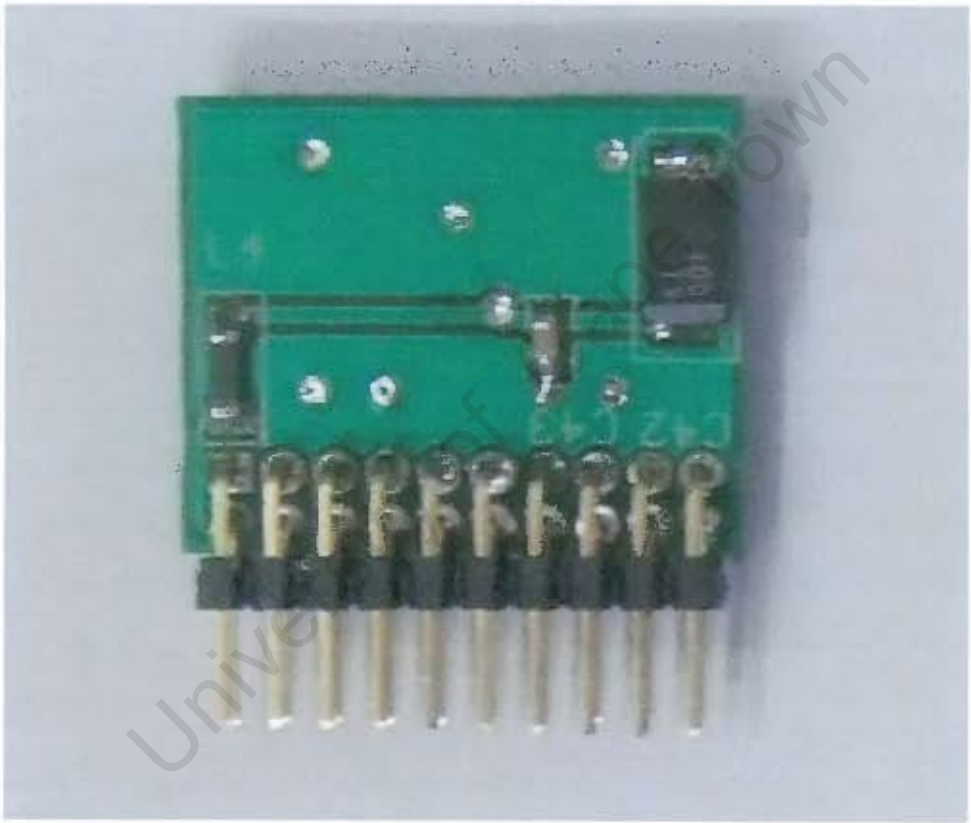
*Vibration Data Logging Board ver. 2.0 – Main Board (Bottom)*



*Vibration Data Logging Board ver. 2.0 –Accelerometer Board (Top)*



*Vibration Data Logging Board ver. 2.0 –Accelerometer Board (Bottom)*



# Appendix J

## Analogue to Digital Sampling Rate Calculations

Auto Sampling is used in the ADC module for the accelerometer and microphone inputs. The following calculations were used to determine the values of the *ADCS<5:0>* and *SAMC<4:0>* bits.

The instruction clock frequency is set by the crystal oscillator frequency and the internal PLL multiplier value. The frequency is set to 20MIPS for all programming operations using a 10 MIIz crystal and a PLL value of x8. This means that the instruction clock period is:

$$\begin{aligned} T_{cy} &= \frac{1}{f_{cy}} \\ &= \frac{1}{20MIPS} \\ &= 50ns \end{aligned}$$

A minimum analogue clock time of 668ns is required to ensure that the sampling capacitors do not lose charge. This means that there will be a minimum conversion time of

$$\begin{aligned} T_{AD} &\geq 668ns \\ T_{conv} &\geq 14T_{AD} \\ &\geq 9.352\mu s \end{aligned}$$

Therefore in this case the conversion time is set to a value of

$$T_{conv} = 20\mu s$$

### *Accelerometer Inputs*

The accelerometer signals must be sampled at 32 kHz. This gives a sampling period of

$$\begin{aligned} T_{SAMP} &= \frac{1}{f_S} \\ &= \frac{1}{32\times10^3} \\ &= 31.25\mu s \end{aligned}$$

By rearranging Equation 6.1 the *ADCS<5:0>* bits can be calculated

$$\begin{aligned} ADCS<5:0> &= \frac{2}{14}\left(\frac{T_{conv}}{T_{cy}}\right)-1 \\ &= \frac{1}{7}\left(\frac{20\times10^{-6}}{50\times10^{-9}}\right)-1 \\ &= 57.14285714 \\ &\approx 57 = 111001_b \end{aligned}$$

From this the actual value of the analogue clock can be calculated

$$\begin{aligned} Actual\ T_{AD} &= \frac{T_{cy}\left(ADCS<5:0>+1\right)}{2} \\ &= \frac{50\times10^{-9}\cdot\left(57+1\right)}{2} \\ &= 1.45\mu s \end{aligned}$$

This value can now be used in Equation 6.2 to find the *SAMC<4:0>* bits

$$\begin{aligned} SAMC<4:0> &= \frac{T_{SAMP}-14T_{AD}}{T_{AD}} \\ &= \frac{\left(31.25\times10^{-6}\right)-14\left(1.45\times10^{-6}\right)}{1.45\times10^{-6}} \\ &= 7.551724138 \\ &\approx 8 = 01000_b \end{aligned}$$

### *Electret Condenser Microphone*

The electret condenser microphone signal must be sampled at 20 kHz. This gives a sampling period of

$$\begin{aligned} T_{SAMP} &= \frac{1}{f_S} \\ &= \frac{1}{20\times10^{3}} \\ &= 50\mu s \end{aligned}$$

By rearranging Equation 6.1 the *ADCS<5:0>* bits can be calculated

$$\begin{aligned} ADCS<5:0> &= \frac{2}{14}\left(\frac{T_{conv}}{T_{cy}}\right)-1 \\ &= \frac{1}{7}\left(\frac{20\times10^{-6}}{50\times10^{-9}}\right)-1 \\ &= 57.14285714 \\ &\approx 57 = 111001_{b} \end{aligned}$$

From this the actual value of the analogue clock can be calculated

$$\begin{aligned} Actual\ T_{AD} &= \frac{T_{cy}\left(ADCS<5:0>+1\right)}{2} \\ &= \frac{50\times10^{-9}\cdot\left(57+1\right)}{2} \\ &= 1.45\mu s \end{aligned}$$

This value can now be used in Equation 6.2 to find the *SAMC<4:0>* bits

$$\begin{aligned} SAMC<4:0> &= \frac{T_{SAMP}-14T_{AD}}{T_{AD}} \\ &= \frac{\left(50\times10^{-6}\right)-14\left(1.45\times10^{-6}\right)}{1.45\times10^{-6}} \\ &= 20.48275862 \\ &\approx 21 = 10101_{b} \end{aligned}$$

# Appendix K

## Code for the Vibration Data Logging Board

### *Header Files – dsPIC30F6014Apins.h*

```
//=================================================================
//              VIBRATION SENSOR PIN AND INTERFACE DEFINITIONS
//=================================================================
//      BY:             ROBYN VERRINDER
//      DATE:           23 MAY 2005
//=================================================================
//      PROCESSOR:      dsPIC30F6014A
//      BOARD:          VIBRATION SENSOR PROTOTYPE BOARD
//      IDE:            MPLAB IDE v7.30
//      COMPILER:       C30 C COMPILER
//=================================================================
//      DESCRIPTION:    Definition of all connections to the dsPIC30F6014A
//                      microcontroller.
//
//                      The pin descriptions are as described in the various
//                      data sheets for the devices and follow the naming
//                      standard on the schematics for the Vibration Data
//                      Logging Prototype Board (Chapter 5)
//=================================================================
//      PIN CONNECTIONS AND CONFIGURATIONS:
//
//              1       =       COFS/RG15 ............ NOT CONNECTED
//              2       =       T2CK/RC1 ............. LED1 .................. OUTPUT
//              3       =       T3CK/RC2 ............. LED2 .................. OUTPUT
//              4       =       T4CK/RC3 ............. LED3 .................. OUTPUT
//              5       =       T5CK/RC4 ............. LED4 .................. OUTPUT
//              6       =       SCK2/CN8/RG6 ......... RECORD_LED ............ OUTPUT
//              7       =       SDI2/CN9/RG7 ......... SEND_LED .............. OUTPUT
//              8       =       SDO2/CN10/RG8 ........ ERASE_LED ............. OUTPUT
//              9       =       MCLR ................. MCLR .................. ICD2
//              10      =       SS2/CN11/RG9 ......... NOT CONNECTED
//              11      =       VSS .................. SGND (Signal Gnd) ..... POWER
//              12      =       VDD .................. VDD (Signal Power)..... POWER
//              13      =       INT1/RA12 ............ NOT CONNECTED
//              14      =       INT2/RA13 ............ NOT CONNECTED
//              15      =       AN5/CN7/RB5 .......... NOT CONNECTED
//              16      =       AN4/CN6/RB4 .......... NOT CONNECTED
//              17      =       AN3/CN5/RB3 .......... NOT CONNECTED
//              18      =       AN2/CN4/RB2 .......... NOT CONNECTED
//              19      =       AN1/PGC/CN3 .......... PGC ................... ICD2
//              20      =       AN0/PGD/CN2/RB0 ...... PGD ................... ICD2
//              21      =       AN6/OCFA/RB6 ......... NOT CONNECTED
//              22      =       AN7/RB7 .............. NOT CONNECTED
//              23      =       VREF-/RA9 ............ NOT CONNECTED
//              24      =       VREF+/RA10 ........... NOT CONNECTED
//              25      =       AVDD ................. AVDD (Analog Power) ... ANALOG POWER
//              26      =       AVSS ................. AGND (Analog Power) ... ANALOG POWER
//              27      =       AN8/RB8 .............. TEMP_ADC (MAX6608) .... ANALOG INPUT
//              28      =       AN9/RB9 .............. ACCX_ADC (ADXL202) .... ANALOG INPUT
//              29      =       AN10/RB10 ............ ACCY_ADC (ADXL202) .... ANALOG INPUT
//              30      =       AN11/RB11 ............ MIC_ADC (Electret Mic). ANALOG INPUT
//              31      =       VSS .................. SGND (Signal Gnd) ..... POWER
//              32      =       VDD .................. VDD (Signal Power) .... POWER
//              33      =       AN12/RB12 ............ NOT CONNECTED
//              34      =       AN13/RB13 ............ NOT CONNECTED
//              35      =       AN14/RB14 ............ NOT CONNECTED
//              36      =       AN15/OCFB/CN12/RB15 .. NOT CONNECTED
//              37      =       IC7/CN20/RD14 ........ NOT CONNECTED
//              38      =       IC8/CN21/RD15 ........ NOT CONNECTED
//              39      =       U2RX/CN17/RF4 ........ NOT CONNECTED
```

```
//              40      =       U2TX/CN18/RF5 ........ NOT CONNECTED
//              41      =       U1TX/RF3 ............. TX_IN (RS232 TX) ..... OUTPUT
//              42      =       U1RX/RF2 ............. RX_OUT (RS232 RX) .... INPUT
//              43      =       SDO1/RF8 ............. SO (Serial Output) ... OUTPUT
//              44      =       SDI1/RF7 ............. SI (Serial Input) .... INPUT
//              45      =       SCK1/INT0/RF6 ........ SCK (Serial Clock) ... OUTPUT
//              46      =       SDA/RG3 .............. NOT CONNECTED
//              47      =       SCL/RG2 .............. NOT CONNECTED
//              48      =       VDD .................. VDD (Signal Power) ... POWER
//              49      =       OSC1 ................. OSC1 ................. XTAL OSC
//              50      =       OSC2 ................. OSC2 ................. XTAL OSC
//              51      =       VSS .................. SGND (Signal Gnd) .... POWER
//              52      =       INT3/RA14 ............ NOT CONNECTED
//              53      =       INT4/RA15 ............ NOT CONNECTED
//              54      =       IC1/RD8 .............. CS (Chip Select) ..... OUTPUT
//              55      =       IC2/RD9 .............. READY/BUSY ........... INPUT
//              56      =       IC3/RD10 ............. RESET ................ OUTPUT
//              57      =       IC4/RD11 ............. WP (Write Protect) ... OUTPUT
//              58      =       OC1/RD0 .............. NOT CONNECTED
//              59      =       SOSC2/CN1/RC13 ....... NOT CONNECTED
//              60      =       SOSC1/T1CK/RC14 ...... NOT CONNECTED
//              61      =       OC2/RD1 .............. SEND ................. INPUT
//              62      =       OC3/RD2 .............. RECORD ............... INPUT
//              63      =       OC4/RD3 .............. ERASE ................ INPUT
//              64      =       IC5/RD12 ............. NOT CONNECTED
//              65      =       IC6/CN19/RD13 ........ NOT CONNECTED
//              66      =       OC5/CN13/RD4 ......... NOT CONNECTED
//              67      =       OC6/CN14/RD5 ......... NOT CONNECTED
//              68      =       OC7/CN15/RD6 ......... NOT CONNECTED
//              69      =       OC8/CN16/RD7 ......... NOT CONNECTED
//              70      =       VSS .................. SGND (Gnd) ........... POWER
//              71      =       VDD .................. VDD (Power) .......... POWER
//              72      =       C1RX/RF0 ............. NOT CONNECTED
//              73      =       C1TX/RF1 ............. NOT CONNECTED
//              74      =       C2TX/RG1 ............. NOT CONNECTED
//              75      =       C2RX/RG0 ............. NOT CONNECTED
//              76      =       CN22/RA6 ............. NOT CONNECTED
//              77      =       CN23/RA7 ............. NOT CONNECTED
//              78      =       CSCK/RG14 ............ NOT CONNECTED
//              79      =       CSDI/RG12 ............ NOT CONNECTED
//              80      =       CSDO/RG13 ............ NOT CONNECTED
//=============================================================================
//      VIBRATION DATA LOGGING PROTOTYPE BOARD IC LIST:
//      U1      =       LM2623MM ............. General Purpose, Gated Oscillator Based, DC/DC
//                                              Boost Converter, 5V Output [50]
//      U2      =       TC1264-3.3V .......... 3.3V, 800mA Low Voltage Dropout Regulator [51]
//      U3      =       TC2185-3.3V .......... 3.3V, 150mA Low Voltage Dropout Regulator [52]
//      U4      =       MCP6024 .............. Quad 2.7V to 5.5V Single-Supply CMOS Op Amp
//      U5      =       MCP6024 .............. Quad 2.7V to 5.5V Single-Supply CMOS Op Amp
//      U6      =       MCP6022 .............. Dual 2.7V to 5.5V Single-Supply CMOS Op Amp [56]
//      U7      =       MAX6608UK ............ Low-Voltage Analogue Temperature Sensor [46]
//      U8      =       dsPIC30F6014A ........ General Purpose Digital Signal Controller [47]
//      U9      =       AT45DB321C ........... 32-Mbit Serial Data Flash [48]
//      U10     =       SP3222EET ............ +3.0V to +5.5V RS-232 Transceivers [49]
//      U11     =       ADXL202E ............. +/-2g Dual-Axis Accelerometer with Duty Cycle
//                                              Output [45]
//=============================================================================

#ifndef         __DSPIC30F6014APINS_H
#define         __DSPIC30F6014APINS_H

#include        "p30F6014A.h"

//=============================================================================
//      U11: ADXL202E   =       +/- 2 g Dual-Axis Accelerometer with Duty Cycle Output
//=============================================================================
//      PIN OUTS:
//      1       =       ST ................... SELF TEST
//      2       =       T2 ................... CONNECT RESET TO SET T2 PERIOD
//      3       =       COM .................. COMMON
//      4       =       YOUT ................. Y-CHANNEL DUTY CYCLE OUTPUT
//      5       =       XOUT ................. X-CHANNEL DUTY CYCLE OUTPUT
//      6       =       YFILT ................ Y-CHANNEL FILTER PIN
//      7       =       XFILT ................ X-CHANNEL FILTER PIN
//      8       =       VDD .................. 3V - 5.25V
//
```

```
//      PIN CONNECTIONS: (ADXL202E to dsPIC30F6014A)
//      VDD     =       AVDD
//      COM     =       AGND
//      ACCX -> U4 (PIN3) -> (PIN14) ACCX_FILT -> JP4 -> ACCX_ADC    =       AN9   (PIN 28)
//      ACCY -> U5 (PIN3) -> (PIN14) ACCY_FILT -> JP5 -> ACCY_ADC    =       AN10  (PIN 29)
//
//-------------------------------------------------------------------------
//      PIN DEFINITIONS:
//-------------------------------------------------------------------------

#define         ACCX_ADC        PORTBbits.RB9
#define         ACCY_ADC        PORTBbits.RB10

//=========================================================================
//      U7:     MAX6608 =       Low-Voltage Analogue Temperature Sensor
//=========================================================================
//      PIN OUTS:
//      1       =       NC ................. NOT CONNECTED
//      2       =       GND ................ GROUND
//      3       =       A .................. MUST BE CONNECTED TO GND
//      4       =       VCC ................ SUPPLY INPUT
//      5       =       OUT ................ TEMPERATURE SENSOR OUTPUT
//
//      PIN CONNECTIONS: (MAX6608 to dsPIC30F6014A)
//      VCC     =       AVDD
//      GND     =       AGND
//      A       =       AGND
//      OUT (TEMP) -> JP7 -> TEMP_ADC =       AN8 (PIN 27)
//-------------------------------------------------------------------------
//      PIN DEFINITIONS:
//-------------------------------------------------------------------------

#define         TEMP_ADC        PORTBbits.RB8

//=========================================================================
//      MK1:    Electret Condenser Microphone
//=========================================================================
//      PIN OUTS:
//      1       =       GND ................ GROUND
//      2       =       SIGNAL ............. OUTPUT SIGNAL
//      3       =       BIAS ............... BIAS VOLTAGE 3.3V - 12V
//
//      PIN CONNECTIONS: (MK1 to dsPIC30F6014A)
//      GND     =       AGND
//      SIGNAL (MIC_IN) -> U6 (PIN2) -> (PIN7) MIC_FILT -> JP6 -> MIC_ADC   =  AN11 (PIN 30)
//      BIAS(VIA R22)   =       AVDD
//-------------------------------------------------------------------------
//      PIN DEFINITIONS:
//-------------------------------------------------------------------------

#define         MIC_ADC                 PORTBbits.RB11

//=========================================================================
//      U9:  AT45DB321C =       32-Mbit Serial Data Flash
//=========================================================================
//      PIN OUTS:
//      1       =       GND ................ GROUND
//      2       =       NC ................. NOT CONNECTED
//      3       =       NC ................. NOT CONNECTED
//      4       =       CS ................. CHIP SELECT
//      5       =       SCK ................ SERIAL CLOCK
//      6       =       SI ................. SERIAL INPUT
//      7       =       SO ................. SERIAL OUTPUT
//      8       =       NC ................. NOT CONNECTED
//      9       =       NC ................. NOT CONNECTED
//      10      =       NC ................. NOT CONNECTED
//      11      =       NC ................. NOT CONNECTED
//      12      =       NC ................. NOT CONNECTED
//      13      =       NC ................. NOT CONNECTED
//      14      =       NC ................. NOT CONNECTED
//      15      =       NC ................. NOT CONNECTED
//      16      =       NC ................. NOT CONNECTED
//      17      =       NC ................. NOT CONNECTED
//      18      =       NC ................. NOT CONNECTED
//      19      =       NC ................. NOT CONNECTED
//      20      =       NC ................. NOT CONNECTED
//      21      =       NC ................. NOT CONNECTED
```

```
//      22      =       NC ..................... NOT CONNECTED
//      23      =       RDY/BUSY ............... READY/BUSY
//      24      =       RESET .................. CHIP RESET
//      25      =       WP ..................... HARDWARE PAGE WRITE PROTECT
//      26      =       NC ..................... NOT CONNECTED
//      27      =       NC ..................... NOT CONNECTED
//      28      =       VCC .................... POWER SUPPLY
//
//      PIN CONNECTIONS: (AT45DB321B to dsPIC30F6014A)
//      VCC             =  VDD
//      GND             =  SGND
//      SCK             =  SCK1 (PIN 45)
//      SI              =  SDI1 (PIN 44)
//      SO              =  SDO1 (PIN 43)
//      CS              =  RD8  (PIN 54)
//      RDY/BUSY        =  RD9  (PIN 55)
//      RESET           =  RD10 (PIN 56)
//      WP              =  RD11 (PIN 57)
//---------------------------------------------------------------------
//      PIN DEFINITIONS:
//---------------------------------------------------------------------

#define         SCK                     PORTFbits.RF6
#define         SI                      PORTFbits.RF7
#define         SO                      PORTFbits.RF8
#define         CS                      PORTDbits.RD8
#define         READY_BUSY              PORTDbits.RD9
#define         RESET                   PORTDbits.RD10
#define         WP                      PORTDbits.RD11

//=====================================================================
//      U10:  SP3222 = +3.0V to +5.5V RS-232 Transceivers
//=====================================================================
//      PIN OUTS:
//      1       =       EN ..................... RECEIVER ENABLE CONTROL
//      2       =       C1+ .................... +VE TERMINAL FOR 1ST CHARGE PUMP CAP
//      3       =       V+ ..................... 5.5V GENERATED BY THE CHARGE PUMP
//      4       =       C1- .................... -VE TERMINAL FOR 1ST CHARGE PUMP CAP
//      5       =       C2+ .................... +VE TERMINAL FOR 2ND CHARGE PUMP CAP
//      6       =       C2- .................... -VE TERMINAL FOR 2ND CHARGE PUMP CAP
//      7       =       V- ..................... -5.5V GENERATED BY THE CHARGE PUMP
//      8       =       T2OUT .................. 2ND TRANSMITTER OUTPUT VOLTAGE
//      9       =       R2IN ................... 2ND RECEIVER INPUT VOLTAGE
//      10      =       R2OUT .................. 2ND RECEIVER OUTPUT VOLTAGE
//      11      =       T2IN ................... 2ND TRANSMITTER INPUT VOLTAGE
//      12      =       T1IN ................... 1ST TRANSMITTER INPUT VOLTAGE
//      13      =       R1OUT .................. 1ST RECEIVER OUTPUT VOLTAGE
//      14      =       R1IN ................... 1ST RECEIVER INPUT VOLTAGE
//      15      =       T1OUT .................. 1ST TRANSMITTER OUTPUT VOLTAGE
//      16      =       GND .................... GROUND
//      17      =       VCC .................... SUPPLY VOLTAGE
//      18      =       SHDN ................... ACTIVE LOW SHUTDOWN CONTROL INPUT

//
//  PIN CONNECTIONS: (SP3222 to dsPIC30F6014A)
//      VCC     =  VDD
//      GND     =  SGND
//      TX_IN   =  RF3  (PIN 41)
//      RX_OUT  =  RF2  (PIN 42)
//---------------------------------------------------------------------
//      PIN DEFINITIONS:
//---------------------------------------------------------------------

#define         TX_IN   PORTFbits.RF3
#define         RX_OUT  PORTFbits.RF2

//=====================================================================
//      DIP SWITCH INPUT (SWITCHES)
//=====================================================================
//      SWITCHES:
//      1       =       SEND
//      2       =       RECORD
//      3       =       ERASE
//      4       =       NOT USED
//
```

```
//      PIN CONNECTIONS: (DIP SWITCH TO dsPIC30F6014A)
//      SEND    =       RD1 (PIN 61)
//      RECORD  =       RD2 (PIN 62)
//      ERASE   =       RD3 (PIN 63)
//-------------------------------------------------------------------
//      PIN DEFINITIONS:
//-------------------------------------------------------------------

#define         SEND    PORTDbits.RD1
#define         RECORD  PORTDbits.RD2
#define         ERASE   PORTDbits.RD3

//===================================================================
//      LEDS
//===================================================================
//  PIN CONNECTIONS: (LEDS to dsPIC30F6014A)
//      LED1            =       RC1 (PIN 2)
//      LED2            =       RC2 (PIN 3)
//      LED3            =       RC3 (PIN 4)
//      LED4            =       RC4 (PIN 5)
//
//      RECORD_LED      =       RG6 (PIN 6)
//      SEND_LED        =       RG7 (PIN 7)
//      ERASE_LED       =       RG8 (PIN 8)
//-------------------------------------------------------------------
//      PIN DEFINITIONS:
//-------------------------------------------------------------------

#define         LED1    PORTCbits.RC1
#define         LED2    PORTCbits.RC2
#define         LED3    PORTCbits.RC3
#define         LED4    PORTCbits.RC4

#define         RECORD_LED      PORTGbits.RG6
#define         SEND_LED        PORTGbits.RG7
#define         ERASE_LED       PORTGbits.RG8

//===================================================================

#endif
```

## *Header Files – AT45DB321C.h*

```
//=====================================================================
//                      AT45DB321C HEADER FILE
//=====================================================================
//      BY:     ROBYN VERRINDER
//      DATE:   13 APRIL 2006
//=====================================================================
//      PROCESSOR:      dsPIC30F6014A
//      BOARD:          VIBRATION DATA LOGGING PROTOTYPE BOARD
//      IDE:            MPLAB IDE v7.30
//      COMPILER:       C30 C COMPILER
//=====================================================================
//      DESCRIPTION:    Header file for AT45DB321B Library functions and
//                      definitions.
//      CONTENTS:       Function Prototypes
//                      Opcode Command Constants
//=====================================================================
//      CHIP:           AT45DB321C 32-megabit DataFlash Chip
//      INTERFACE:      SPI (Serial Peripheral Interface)
//=====================================================================
//      MEMORY SIZE:    32 Mbits
//      BYTES/PAGE:     528 bytes/page
//      PAGES/BLOCK:    8 blocks/page
//      NO OF PAGES:    8192 pages
//      NO OF BLOCKS:   1024 blocks
//=====================================================================

#ifndef         __AT45DB321C_H
#define         __AT45DB321C_H

//=====================================================================
//      PIN OUTS:
//       CS       =  Chip Select ................ LOW = Selected, HIGH = Deselected
//       SCK      =  Serial Clock ............... Can be clocked up to 20Mhz
//       SI       =  Serial Data In ............. Input only, used to shift data into device
//       SO       =  Serial Data Out ............ Output only, used to shift data out of device
//       WP       =  Hardware Page Write Protect  LOW = cannot reprogramme 1st 256 pages of
//                                                main mem, HIGH = can programme
//       RESET    =  Chip Reset ................. LOW = reset chip, HIGH = normal operation
//       RDY/BUSY =  Ready/Busy ................. Output pin - LOW = Device is BUSY
//
//  PIN CONNECTIONS: (AT45DB321C to dsPIC30F6014A)      (dsPIC30F6014A PORT DIRECTION)
//       SCK      =  SCK1 (PIN 45) ................. OUTPUT
//       SI       =  SDO1 (PIN 43) ................. OUTPUT
//       SO       =  SDI1 (PIN 44) ................. INPUT
//       CS       =  RD8  (PIN 54) ................. OUTPUT
//       RDY/BUSY =  RD9  (PIN 55)................. INPUT
//       RESET    =  RD10 (PIN 56) ................. OUTPUT
//       WP       =  RD11 (PIN 57) ................. OUTPUT
//=====================================================================
//      MODES:
//      Inactive Clock Polarity:
//       LOW ........... (SPIxCONbits.CKP = 0) .... Idle clock state is LOW
//       HIGH .......... (SPIxCONbits.CKP = 1) .... Idle clock state is HIGH
//
//      SPI Mode:
//      SPI Mode 0 .... (SPIxCONbits.CKE = 0) .... Data clocked out on 2nd pulse (ACTIVE ->
//                                                 IDLE State)
//      SPI Mode 3 .... (SPIxCONbits.CKE = 1) .... Data clocked out on 1st pulse (IDLE ->
//                                                 Active State)
//=====================================================================
//      DEFINING DATA CLOCKING MODES
//---------------------------------------------------------------------
//      DESCRIPTION: Defines constants for data mode selection
//---------------------------------------------------------------------

#define         INACTIVE_CLKPOLARITY_LOW        0       //      State of clock in IDLE Mode
#define         INACTIVE_CLKPOLARITY_HIGH       1

#define         SPI_MODE0                       0       //      Edge of CLK on which to CLK in
                                                        //      data
#define         SPI_MODE3                       1
```

```
#define         INACTIVE_CLKPOLARITY    INACTIVE_CLKPOLARITY_LOW        //      Selects Clock LOW
                                                                        //      in IDLE mode
#define         SPI_MODE                SPI_MODE0       //      Selects SPI Mode 3


//---------------------------------------------------------------------
//      STATUS REGISTER - 8 BIT
//---------------------------------------------------------------------
//      BIT7    =       RDY/BUSY ..............  LOW = busy, HIGH = not busy
//      BIT6    =       COMP ..................  LOW = data in main memory matches buffer, HIGH
//                                                = at least 1 bit of data in main memory does
//                                                not match data in buffer
//      BIT5    =       1 ......................  Bits 5-2 = device density
//      BIT4    =       1
//      BIT3    =       0
//      BIT2    =       1
//      BIT1    =       X
//      BIT0    =       X
//---------------------------------------------------------------------
//      BUFFER CONSTANTS
//---------------------------------------------------------------------
#define         BUFFER1         0x01
#define         BUFFER2         0x02

#define         MEM_BUFFERSIZE 528      //      Size of SRAM buffer onboard AT45DB321C
//=====================================================================
//      OPCODE INSTRUCTIONS: (COMMAND_TYPE_NAME_MODE)
//=====================================================================
//  Read Commands: (CMD_Read_....)
//---------------------------------------------------------------------
//      CONTINUOUSARRAY ......... 0xE8 .....  Reads a continuous stream of data from device
//                                            with just clk signal
//      MAINMEMPAGE ............. 0xD2 .....  Reads data directly from main memory bypassing
/                                             buffers
//      BUFFER1 ................. 0xD4 .....  Reads data from buffer 1
//      BUFFER2 ................. 0xD6 .....  Reads data from buffer 2
//      STATUSREG ............... 0xD7 .....  Reads status register of device
//---------------------------------------------------------------------
#define         CMD_READ_CONTINUOUSARRAY        0xE8
#define         CMD_READ_MAINMEMPAGE            0xD2
#define         CMD_READ_BUFFER1                0xD4
#define         CMD_READ_BUFFER2                0xD6
#define         CMD_READ_STATUSREG              0xD7
//---------------------------------------------------------------------
//  Write Commands: (CMD_Write/Prog_....)
//---------------------------------------------------------------------
//      BUFFER1 ................. 0x84 .....  Writes to buffer 1
//      BUFFER2 ................. 0x87 .....  Writes to buffer 2
//      BUF1toMAINMEM_WERASE .... 0x83 .....  First erases page (makes every entry a 1), then
//                                            programs main memory from buffer 1
//      BUF2toMAINMEM_WERASE .... 0x86 .....  First erases page (makes every entry a 1), then
//                                            programs main memory from buffer 2
//      BUF1toMAINMEM_WOERASE ... 0x88 .....  Programs main memory from buffer 1 with out erase
//                                            (only program a page if it has previously been
//                                            erased)
//      BUF2toMAINMEM_WOERASE ... 0x89 .....  Programs main memory from buffer 2 with out erase
//                                            (only program a page if it has previously been
//                                            erased)
//      MAINMEMPAGE_THRUBUF1 .... 0x82 .....  Data is shifted in buffer 1, erases page and then
//                                            programs a page in main memory
//      MAINMEMPAGE_THRUBUF2 .... 0x85 .....  Data is shifted in buffer 2, erases page and then
//                                            programs a page in main memory
//---------------------------------------------------------------------
#define         CMD_WRITE_BUFFER1               0x84
#define         CMD_WRITE_BUFFER2               0x87
#define         CMD_PROG_BUF1toMAINMEM_WERASE 0x83
#define         CMD_PROG_BUF2toMAINMEM_WERASE 0x86
#define         CMD_PROG_BUF1toMAINMEM_WOERASE0x88
#define         CMD_PROG_BUF2toMAINMEM_WOERASE0x89
#define         CMD_PROG_MAINMEMPAGE_THRUBUF1 0x82
#define         CMD_PROG_MAINMEMPAGE_THRUBUF2 0x85
//---------------------------------------------------------------------
//      Erase Commands: (CMD_Erase_....)
//---------------------------------------------------------------------
//      PAGE .................... 0x81 .....  Erases an individual page in memory
//      BLOCK ................... 0x50 .....  Erases 8 pages of main memory
//---------------------------------------------------------------------
```

```
#define         CMD_ERASE_PAGE          0x81
#define         CMD_ERASE_BLOCK         0x50

//-------------------------------------------------------------------
//  Transfer Commands: (CMD_Transfer_....)
//-------------------------------------------------------------------
//      MAINMEMPAGEtoBUF1 ....... 0x53 ..... Transfers a page of data from main memory to
//                                           buffer 1
//      MAINMEMPAGEtoBUF2 ....... 0x55 ..... Transfers a page of data from main memory to
//                                           buffer 2
//-------------------------------------------------------------------
#define         CMD_TRANSFER_MAINMEMPAGEtoBUF1  0x53
#define         CMD_TRANSFER_MAINMEMPAGEtoBUF2  0x55
//-------------------------------------------------------------------
//      Compare Commands: (CMD_Compare_....)
//-------------------------------------------------------------------
//      MAINMEMPAGEtoBUF1 ....... 0x60 ..... Compares a page of data in main mem to buffer 1
//      MAINMEMPAGEtoBUF2 ....... 0x61 ..... Compares a page of data in main mem to buffer 2
//-------------------------------------------------------------------
#define         CMD_COMPARE_MAINMEMPAGEtoBUF1 0x60
#define         CMD_COMPARE_MAINMEMPAGEtoBUF2 0x61
//-------------------------------------------------------------------
//      Rewrite Commands: (CMD_ReWrite_....)
//-------------------------------------------------------------------
//      AUTOPAGE_THRUBUF1 ....... 0x58 ..... Combination of main memory to buffer 1 transfer
//                                           and main memory page program with erase
//      AUTOPAGE_THRUBUF2 ....... 0x59 ..... Combination of main memory to buffer 2 transfer
//                                           and main memory page program with erase
//-------------------------------------------------------------------
#define         CMD_REWRITE_AUTOPAGE_THRUBUF1 0x58
#define         CMD_REWRITE_AUTOPAGE_THRUBUF2 0x59
//===================================================================

#endif
```

## *Header Files – common.h*

```
//=====================================================================
//                    COMMON DEFINITIONS HEADER FILE
//=====================================================================
//      BY:     ROBYN VERRINDER
//      DATE:   18 APRIL 2006
//=====================================================================
//      PROCESSOR: dsPIC30F6014A
//      BOARD:     VIBRATION DATA LOGGING PROTOTYPE BOARD
//      IDE:       MPLAB IDE v7.30
//      COMPILER: C30 C COMPILER
//=====================================================================
//      DESCRIPTION:    Header file of common definitions used in the software
//                      for the Vibration Data Logging Prototype Board.
//=====================================================================

#ifndef         __COMMON_H
#define         __COMMON_H

//=====================================================================
//      BOOLEAN CONSTANTS
//=====================================================================
#define         TRUE    0x01
#define         FALSE   0x00
//=====================================================================
//      TIMING
//=====================================================================
#define         FXTAL   10000000                //      Crystal Frequency

#define         PLL_4   4                       //      PLLx4 Mode
#define         PLL_8   8                       //      PLLx8 Mode
#define         PLL     PLL_8                   //      PLL Mode Selection

#define         FCY     (FXTAL*PLL)/4           //      FCY = 20MIPS
#define         TMS     FCY/100                 //      Time divider for milliseconds

#define         MILLISEC  FCY/TMS               //      Millisecond delay for delay loop

#define         ACC_FS  30000                   //      Accelerometers Sampling rate = 15kHz per
                                                //      channel
#define         MIC_FS  20000                   //      Microphone Sampling rate = 20kHz
#define         SINE_FS 5000                    //      Sinewave ADC Sampling rate = 5kHz
//=====================================================================
//      ADC SELECTION
//=====================================================================

#define         ADC_BUFFERSIZE 8                //      Size of the ADC Buffer
#define         ADC_ACCURACY   12               //      Accuracy of the ADC Module in bits

#define         OFFSET         0x0819           //      Offset due to amplifier

//=====================================================================
//      COMMUNICATION SETTINGS
//=====================================================================
#define         BAUD    57600                   //      Baud rate setting for UART module

#define         TX_BUFFERSIZE  9                //      Transmit buffer size for UART module
//=====================================================================
#endif
```

## *Main Program*

```
//=====================================================================
//          VIBRATION DATA LOGGING BOARD SOFTWARE PROGRAM
//=====================================================================
//      BY:             ROBYN VERRINDER
//                      Parts of this program were adapted from the
//                      Example Code for the dsPICDEM Low Cost Starter Board
//                      ADC12_UART.c (D'Souza, S., 2003) [68];
//                      the example code CE001_ADC_DSP_lib_FILTER ADC.c
//                      (Vasuki, H., 2005) [69]; from the source code
//                      to test the validity of the Dataflash® chip
//                      (Stowe, G., 2006) [65].
//      DATE:           06 MAY 2006
//=====================================================================
//      PROCESSOR:      dsPIC30F6014A
//      BOARD:          VIBRATION DATA LOGGING PROTOTYPE BOARD
//      IDE:            MPLAB IDE v7.30
//      COMPILER:       C30 C COMPILER
//=====================================================================
//      DESCRIPTION:    This program integrates all of the functions used to
//                      control the modules on board the Vibration Data Logging
//                      board.
//
//                      Register descriptions are as in dsPIC30F Family Reference Manual
//                      Section 12: Timers, Section 18: 12 bit A/D Converter and Section 19:
//                      UART(Microchip Technology Incorporated Reference Manual, 2003) [63].
//=====================================================================
//      XTAL FREQ (Fxtal):      10MHz
//      MIPS:                   20MIPS  (XT PLLx8 Mode)
//      TCY:                    50ns
//      Tconv:                  20us    (this value is greater than the 10us minimum)
//      Tad:                    1.45us  (calculated value using ADCS<5:0> bits)
//=====================================================================
#include        "p30f6014a.h"                   //      (Sinha, P., 2005) [64].
#include        <math.h>                        //      Standard C Math header file
#include        "dsPIC30F6014APins.h"
#include        "AT45DB321B.h"
#include        "common.h"
//---------------------------------------------------------------------
//      dsPIC30F6014A CONFIGURATION SETTINGS
//---------------------------------------------------------------------
//      CLOCK SWITCHING AND MONITOR ..... Sw Disabled, Mon Disabled
//      OSCILLATOR ...................... XT w/PLL 8x
//      WATCHDOG TIMER .................. Disabled
//      MASTER CLEAR ENABLE ............. Enabled
//      POR TIMER VALUE ................. 64ms
//      GENERAL CODE SEGMENT CODE PROTECT ..... Disabled
//---------------------------------------------------------------------
_FOSC(CSW_FSCM_OFF & XT_PLL8);
_FWDT(WDT_OFF);
_FBORPOR(MCLR_EN & PWRT_64);
_FGS(CODE_PROT_OFF);
//---------------------------------------------------------------------
//      FUNCTION DEFINITIONS
//---------------------------------------------------------------------

void    InitPorts(void)
void    InitADC12(void);
void    InitUART1(void);
void    InitSPI1(void);

unsigned char SendReceiveByte(unsigned char data);

void    Record(void);
void    Send(void);
void    Erase(void);

void    WritetoDataflash(unsigned char data);
void    ReadfromDataflash(void);

void    PagetoBuffer(unsigned char active_buffer, unsigned int page_counter);
void    ReadfromBuffer(unsigned char buffer, unsigned short address,
                       unsigned char *returned_data, unsigned short length);
```

```
void    SendtoUART1(unsigned char data[]);

unsigned int RMSCalculation(unsigned int *dataPtr, int N);

unsigned int Q15toInteger(signed int value);
void     asciiVoltsConv(const unsigned int value);
char     asciiConv(unsigned int value);

void    __attribute__((__interrupt__)) _U1TXInterrupt(void);

//-----------------------------------------------------------------
//      GLOBAL FLAGS
//-----------------------------------------------------------------

unsigned int   mem_is_clear = 0;            //      Main memory is empty flag
unsigned int   mem_is_full     = 0;         //      Main memory is full flag

//-----------------------------------------------------------------
//      ASCII BUFFERS FOR SERIAL DATA
//-----------------------------------------------------------------
//      BYTE         NAME          VALUE
//-----------------------------------------------------------------
//      8 ........ DATA<3> ........ 0x30 ........ Units
//      7 ........ '.' ............ 0x2E ........ Decimal Point
//      6 ........ DATA<2> ........ 0x31 ........ Tenths
//      5 ........ DATA<1> ........ 0x32 ........ Hundredths
//      4 ........ DATA<0> ........ 0x33 ........ Thousands
//      3 ........ SPACE .......... 0x20
//      2 ........ 'V' ............ 0x43
//      1 ........ LF ............. 0x0A
//      0 ........ CR ............. 0x0D
//-----------------------------------------------------------------

unsigned char  OutData[TX_BUFFERSIZE] = (0x30,0x2E,0x31,0x32,0x33,0x20,0x56,0x0A,0x0D);

//-----------------------------------------------------------------
//      GLOBAL VARIABLES
//-----------------------------------------------------------------

unsigned char  units;
unsigned char  tenths;
unsigned char  hundredths;
unsigned char  thousandths;

unsigned char  txCount;

//-----------------------------------------------------------------
//      MAIN FUNCTION
//-----------------------------------------------------------------

int     main(void)
{

        InitPorts();                            //      Initialises Ports
        InitADC12();                            //      Initialises the ADC12 module
        InitUART1();                            //      Initialises the UART1 module
        InitSPI1();                             //      Initialises the SPI1 module

        while(1)                                //      Do forever
        {
                if      (RECORD)                //      If RECORD has been selected
                {
                        Record();               //      Record Data
                )

                else if ((SEND)&(!RECORD))      //      Elseif SEND is selected and RECORD is not
                {
                        Send();                 //      Send collected data to PC via UART1
                )
                        else if ((ERASE)&(!SEND)&(!RECORD))   //      If ERASE is selected and
                                                              //      SEND and RECORD are not
                {
                        Erase();                              //      ERASE dataflash®
                )
        }
```

```
//=============================================================
//      PUBLIC FUNCTION:       InitPorts
//=============================================================
//      FUNCTION CALL: InitPorts();
//      ARGUMENTS:     None
//      RETURNS:       Nothing
//-------------------------------------------------------------
//      DESCRIPTION:   Initialises ports of the dsPIC30F6014A microprocessor
//                     on the Vibration Data Logging Prototype Board.
//
//                     TRISx ..... Sets direction of Port Pin (I/O)
//                     LATx ..... Sets the value of the internal latch
//                                of the port pin.
//                     X ..... Pin is unconnected
//-------------------------------------------------------------
//      PORT A:        PIN DEFINITIONS       TRISA = 0x0000 LATA = 0x0000
//-------------------------------------------------------------
//              PIN                    USE            TRISA          LATA
//      ------------------------------------------------------------------------
//              RA0     .....    NOT I/O ..... 0 ..... 0 ..... X
//              RA1     .....    NOT I/O ..... 0 ..... 0 ..... X
//              RA2     .....    NOT I/O ..... 0 ..... 0 ..... X
//              RA3     .....    NOT I/O ..... 0 ..... 0 ..... X
//              RA4     .....    NOT I/O ..... 0 ..... 0 ..... X
//              RA5     .....    NOT I/O ..... 0 ..... 0 ..... X
//              RA6     .....    OUTPUT ..... 0 ..... 0 ..... X
//              RA7     .....    OUTPUT ..... 0 ..... 0 ..... X
//              RA8     .....    NOT I/O ..... 0 ..... 0 ..... X
//              RA9     .....    OUTPUT ..... 0 ..... 0 ..... X
//              RA10    .....    OUTPUT ..... 0 ..... 0 ..... X
//              RA11    .....    NOT I/O ..... 0 ..... 0 ..... X
//              RA12    .....    OUTPUT ..... 0 ..... 0 ..... X
//              RA13    .....    OUTPUT ..... 0 ..... 0 ..... X
//              RA14    .....    OUTPUT ..... 0 ..... 0 ..... X
//              RA15    .....    OUTPUT ..... 0 ..... 0 ..... X
//-------------------------------------------------------------
//      PORT B:        PIN DEFINITIONS       TRISB = 0x0F03 LATB = 0x0000
//-------------------------------------------------------------
//              PIN                    USE            TRISB          LATB
//      ------------------------------------------------------------------------
//              RB0     .....    INPUT ..... 1 ..... 0 ..... PGD
//              RB1     .....    INPUT ..... 1 ..... 0 ..... PGC
//              RB2     .....    OUTPUT ..... 0 ..... 0 ..... X
//              RB3     .....    OUTPUT ..... 0 ..... 0 ..... X
//              RB4     .....    OUTPUT ..... 0 ..... 0 ..... X
//              RB5     .....    OUTPUT ..... 0 ..... 0 ..... X
//              RB6     .....    OUTPUT ..... 0 ..... 0 ..... X
//              RB7     .....    OUTPUT ..... 0 ..... 0 ..... X
//              RB8     .....    ANALOG INPUT ..... 1 ..... 0 ..... TEMP_ADC
//              RB9     .....    ANALOG INPUT ..... 1 ..... 0 ..... ACCX_ADC
//              RB10    .....    ANALOG INPUT ..... 1 ..... 0 ..... ACCY_ADC
//              RB11    .....    ANALOG INPUT ..... 1 ..... 0 ..... MIC_ADC
//              RB12    .....    OUTPUT ..... 0 ..... 0 ..... X
//              RB13    .....    OUTPUT ..... 0 ..... 0 ..... X
//              RB14    .....    OUTPUT ..... 0 ..... 0 ..... X
//              RB15    .....    OUTPUT ..... 0 ..... 0 ..... X
//-------------------------------------------------------------
//      PORT C:        PIN DEFINITIONS       TRISC = 0x0000  LATC = 0x0000
//-------------------------------------------------------------
//              PIN                    USE            TRISC          LATC
//      ------------------------------------------------------------------------
//              RC0     .....    OUTPUT ..... 0 ..... 0 ..... LED1
//              RC1     .....    OUTPUT ..... 0 ..... 0 ..... LED2
//              RC2     .....    OUTPUT ..... 0 ..... 0 ..... LED3
//              RC3     .....    OUTPUT ..... 0 ..... 0 ..... LED4
//              RC4     .....    NOT I/O ..... 0 ..... 0 ..... X
//              RC5     .....    NOT I/O ..... 0 ..... 0 ..... X
//              RC6     .....    NOT I/O ..... 0 ..... 0 ..... X
//              RC7     .....    NOT I/O ..... 0 ..... 0 ..... X
//              RC8     .....    NOT I/O ..... 0 ..... 0 ..... X
//              RC9     .....    NOT I/O ..... 0 ..... 0 ..... X
//              RC10    .....    NOT I/O ..... 0 ..... 0 ..... X
//              RC11    .....    NOT I/O ..... 0 ..... 0 ..... X
//              RC12    .....    NOT I/O ..... 0 ..... 0 ..... X
//              RC13    .....    OUTPUT ..... 0 ..... 0 ..... X
//              RC14    .....    OUTPUT ..... 0 ..... 0 ..... X
//              RC15    .....    OUTPUT ..... 0 ..... 0 ..... X
```

```
//-------------------------------------------------------------------
//      PORT D:         PIN DEFINITIONS        TRISD = 0x020E LATD = 0x0100
//-------------------------------------------------------------------
//              PIN                     USE             TRISD         LATD
//      ---------------------------------------------------------------------
//              RD0     ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RD1     ...........    INPUT ..... 1 ..... 0 ..... SEND
//              RD2     ...........    INPUT ..... 1 ..... 0 ..... RECORD
//              RD3     ...........    INPUT ..... 1 ..... 0 ..... ERASE
//              RD4     ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RD5     ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RD6     ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RD7     ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RD8     ...........    OUTPUT ..... 0 ..... 1 ..... CS
//              RD9     ...........    INPUT ..... 1 ..... 0 ..... R_B
//              RD10    ...........    OUTPUT ..... 0 ..... 0 ..... RESET
//              RD11    ...........    OUTPUT ..... 0 ..... 0 ..... WP
//              RD12    ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RD13    ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RD14    ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RD15    ...........    OUTPUT ..... 0 ..... 0 ..... X
//-------------------------------------------------------------------
//      PORT F:         PIN DEFINITIONS        TRISF = 0x0104  LATF = 0x0000
//-------------------------------------------------------------------
//              PIN                     USE             TRISF         LATF
//      ---------------------------------------------------------------------
//              RF0     ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RF1     ...........    INPUT ..... 0 ..... 0 ..... X
//              RF2     ...........    INPUT ..... 1 ..... 0 ..... RX_OUT
//              RF3     ...........    OUTPUT ..... 0 ..... 0 ..... TX_IN
//              RF4     ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RF5     ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RF6     ...........    OUTPUT ..... 0 ..... 0 ..... SCK
//              RF7     ...........    OUTPUT ..... 0 ..... 0 ..... SI
//              RF8     ...........    INPUT ..... 1 ..... 0 ..... SO
//              RF9     ...........    NOT I/O ..... 0 ..... 0 ..... X
//              RF10    ...........    NOT I/O ..... 0 ..... 0 ..... X
//              RF11    ...........    NOT I/O ..... 0 ..... 0 ..... X
//              RF12    ...........    NOT I/O ..... 0 ..... 0 ..... X
//              RF13    ...........    NOT I/O ..... 0 ..... 0 ..... X
//              RF14    ...........    NOT I/O ..... 0 ..... 0 ..... X
//              RF15    ...........    NOT I/O ..... 0 ..... 0 ..... X
//-------------------------------------------------------------------
//      PORT G:         PIN DEFINITIONS        TRISG = 0x0000  LATG = 0x0000
//-------------------------------------------------------------------
//              PIN                     USE             TRISG         LATG
//      ---------------------------------------------------------------------
//              RG0     ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RG1     ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RG2     ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RG3     ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RG4     ...........    NOT I/O ..... 0 ..... 0 ..... X
//              RG5     ...........    NOT I/O ..... 0 ..... 0 ..... X
//              RG6     ...........    OUTPUT ..... 0 ..... 0 ..... RECORD_LED
//              RG7     ...........    OUTPUT ..... 0 ..... 0 ..... SEND_LED
//              RG8     ...........    OUTPUT ..... 0 ..... 0 ..... ERASE_LED
//              RG9     ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RG10    ...........    NOT I/O ..... 0 ..... 0 ..... X
//              RG11    ...........    NOT I/O ..... 0 ..... 0 ..... X
//              RG12    ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RG13    ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RG14    ...........    OUTPUT ..... 0 ..... 0 ..... X
//              RG15    ...........    OUTPUT ..... 0 ..... 0 ..... X
//-------------------------------------------------------------------
void            InitPorts(void)
{
        TRISA   =       0x0000;                 //      Port A I/O direction
        TRISB   =       0x0F03;                 //      Port B I/O direction
        TRISC   =       0x0000;                 //      Port C I/O direction
        TRISD   =       0x020E;                 //      Port D I/O direction
        TRISF   =       0x0104;                 //      Port F I/O direction
        TRISG   =       0x0000;                 //      Port G I/O direction

        LATA    =       0x0000;                 //      Port A latch settings set to 0
        LATB    =       0x0000;                 //      Port B latch settings set to 0
        LATC    =       0x0000;                 //      Port C latch settings set to 0
```

```
        LATD    =       0x0F00;                 //      Port D latch settings set to 0
                                                //      (except CS,WP,RESET & READY/BUSY)
                                                //      R/B for the memory chip)
        LATF    =       0x0000;                 //      Port F latch settings set to 0
        LATG    =       0x0000;                 //      Port G latch settings set to 0

}

//=====================================================================
//      PUBLIC FUNCTION:        InitADC12
//=====================================================================
//      FUNCTION CALL:          InitADC12();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:            Initialises the 12 bit Analogue to Digital Converter
//                              All the registers are set as laid out below
//---------------------------------------------------------------------
//      SAMPLING RATE:          20 kHz
//---------------------------------------------------------------------
//      ADCON1:         A/D     CONTROL REGISTER 1              ADCON1 = 0x03E0
//---------------------------------------------------------------------
//      BIT       NAME            FUNCTION                VALUE       DESCRIPTION
//---------------------------------------------------------------------
//      15 ..... ADON ..... A/D OPERATING MODE ..... 0 ..... ADC is OFF
//      14 ................ UNIMPLEMENTED ........... 0 ..... X
//      13 ..... ADSIDL .... STOP IN IDLE MODE ...... 0 ..... Operates in IDLE mode
//      12 ................ UNIMPLEMENTED ........... 0 ..... X
//      11 ................ UNIMPLEMENTED ........... 0 ..... X
//      10 ................ UNIMPLEMENTED ........... 0 ..... X
//      9 ........ FORM<1> .. DATA OUTPUT FORMAT ..... 1 ..... Signed Fractional
//      8 ........ FORM<0>                            1       (DOUT = sddd dddd dddd 0000)
//      7 ........ SSRC<2> .. CONVERSION TRIGGER SOURCE 1 ..... Internal counter ends
//      6 ........ SSRC<1>                            1       sampling and starts
//      5 ........ SSRC<0>                            1       conversion (Auto convert)
//      4 ................ UNIMPLEMENTED ........... 0 ..... X
//      3 ................ UNIMPLEMENTED ........... 0 ..... X
//      2 ........ ASAM ..... A/D SAMPLE AUTO START ... 0 ..... Sampling begins when SAMP bit
//                                                              is set
//      1 ........ SAMP ..... A/D SAMPLE ENABLE ..... 0 ..... SAMP bit is auto set by ASAM
//                                                              setting
//      0 ........ DONE ..... A/D CONVERSION STATUS ... 0 ..... Automatically cleared and set
//                                                              by MCU
//---------------------------------------------------------------------
//      ADCON2:         A/D     CONTROL REGISTER 2              ADCON2 = 0x0020
//---------------------------------------------------------------------
//      BIT       NAME            FUNCTION                VALUE       DESCRIPTION
//---------------------------------------------------------------------
//      15 ..... VCFG<2> ... VOLTAGE REF CONFIG ..... 0 ..... A/D VREFH = AVDD
//      14 ..... VCFG<1>                              0       A/D VREFL = AVSS
//      13 ..... VCFG<0>                              0
//      12 ................ RESERVED ................ 0
//      11 ................ UNIMPLEMENTED ........... 0 ..... X
//      10 ..... CSCNA .... SCAN INPUTS ............ 0 ..... Do not scan inputs
//      9 ................. UNIMPLEMENTED ........... 0 ..... X
//      8 ................. UNIMPLEMENTED ........... 0 ..... X
//      7 ........ BUFS ..... BUFFER FILL STATUS ..... 0 ..... Auto set (1 = A/D is filling
//                                                              0x8-0xF find data
//                                                              in 0x0-0x7; 0 = A/D is
//                                                              filling 0x0-0x7 find
//                                                              data in 0x8-0xF)
//      6 ................. UNIMPLEMENTED ........... 0 ..... X
//      5 ........ SMPI<3> .. SAMP-CONV SEQ/INTERRUPT . 1 ..... Interrupts at the completion

//      4 ........ SMPI<2>                            0       of the conversion sequence
//      3 ........ SMPI<1>                            0       for each 8th sample/convert
//                                                              sequence
//      2 ........ SMPI<0>                            0
//      1 ........ BUFM ..... BUFFER MODE SELECT ..... 0 ..... Buffer configured as one 16-
//                                                              word buffer
//      0 ........ ALTS ..... ALTERNATE INPUT SAMPLE .. 0 ..... Always use MUX A input
//                                                              multiplexer settings
```

```
//-----------------------------------------------------------------
//      ADCON3:         A/D     CONTROL REGISTER 3              ADCON3 = 0x1539
//-----------------------------------------------------------------
//      BIT         NAME            FUNCTION                VALUE       DESCRIPTION
//-----------------------------------------------------------------
//      15 ................. UNIMPLEMENTED ........... 0 ...... X
//      14 ................. UNIMPLEMENTED ........... 0 ...... X
//      13 ................. UNIMPLEMENTED ........... 0 ...... X
//      12 ....... SAMC<4> .. AUTO SAMPLE TIME ........ 1 ...... SAMC = 21Tad
//      11 ........SAMC<3>                             0
//      10 ........SAMC<2>                             1
//      9 .........SAMC<1>                             0
//      8 .........SAMC<0>                             1
//      7 ........ ADRC ..... A/D CONVERSION CLK SOURCE 0 ...... Clock derived from system clk
//      6 ................. UNIMPLEMENTED ........... 0 ...... X
//      5 ........ ADCS<5> .. A/D CONVERSION CLOCK .... 1 ...... TCY/2(ADCS<5:0> + 1) = 29TCY
//      4 ........ ADCS<4>                             1
//      3 ........ ADCS<3>                             1
//      2 ........ ADCS<2>                             0
//      1 ........ ADCS<1>                             0
//      0 ........ ADCS<0>                             1
//-----------------------------------------------------------------
//      ADCHS:          A/D     INPUT SELECT REGISTER           ADCHS = 0x000B
//-----------------------------------------------------------------
//      BIT         NAME            FUNCTION                VALUE       DESCRIPTION
//-----------------------------------------------------------------
//      15 ................. UNIMPLEMENTED ........... 0 ...... X
//      14 ................. UNIMPLEMENTED ........... 0 ...... X
//      13 ................. UNIMPLEMENTED ........... 0 ...... X
//      12 ....... CH0NB .... CH0- INPUT SELECT MUXB .. 0 ...... NOT USED
//      11 ....... CH0NB<3> . CH0+ INPUT SELECT MUXB .. 0 ...... NOT USED
//      10 ....... CH0NB<2>                            0
//      9 ........ CH0NB<1>                            0
//      8 ........ CH0NB<0>                            0
//      7 ................. UNIMPLEMENTED ........... 0 ...... X
//      6 ................. UNIMPLEMENTED ........... 0 ...... X
//      5 ................. UNIMPLEMENTED ........... 0 ...... X
//      4 ........ CH0NA .... CH0- INPUT SELECT MUXA .. 0 ...... NOT USED
//      3 ........ CH0NA<3> . CH0+ INPUT SELECT MUXA .. 1 ...... Channel 0 positive input is
//                                                              AN11
//      2 ........ CH0NA<2>                            0
//      1 ........ CH0NA<1>                            1
//      0 ........ CH0NA<0>                            1
//-----------------------------------------------------------------
//      ADPCFC:         A/D     PORT CONFIGURATION REGISTER         ADPCFC = 0xF7FF
//-----------------------------------------------------------------
//      BIT         NAME            FUNCTION                VALUE       DESCRIPTION
//-----------------------------------------------------------------
//      15 ....... PCFC<15> . ANALOG INPUT PIN CONFIG . 1 ...... X
//      14 ....... PCFG<14>                            1 ...... X
//      13 ....... PCFG<13>                            1 ...... X
//      12 ....... PCFG<12>                            1 ...... X
//      11 ....... PCFG<11>                            0 ...... MIC_ADC
//      10 ....... PCFG<10>                            1 ...... X
//      9  ....... PCFG<9>                             1 ...... X
//      8 ........ PCFG<8>                             1 ...... X
//      7 ........ PCFG<7>                             1 ...... X
//      6 ........ PCFG<6>                             1 ...... X
//      5 ........ PCFG<5>                             1 ...... X
//      4 ........ PCFG<4>                             1 ...... X
//      3 ........ PCFG<3>                             1 ...... X
//      2 ........ PCFG<2>                             1 ...... X
//      1 ........ PCFG<1>                             1 ...... PGC
//      0 ........ PCFG<0>                             1 ...... PGD
//-----------------------------------------------------------------
//      ADCSSL:         A/D     INPUT SCAN SELECT REGISTER          ADCSSL = 0x0000
//-----------------------------------------------------------------
//      BIT         NAME            FUNCTION                VALUE       DESCRIPTION
//-----------------------------------------------------------------
//      15 ....... CSSL<15> . A/D INPUT PIN SCAN SELECT 0 ...... X
//      14 ....... CSSL<14>                            0 ...... X
//      13 ....... CSSL<13>                            0 ...... X
//      12 ....... CSSL<12>                            0 ...... X
//      11 ....... CSSL<11>                            0 ...... MIC_ADC
//      10 ....... CSSL<10>                            0 ...... X
//      9  ....... CSSL<9>                             0 ...... X
//      8 ........ CSSL<8>                             0 ...... X
```

```
//      7 ........ CSSL<7>                                  0 ...... X
//      6 ........ CSSL<6>                                  0 ...... X
//      5 ........ CSSL<5>                                  0 ...... X
//      4 ........ CSSL<4>                                  0 ...... X
//      3 ........ CSSL<3>                                  0 ...... X
//      2 ........ CSSL<2>                                  0 ...... X
//      1 ........ CSSL<1>                                  0 ...... PGC
//      0 ........ CSSL<0>                                  0 ...... PGD
//-------------------------------------------------------------------

void            InitADC12(void)
{
        ADCON1 = 0x03E0;
        ADCON2 = 0x0020;
        ADCON3 = 0x1539;

        ADCHS  = 0x000B;
        ADPCFG = 0xF7FF;
        ADCSSL = 0x0000;

        IFS0bits.ADIF  =        0;              //      Clear ADC interrupt has occurred flag
        IEC0bits.ADIE  =        0;              //      Disable ADC interrupts
}

//===================================================================
//      PUBLIC FUNCTION:        InitUART1
//===================================================================
//      FUNCTION CALL:          InitUART1();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//-------------------------------------------------------------------
//      DESCRIPTION:            Initialises the UART1 module on the dsPIC30F6014A.
//                              The value of all the registers are laid out below
//-------------------------------------------------------------------
//      BAUD RATE:              57600 bps
//      DATA SIZE:              8 Bits
//      PARITY:                 No parity
//      STOP BIT:               1 stop bit
//-------------------------------------------------------------------
//      U1MODE:         UART1 MODE REGISTER             U1MODE = 0x8000
//-------------------------------------------------------------------
//      BIT             NAME            FUNCTION        VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ....... UARTEN ... UART ENABLE BIT ......... 1 ...... UART is ENABLED
//      14 ..................... UNIMPLEMENTED ........... 0 ...... X
//      13 ....... USIDL .... STOP IN IDLE MODE ....... 0 ...... Operates in IDLE mode
//      12 ..................... UNIMPLEMENTED ........... 0 ...... X
//      11 ..................... RESERVED ................ 0 ...... X
//      10 ....... ALTIO .... ALTERNATE I/O SELECTION.. 0 ...... UART communicates using U1TX
//                                                                  and U1RX I/O pins
//      9 ..................... RESERVED ................ 0 ...... X
//      8 ..................... RESERVED ................ 0 ...... X
//      7 ........ WAKE ..... WAKE-UP ON START BIT .... 0 ...... Wake-up disabled
//      6 ........ LPBACK ... LOOPBACK MODE SELECT .... 0 ...... Loopback mode disabled
//      5 ........ ABAUD .... AUTO BAUD ENABLE ........ 0 ...... Input to capture module from
//                                                                  ICx pin
//      4 ..................... UNIMPLEMENTED ........... 0 ...... X
//      3 ..................... UNIMPLEMENTED ........... 0 ...... X
//      2 ........ PDSEL<1> . PARITY AND DATA SELECTION 0 ...... 8-bit data
//      1 ........ PDSEL<0>                             0         No parity
//      0 ........ STSEL .... STOP SELECTION .......... 0 ...... 1 Stop bit
//-------------------------------------------------------------------
//      U1STA:          UART1 STATUS AND CONTROL REGISTER           U1STA = 0x0000
//-------------------------------------------------------------------
//      BIT             NAME            FUNCTION        VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ....... UTXISEL .. TRANSMISSION INTERRUPT .. 0 ...... Interrupt when a character is
//                                                                  transferred to shift register
//      14 ..................... UNIMPLEMENTED ........... 0 ...... X
//      13 ..................... UNIMPLEMENTED ........... 0 ...... X
//      12 ..................... UNIMPLEMENTED ........... 0 ...... X
//      11 ....... UTXBRK ... TRANSMIT BREAK ........... 0 ...... U1TX pin operates normally
//      10 ....... UTXEN .... TRANSMIT ENABLE ......... 0 ...... UART transmitter NOT ENABLED
//      9 ........ UTXBF .... TRANSMIT BUFFER FULL .... 0 ...... (READ ONLY)
//      8 ........ TRMT ..... TRANSMIT SHIFT REG EMPTY  0 ...... (READ ONLY)
//      7 ........ URXISEL<1> RECEIVE INTERRUPT MODE .. 0 ...... Interrupt flag set when
//                                                                  character is received
```

```
//      6 ........ URXISEL<0>                              0
//      5 ........ ADDEN .... ADDRESS CHARCTER DETECT . 0 ...... Address detect mode disabled
//      4 ........ RIDLE .... RECEIVER IDLE ........... 0 ...... (READ ONLY)
//      3 ........ PERR ..... PARITY ERROR STATUS ..... 0 ...... (READ ONLY)
//      2 ........ FERR ..... FRAMING ERROR STAUS ..... 0 ...... (READ ONLY)
//      1 ........ OERR ..... RX BUFFER OVERRUN ERROR . 0 ...... (READ ONLY)
//      0 ........ URXDA .... RX BUFFER DATA AVAILABLE. 0 ...... (READ ONLY)
//-----------------------------------------------------------------------
void    InitUART1(void)
(
        U1MODE  =       0x8000;
        U1STA   =       0x0000;

        U1BRG   = ((FCY/16)/BAUD)-1;          //      Sets the baud rate to 57600 baud
)
//=======================================================================
//      PUBLIC FUNCTION:        InitSPI1
//=======================================================================
//      FUNCTION CALL:          InitSPI1();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//-----------------------------------------------------------------------
//      DESCRIPTION:            Initialises the SPI1 module on the dsPIC30F6014A
//                              The values of all the registers are laid out below
//      FSCLK:                  FCY/(PRIMARY PRESCALE x SECONDARY PRESCALE)
//                              1.67MHz
//-----------------------------------------------------------------------
//      SPI1CON:        SPI1 CONTROL REGISTER          SPI1CON = 0x0020
//-----------------------------------------------------------------------
//      BIT         NAME            FUNCTION            VALUE       DESCRIPTION
//-----------------------------------------------------------------------
//      15 ..................... UNIMPLEMENTED ........... 0 ...... X
//      14 ....... FRMEN .... FRAMED SPI SUPPORT ...... 0 ...... Framed SPI support DISABLED
//      13 ....... SPIFSD ... FRAME SYNC PUL DIRECTION  0 ...... Frame sync pulse output
//                                                               (Master)
//      12 ..................... UNIMPLEMENTED ........... 0 ...... X
//      11 ....... DISSDO ... DISABLE SDO1 PIN ........ 0 ...... SDO1 pin is controlled by the
//                                                                module
//      10 ....... MODE16 ... WORD/BYTE COMMS ......... 0 ...... Communication is 8 bits wide
//      9 ........ SMP ...... SPI DATA INPUT SAMP PHASE 0 ...... Input data sampled at middle
//                                                                of data output time
//      8 ........ CKE ...... SPI CLOCK EDGE SELECT ... 0 ...... Serial output data changes on
//                                                               transition from IDLE CLK to
//                                                               ACTIVE CLK (SPIMODE0)
//      7 ........ SSEN ..... SLAVE SELECT ENABLE ..... 0 ...... SS pin not used by module.
//      6 ........ CKP ...... SPI CLOCK POLARITY SELECT 0 ...... IDLE state for CLK is a HIGH
//      5 ........ MSTEN .... MASTER MODE ENABLE ...... 1 ...... Master mode
//      4 ........ SPRE<2> .. SECONDARY PRESCALE ...... 0 ...... Secondary Prescale 8:1
//      3 ........ SPRE<1>                              0
//      2 ........ SPRE<0>                              0
//      1 ........ PPRE<1>... PRIMARY PRESCALE ........ 0 ...... Primary Prescale 64:1
//      0 ........ PPRE<0>                              0
//-----------------------------------------------------------------------
//      SPI1STAT:       SPI STATUS AND CONTROL REGISTER       SPI1STAT = 0x8000
//-----------------------------------------------------------------------
//      BIT             NAME            FUNCTION                  VALUE       DESCRIPTION
//-----------------------------------------------------------------------
//      15 ....... SPIEN .... SPI ENABLE ............... 1 ...... SPI1 is DISABLED
//      14 ..................... UNIMPLEMENTED ........... 0 ...... X
//      13 ....... SPISIDL .. STOP IN IDLE MODE ....... 0 ...... Operates in IDLE mode
//      12 ..................... UNIMPLEMENTED ........... 0 ...... X
//      11 ..................... UNIMPLEMENTED ........... 0 ...... X
//      10 ..................... UNIMPLEMENTED ........... 0 ...... X
//      9 ...................... UNIMPLEMENTED ........... 0 ...... X
//      8 ...................... UNIMPLEMENTED ........... 0 ...... X
//      7 ...................... UNIMPLEMENTED ........... 0 ...... X
//      6 ........ SPIROV ... RECEIVE OVERFLOW FLAG ... 0 ...... No overflow has occured(READ
//                                                               ONLY)
//      5 ...................... UNIMPLEMENTED ........... 0 ...... X
//      4 ...................... UNIMPLEMENTED ........... 0 ...... X
//      3 ...................... UNIMPLEMENTED ........... 0 ...... X
//      2 ...................... UNIMPLEMENTED ........... 0 ...... X
//      1 ........ SPITBF ... SPI TRANSMIT BUFFER FULL  0 ...... Transmit started, SPI1TXB is
//                                                                EMPTY (READ ONLY)
//      0 ........ SPIRBF ... SPI RECEIVE BUFFER FULL.. 0 ...... Receive is not complete,
//                                                                SPI1RXB is EMPTY (READ ONLY)
//-----------------------------------------------------------------------
```

```
void    InitSPI1(void)
{
        CS      =       1;                      //      Disable AT45DB321B
        RESET   =       1;                      //      Normal Operation
        WP      =       1;                      //      Can write to Dataflash chip

        SPI1CON =       0x0020;
        SPI1STAT|=      0x8000;                 //      Enable device and leave status of other
                                                //      bits
        SPI1STAT&=      0xFFBF;                 //      SPIROV flag = 0; leaves all other bits
                                                //      the same

        IFS0bits.SPI1IF = 0;                    //      Clears SPI interrupt has occurred flag
        IEC0bits.SPI1IE = 0;                    //      Disables SPI interrupts
}

//=====================================================================
//      PUBLIC FUNCTION:        SendReceiveByte
//=====================================================================
//      FUNCTION CALL: SendReceiveByte(data);
//      ARGUMENTS:      unsigned char data
//      RETURNS:        unsigned char byte_received
//---------------------------------------------------------------------
//      DESCRIPTION:    Sends a 8bit byte to the AT45DB321B 32Mbit Data
//                      flash chip via the SPI data interface and returns
//                      the received byte.
//---------------------------------------------------------------------

unsigned char  SendReceiveByte(unsigned char data)
{

        SPI1BUF = data;                         //      Load 1 data byte into SPI1BUF -> SPI1TXB
                                                //      -> SPI1SR
        SPI1STAT &= 0xA003;                     //      Leaves SPIEN, SPITBF and SPIRBF bits same

        while (SPI1STATbits.SPITBF)             //      Wait until transfer is complete
        {
                /* NULL STATEMENT */;
        }
        while (!SPI1STATbits.SPIRBF)            //      Wait until receive is complete
        {
                /* NULL STATEMENT */;
        }

        return SPI1BUF;                         //      Returns the received byte

}
```

```
//=====================================================================
//      PUBLIC FUNCTION:       Erase
//=====================================================================
//      FUNCTION CALL: Erase();
//      ARGUMENTS:     None
//      RETURNS:       Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:   Erases entire data flash, block by block by making
//                     all values 1
//
//      ERASE INSTRUCTION FORMAT:
//                     CCCC CCCC RRAA AAAA AAAX XXXX XXXX XXXX
//                     8bit command, 2bits reserved, 9bit address, 13 don't care bits
//
//      MEMORY SIZE:   32 MbitS
//      BYTES/PAGE:    528
//      PAGES/BLOCK:   8
//      NO OF PAGES:   8192
//      NO OF BLOCKS:  1024
//---------------------------------------------------------------------

void    Erase(void)
{
        ERASE_LED = 1;                                  //      Turn erase led ON

        static unsigned int block_counter = 0;          //      Number of blocks erased counter

        mem_is_clear = 1;                               //      Set main memory is empty flag
        mem_is_full  = 0;                               //      Clear main memory is full flag

        while(block_counter < 1024)                     //      Do until all the pages have been
                                                        //      erased
        {
                CS = 0;                                 //      ENABLE AT45DB321B Dataflash chip

                SendReceiveByte(CMD_ERASE_BLOCK);       //      Send erase block command
                SendReceiveByte((char)(block_counter >> 3)); //      Puts address in correct
                                                        //      erase instruction
                                                        //      format
                SendReceiveByte((char)(block_counter << 5));
                SendReceiveByte(0x00);                  //      Don't care bits

                CS = 1;                                 //      DISABLE AT45DB321B Dataflash chip

                block_counter++;                        //      Increment block counter

                while(!READY_BUSY)
                {
                        /* NULL STATEMENT */;          //      Wait until the chip has completed
                                                        //      the erase
                }
        }

        ERASE_LED = 0;                                  //      Turn erase led OFF
}
```

```
//=====================================================================
//      PUBLIC FUNCTION:       Record
//=====================================================================
//      FUNCTION CALL: Record();
//      ARGUMENTS:     None
//      RETURNS:       Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:   Records data from one of the inputs to the ADC to
//                     the AT45DB321C Dataflash.
//---------------------------------------------------------------------

void    Record(void)
{
        RECORD_LED = 1;                         //      Turn ON the Record led


        int i = 0;                              //      Initialise Counter

        unsigned int    *ADCbufferPtr;          //      Pointer to ADC buffer
        unsigned int    *ADCvaluePtr;           //      Pointer to ADC value

        unsigned char   ADCValue[8];
        unsigned int    RMS_value;


        ADCON1bits.ADON = 1;                    //      Turn the ADC Module ON

        while(!memory_full)                     //      Do while there is space in memory
        {

                RMS_value = 0;                  //      Reset RMS value
                ADCbufferPtr = &ADCBUF0;        //      Point to the first value in the ADC
                                                //      buffer
                ADCvaluePtr = &ADCValue
                IFS0bits.ADIF = 0;              //      Clear 8 samples have been converted flag

                ADCON1bits.ASAM = 1;            //      Sampling begins immediately after last
                                                //      conversion ends

                while(!IFS0bits.ADIF)           //      Wait for conversion to finish
                {
                        /* NULL STATEMENT */;
                }

                ADCON1bits.ASAM = 0;            //      Sampling begins when SAMP bit is set

                for (i=0; i<ADC_BUFFERSIZE; i++)
                {
                        *ADCvaluePtr++ = *ADCbufferPtr++ - OFFSET;   //      Subtract offset
                                                                     //      from each sample
                }

                RMS_value = RMSCalcultaion(ADCvaluePtr, ADC_BUFFERSIZE);

                WritetoDataflash(RMS_value);
        }

        RECORD_LED = 0;                         //      Switch OFF the Record led
}
```

```
//=====================================================================
//      PUBLIC FUNCTION:       WritetoDataflash
//=====================================================================
//      FUNCTION CALL: WritetoDataflash(data);
//      ARGUMENTS:     unsigned char data
//      RETURNS:       Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:   Writes data to the AT45DB321B 32Mbit Data Flash Chip's
//                     main memory via buffer 1
//
//      WRITE TO BUFFER 1 COMMAND FORMAT:
//                     CCCC CCCC XXXX XXXX XXXX XXAA AAAA AAAA DDDD DDDD
//                     8bit command, 14bits dont care, 10bit address, 8bits of data
//
//      WRITE BUFFER 1 TO MAIN MEMORY COMMAND FORMAT:
//                     CCCC CCCC RAAA AAAA AAAA AAXX XXXX XXXX
//                     8bit command, 1bit reserved, 13bits address, 10bits don't care
//---------------------------------------------------------------------

void    WritetoDataflash(unsigned char data)
{

        static unsigned int buffer_counter;   //      Position in buffer 1 counter
        static unsigned int page_counter;     //      Page position counter

        if (mem_is_clear && !mem_is_full)     //      If the chip is clear, reset buffer and
                                              //      page counters
        {
                buffer_counter = 0;
                page_counter   = 0;

                mem_is_clear   = 0;           //      Chip is no longer clear
        }

        while(!READY_BUSY)                    //      Check to see if chip is busy
        {
                /* NULL STATEMENT */;
        }

//      WRITES DATA TO BUFFER 1

        CS = 0;                               //      ENABLE AT45DB321B Dataflash chip

        SendReceiveByte(CMD_WRITE_BUFFER1);           //      Send write to Buffer 1 command
        SendReceiveByte(0x00);                        //      Don't care bits
        SendReceiveByte((char)(buffer_counter >> 8));//      Don't care bits and 1st 2 bits of
                                                      //      address
        SendReceiveByte((char)(buffer_counter));      //      Buffer address
        SendReceiveByte(data);                        //      Writes the data to buffer 1

        CS = 1;                               //      DISABLE AT45DB321B Dataflash chip

        buffer_counter++;                     //      Increments the buffer counter

//      WRITES DATA FROM BUFFER 1 TO MAIN MEMORY

        if(buffer_counter > 528)              //      If buffer is full
        {
                buffer_counter = 0;           //      Reset buffer_counter

                if(page_counter < 8192)       //      If main memory is not full then write to
                                              //      appropriate page in main memory
                {
                CS = 0;                 //      ENABLE AT45DB321B Dataflash chip

                SendReceiveByte(CMD_PROG_BUF1toMAINMEM_WOERASE);     //      Send write to data
                                                                     //      from buffer 1 to
                                                                     //      main memory without
                                                                     //      erase command
                SendReceiveByte((char)(page_counter >> 6));          //      1 Reserved bit and
                                                                     //      7 Page  Address
                SendReceiveByte((char)(page_counter << 2));          //      6 Page Address bits
                                                                     //      and2 don't care
                                                                     //      bits
                SendReceiveByte(0x00);                               //      Don't care bits

                CS = 1;                       //      DISABLE AT45DB321B Dataflash chip
```

```
                page_counter++;                 //      Increment the page counter
            }

            else if (page_counter == 8192)
            {
                mem_is_full = 1;                //      Main memory is full
            }
        }
}

//=====================================================================
//      PUBLIC FUNCTION:        Send
//=====================================================================
//      FUNCTION CALL: Send();
//      ARGUMENTS:      None
//      RETURNS:        Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:    Reads data from the data flash and outputs via the
//                      UART1 Module to a PC.
//---------------------------------------------------------------------

void    Send(void)
{
        SEND_LED = 1;                                   //      Turn send led ON

        unsigned int    page_counter    = 0;            //      Page position counter
        unsigned char   active_buffer   = BUFFER1;      //      Active buffer
        unsigned int    buffer_address  = 0;            //      Buffer address
        unsigned int    buffer_length   = 528;          //      Length of the buffer

        unsigned char   buffer[528] = {0};              //      Buffer array
        unsigned char   *bufferPtr;                     //      Pointer for buffer array

        bufferPtr = &buffer[0];                         //      Point to first element in
                                                        //      buffer array

        PagetoBuffer(active_buffer, page_counter);      //      Transfer first page to
                                                        //      buffer1

        while(!READY_BUSY)                              //      Check to see if chip is
                                                        //      busy
        {
                /* NULL STATEMENT */;
        }

        while(page_counter < 8192)                      //      While there are still
                                                        //      pages to be read
        {
                page_counter++;                         //      Increment the page counter

                PagetoBuffer(active_buffer, page_counter);  //  Transfer next page to
                                                            //  buffer
                ReadfromBuffer(active_buffer,buffer_address,*bufferPtr,MEM_BUFFERSIZE);
                                                        //      Send that buffer via the
                                                        //      UART to the PC

                SendtoUART1(buffer);                    //      Send Buffer to UART1
                                                        //      module to be transmitted
                                                        //       to PC

                if      (active_buffer == BUFFER1)      //      If the active buffer is
                                                        //      buffer1
                {
                        active_buffer++;                //      now activate buffer2
                }
                else                                    //      Else if the active buffer
                                                        //      is buffer2
                {
                        active_buffer--;                //      activate buffer1
                }
        }

        SEND_LED = 0;                                   //      Turn send led OFF
}
```

```
//=====================================================================
//      PUBLIC FUNCTION:       PagetoBuffer
//=====================================================================
//      FUNCTION CALL: PagetoBuffer(active_buffer, page_counter);
//      ARGUMENTS:     unsigned char active buffer, unsigned int page_counter
//      RETURNS:       Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:   Transfers 1 page of data from the main memory
//                     to the active buffer of the AT45DB321B Dataflash
//
//      MAIN MEMORY PAGE TO BUFFER TRANSFER COMMAND FORMAT:
//                     CCCC CCCC RAAA AAAA AAAA AAXX XXXX XXXX
//                     8bit command, 1bit reserved, 13bits address, 10bits don't care
//---------------------------------------------------------------------

void    PagetoBuffer(unsigned char active_buffer, unsigned int page_counter)
{
        while(!READY_BUSY)                              //      Check to see if chip is busy
        (
                /* NULL STATEMENT */;
        )

        CS = 0;                                         //      ENABLE AT45DB321B Dataflash chip

        if      (active_buffer == BUFFER1)              //      If the active buffer is buffer1
        {
                SendReceiveByte(CMD_TRANSFER_MAINMEMPAGEtoBUF1);     //      Transfer data from
                                                                     //      main memory to
                                                                     //      buffer1
        )
        else                                                         //      else if the active
                                                                     //      buffer is buffer2
        {
                SendReceiveByte(CMD_TRANSFER_MAINMEMPAGEtoBUF2);     //      Transfer data from
                                                                     //      main memory to
                                                                     //      buffer2
        )

                SendReceiveByte((char)(page_counter >> 6));          //      1 Reserved bit and
                                                                     //      7 Page Address
                                                                     //      bits
                SendReceiveByte((char)(page_counter << 2));          //      6 Page Address bits
                                                                     //      and 2 don't care
                                                                     //      bits
                SendReceiveByte(0x00);                               //      Don't care bits

        CS = 1;                                         //      DISABLE AT45DB321B Dataflash chip

)
```

```
//=====================================================================
//      PUBLIC FUNCTION:        ReadfromBuffer
//=====================================================================
//      FUNCTION CALL: ReadfromBuffer(buffer, address, *data, offset, length);
//      ARGUMENTS:      unsigned char buffer
//                      unsigned int address
//                      unsigned char *data
//                      unsigned int length
//      RETURNS:        Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:    Transfers the contents of the active buffer
//                      of the AT45DB321B Dataflash via the UART module
//                      to a PC running Hyperterminal
//
//      READ FROM ACTIVE BUFFER COMMAND FORMAT:
//                      CCCC CCCC XXXX XXXX XXXX XXAA AAAA AAAA XXXX XXXX

//                      8bit command, 14bits dont care, 10bit address, 8bits don't care
//---------------------------------------------------------------------

void    ReadfromBuffer(unsigned char buffer, unsigned int address,
                       unsigned char *returned_data, unsigned int length)
{
        unsigned int i;

        while(!READY_BUSY)                      //      Check to see if chip is busy
        {
                /* NULL STATEMENT */;
        }

//      READS DATA FROM BUFFER 1

        CS = 0;                                 //      ENABLE AT45DB321B Dataflash chip

        if      (buffer = BUFFER1)              //      If Buffer is 1
        {
                SendReceiveByte(CMD_READ_BUFFER1);  //      Then send read from
                                                    //      buffer 1 command
        }
        else
        {
                SendReceiveByte(CMD_READ_BUFFER2);  //      Else send read from
                                                    //      buffer 2 command
        }

                SendReceiveByte(0x00);          //      Don't care bits
                SendReceiveByte(address>> 8); //      Don't care bits and 1st 2 bits of address
                SendReceiveByte(address);       //      Buffer address
                SendReceiveByte(0x00);          //      Don't care bits

        for (i=0;i<length;i++)
        {
                *returned_data++ = SendReceiveByte(0x00);
        }

        CS = 1;                                 //      DISABLE AT45DB321B Dataflash chip
}
```

```
//=====================================================================
//      PUBLIC FUNCTION:       SendtoUART1
//=====================================================================
//      FUNCTION CALL: SendtoUART1(data[]);
//      ARGUMENTS:     unsigned char data[]
//      RETURNS:       Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:   This routine transmits the data acquired from the
//                     AT45DB321C via UART1 to the PC running HyperTerminal
//---------------------------------------------------------------------

void    SendtoUART1(data[])
{
        int i;                                          //      Initialise counter
        int outdata1;                                   //      First byte of integer
        int outdata2;                                   //      Second byte of integer
        int outdata;                                    //      Data integer

        for (i=0; i < MEM_BUFFERSIZE;i+2)
        {
                outdata1 = (int)data[i];                //      Most significant char
                outdata2 = (int)data[i+1];              //      Least significant char

                outdata1 = outdata << 8;                //      Shift position in integer
                outdata = outdata1 + outdata2;          //      Combined data character

                asciiVoltsConv(outdata);                //      Convert to a voltage value
                                                        //      in ascii

                OutData[0] = units;                     //      Set outdata values
                OutData[2] = tenths;
                OutData[3] = hundredths;
                OutData[4] = thousandths;

                txCount         = 0;                    //      Sets counter to 0
                IFS0bits.U1TXIF = 0;                    //      Clear TX flag
                IEC0bits.U1TXIE = 1;                    //      Enable interrupt

                U1STA           = 0x0000;               //      Clear all status bits
                U1STAbits.UTXEN = 1;                    //      Start transmission
        }
}
```

```
//=====================================================================
//      PUBLIC FUNCTION:        RMSCalculation
//=====================================================================
//      FUNCTION CALL: RMSCalculation(*dataPtr,N);
//      ARGUMENTS:     unsigned int *dataPtr,
//                     int N
//      RETURNS:       unsigned long Result_RMS_calulation
//---------------------------------------------------------------------
//      DESCRIPTION:   This routine takes an array of length N and converts
//                     it into a single RMS value using the equation below
//
//                                        ____________
//                                       |┌  N        ┐
//                     RMSValue  =       ||  Σ [(ui)²] |
//                                       || i=0        |
//                                       ||  ________  |
//                                      √|└      N     ┘
//---------------------------------------------------------------------

unsigned int RMSCalculation(unsigned int *dataPtr, int N)
{
        int i = 0;                                      //      Initialise counter

        unsigned int  squared_value   = 0;              //      Stores the Squared value
        unsigned int  sum             = 0;              //      Stores sum
        unsigned int  average         = 0;              //      Stores average
        signed int    RMS_valueQ15    = 0;              //      Stores RMS value in Q1.15 format
        unsigned int  RMS_value       = 0;              //      Stores RMS value in unsigned
                                                        //      integer format

        for (i=0;i<N;i++)                               //      Convert all values in array
        {
                squared_value =  pow(*dataPtr++,2);     //      Square each value
                sum += squared_value;                   //      and sum them all together
        }

        average = sum/N;                                //      Divide the sum by the number of
                                                        //      values
                                                        //      to find the average
        RMS_valueQ15 = (int)sqrt(average);              //      Square root the average to find
                                                        //      the RMS value

        RMS_value = Q15toInteger(RMS_valueQ15);         //      Convert from Q1.15 to integer

        return RMS_value;
}
```

```
//=====================================================================
//      PUBLIC FUNCTION:       Q15toInteger
//=====================================================================
//      FUNCTION CALL: Q15toInteger(const signed int value);
//      ARGUMENTS:     Integer value
//      RETURNS:       Unsigned int converted_result
//---------------------------------------------------------------------
//      DESCRIPTION:   This function converts a 16-bit number in Q1.15
//                     data format to an unsigned integer
//
//                     Signed Fractional (Q1.15)
//                     (DOUT = sddd dddd dddd 0000)
//
//                     Unsigned Integer
//                     (DOUT = 0000 dddd dddd dddd)
//---------------------------------------------------------------------

unsigned int Q15toInteger(signed int value)
{
        value = value>>4;

        unsigned int value_bytes12 = 0;             //      Stores bytes 1 and 2
        unsigned int value_byte3   = 0;             //      Stores byte 3
        unsigned int integer_value = 0;             //      Stores the converted integer value

        value_bytes12 = value & 0x00FF;             //      Select bytes 1 and 2

        if (value >= 0x0800)                        //      If the sign bit is 1, CLEAR it
        {
                value_byte3 = value >> 8;
                value_byte3 -= 0x0008;
                value_byte3 = value_byte3 << 8;

                integer_value = value_bytes12 + value_byte3;
        }
        else if (value < 0x0800)                    //      If the sign bit is 0, SET it
                {
                value_byte3 = value >> 8;
                value_byte3 += 0x0008;
                value_byte3 = value_byte3 << 8;

                integer_value = value_bytes12 + value_byte3;
        }

        return integer_value;                       //      Return the converted value
}
```

```
//=====================================================================
//      PUBLIC FUNCTION:        asciiVoltsConv
//=====================================================================
//      FUNCTION CALL: asciiVoltsConv(const unsigned int value);
//      ARGUMENTS:      Integer value
//      RETURNS:        Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:    This function converts a 12-bit binary word to a
//                      BCD ascii decimal value
//
//                      Resolution of this function is 1mV
//---------------------------------------------------------------------
void    asciiVoltsConv(const unsigned int value)
{
    int unitsInt          = 0;                          //      Initialise local variables
    int tenthsInt         = 0;
    int hundredthsInt     = 0;
    int thousandthsInt    = 0;

    int n               = value;

    if (n >= 0x04D9)                                    //      If there are 1Vs
    {
        unitsInt    = n/0x04D9;
        units       = asciiConv(unitsInt);              //      convert to an ascii char
        n           = n%0x04D9;           //  the remainder is ready for more conversion
    }
    if (n >= 0x007C)                                    //      If there are 100mVs
    {
        tenthsInt     = n/0x007C;
        tenths        = asciiConv(tenthsInt);           //      convert to an ascii char
        n             = n%0x007C;         //  the remainder is ready for more conversion
    }
    if (n >= 0x000C)                                    //      If there are 10mVs
    {
        hundredthsInt  = n/0x000C;
        hundredths     = asciiConv(hundredthsInt);      //      convert to an ascii char
        n              = n%0x000C;        // the remainder is ready for more conversion
    }
    if (value >= 0x0001)                                //      If there are 1mVs
    {
        thousandthsInt = n/0x0001;
        thousandths    = asciiConv(thousandsInt);       //      convert to an ascii char
        n              = n%0x0001;        // the remainder is ready for more conversion
    }
}
//=====================================================================
//      PUBLIC FUNCTION:        asciiConv
//=====================================================================
//      FUNCTION CALL:          asciiConv(const unsigned int value);
//      ARGUMENTS:              Integer Value
//      RETURNS:                Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:            This function converts a 16-bit value to an 8-bit
//                              ascii character
//---------------------------------------------------------------------
char    asciiConv(const unsigned int value)
{
        char no;

        no = (char)(value & 0x0F);                      //      Only select bits 0-3

        if      (no     > 9)                            //      If number is > 9
        {
                no = no + 0x37;                         //  Number will be (A = 0x41, B = 0x42, C
                                                        //      = 0x43 etc )
        }
        else
        {
                no = no + 0x30;                         //      Number will be 0 - 9
        }

        return no;
}
```

```
//=====================================================================
//      INTERRUPT SERVICE ROUTINE (ISR):      _U1TXInterrupt
//=====================================================================
//      DESCRIPTION:    This ISR clears the interrupt has occurred flag and
//                      then transmits a word of data
//---------------------------------------------------------------------
void    __attribute__((__interrupt__)) _U1TXInterrupt(void)
{
        IFS0bits.U1TXIF = 0;                            //      Clear transmit interrupt has
                                                        //      occurred flag
        U1TXREG = (int)OutData[txCount++];              //      Write a single word to UART
        if(txCount == TX_BUFFERSIZE)
        {
                IEC0bits.U1TXIE = 0;                    //      Disable transmit interrupts since
                                                        //      all buffer values are transmitted
        }
}
//=====================================================================
```

# Appendix L

## Software Testing Code

### *LED Testing – TestLEDs.c*

```
//=====================================================================
//                      LED TESTING
//=====================================================================
//      BY:             ROBYN VERRINDER
//      DATE:           01 MAY 2006
//=====================================================================
//      PROCESSOR:      dsPIC30F6014A
//      BOARD:          VIBRATION DATA LOGGING PROTOTYPE BOARD
//      IDE:            MPLAB IDE v7.30
//      COMPILER:       C30 C COMPILER
//=====================================================================
//      DESCRIPTION:    This program tests the LEDs on the Vibration Data
//                      Logging Prototype Board. These LEDs are used as
//                      indicator lights.
//
//                      All LEDs are turned ON for the duration of the program
//=====================================================================
//      LEDS
//=====================================================================
//  PIN CONNECTIONS: (LEDS to dsPIC30F6014A)
//              LED1            =       RC1 (PIN 2)
//              LED2            =       RC2 (PIN 3)
//              LED3            =       RC3 (PIN 4)
//              LED4            =       RC4 (PIN 5)
//              RECORD_LED      =       RG6 (PIN 6)
//              SEND_LED        =       RG7 (PIN 7)
//              ERASE_LED       =       RG8 (PIN 8)
//=====================================================================
#include                "p30f6014a.h"           //      (Sinha, P., 2005) [64].
#include                "dsPIC30F6014APins.h"
//---------------------------------------------------------------------
//      dsPIC30F6014A CONFIGURATION SETTINGS
//---------------------------------------------------------------------
//      CLOCK SWITCHING AND MONITOR ..... Sw Disabled, Mon Disabled
//      OSCILLATOR ...................... XT w/PLL 8x
//      WATCHDOG TIMER .................. Disabled
//      MASTER CLEAR ENABLE ............. Enabled
//      POR TIMER VALUE ................. 64ms
//      GENERAL CODE SEGMENT CODE PROTECT ..... Disabled
//---------------------------------------------------------------------
_FOSC(CSW_FSCM_OFF & XT_PLL8);
_FWDT(WDT_OFF);
_FBORPOR(MCLR_EN & PWRT_64);
_FGS(CODE_PROT_OFF);
//---------------------------------------------------------------------
//      FUNCTION DEFINITIONS
//---------------------------------------------------------------------

void    InitPorts(void);                        //      Initialises the LED ports

//---------------------------------------------------------------------
//      MAIN FUNCTION
//---------------------------------------------------------------------

int     main(void)
{
        InitPorts();                            //      Setup ports for LEDs
```

```
while(1)                                //      Turn all LEDs on indefinitely
        {

                LED1            =       1;
                LED2            =       1;
                LED3            =       1;
                LED4            =       1;

                RECORD_LED      =       1;
                SEND_LED        =       1;
                ERASE_LED       =       1;

        }                               //      End of while loop
}                                       //      End of main function

//=====================================================================
//      PUBLIC FUNCTION:        InitPorts
//=====================================================================
//      FUNCTION CALL:          InitPorts();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:            Initialises the LED ports as OUTPUTs and sets Ports LOW
//---------------------------------------------------------------------
void    InitPorts(void)
{

        TRISC   =       0x0000;                 //      Port C I/O direction
        TRISG   =       0x0000;                 //      Port G I/O direction

        PORTC   =       0x0000;                 //      Port C pins levels set to LOW
        PORTG   =       0x0000;                 //      Port C pins levels set to LOW
}
//=====================================================================
```

## *UART1 Testing – TestUART1.c*

```
//=====================================================================
//                      UART1 MODULE TESTING
//=====================================================================
//      BY:             ROBYN VERRINDER
//      DATE:           01 MAY 2006
//=====================================================================
//      PROCESSOR:      dsPIC30F6014A
//      BOARD:          VIBRATION DATA LOGGING PROTOTYPE BOARD
//      IDE:            MPLAB IDE v7.30
//      COMPILER:       C30 C COMPILER
//=====================================================================
//      DESCRIPTION:    This program tests the Universal Asynchronous
//                      Receiver Transmitter (UART) module on the dsPIC30F6014A
//                      microcontroller chip.
//
//                      This module is the primary serial communication link,
//                      on the Vibration Data Logging Prototype Board, to
//                      the PC. HyperTerminal is used to view the data. Jumpers on
//                      JP8 must be connected.
//
//                      The output of the microcontroller is connected to
//                      an RS-232 converter (U10) the pin descriptions are
//                      given below. [49].
//
//                      Register descriptions are as in dsPIC30F Family
//                      Reference Manual Section 19: UART[63].
//=====================================================================
#include              "p30f6014a.h"                 //      (Sinha, P., 2005) [64].
#include              "dsPIC30F6014APins.h"
#include              "common.h"
//---------------------------------------------------------------------
//      dsPIC30F6014A CONFIGURATION SETTINGS
//---------------------------------------------------------------------
//      CLOCK SWITCHING AND MONITOR ..... Sw Disabled, Mon Disabled
//      OSCILLATOR ...................... XT w/PLL 8x
//      WATCHDOG TIMER .................. Disabled
//      MASTER CLEAR ENABLE ............. Enabled
//      POR TIMER VALUE ................. 64ms
//      GENERAL CODE SEGMENT CODE PROTECT ..... Disabled
//---------------------------------------------------------------------
_FOSC(CSW_FSCM_OFF & XT_PLL8);
_FWDT(WDT_OFF);
_FBORPOR(MCLR_EN & PWRT_64);
_FGS(CODE_PROT_OFF);
//---------------------------------------------------------------------
//      FUNCTION DEFINITIONS
//---------------------------------------------------------------------

void    InitUART1(void);                        //      Initialises the UART1 Module

void    __attribute__((__interrupt__)) _U1TXInterrupt(void);//      UART1 transmit ISR

//---------------------------------------------------------------------
//      GLOBAL VARIABLES
//---------------------------------------------------------------------

unsigned char  data;                            //      Value to be transmitted

//---------------------------------------------------------------------
//      MAIN FUNCTION
//---------------------------------------------------------------------

int     main(void)
{
                data    =       'A';            //      Set the value to be transmitted

                InitUART1();                    //      Initialises the UART1 module

                IFS0bits.U1TXIF = 0;            //      Clears TX flag
                IEC0bits.U1TXIE = 1;            //      Enables UART1 interrupts
```

```
while (1)
        {
                U1TXREG = data;                 //      Write a single word to UART

                U1STAbits.UTXEN = 1;            //      Start transmission

                while (!U1STAbits.TRMT)         //      Wait until the TSR is empty
                {
                        /* NULL STATEMENT */;
                }
        }
}                                               //      End of main
//=============================================================================
//      PUBLIC FUNCTION:        InitUART1
//=============================================================================
//      FUNCTION CALL:          InitUART1();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//-----------------------------------------------------------------
//      DESCRIPTION:            Initialises the UART1 module on the dsPIC30F6014A.
//                              The value of all the registers are laid out below
//-----------------------------------------------------------------
//      BAUD RATE:              9600 bps
//      DATA SIZE:              8 Bits
//      PARITY:                 No parity
//      STOP BIT:               1 stop bit
//      FLOW CONTROL:           None
///----------------------------------------------------------------
//      U1MODE:         UART1 MODE REGISTER             U1MODE = 0x8000
//-----------------------------------------------------------------
//      BIT          NAME            FUNCTION           VALUE      DESCRIPTION
//-----------------------------------------------------------------
//      15 ....... UARTEN ... UART ENABLE BIT ......... 1 ...... UART is ENABLED
//      14 ..................... UNIMPLEMENTED ........... 0 ...... X
//      13 ....... USIDL .... STOP IN IDLE MODE ....... 0 ...... Operates in IDLE mode
//      12 ..................... UNIMPLEMENTED ........... 0 ...... X
//      11 ..................... RESERVED ................ 0 ...... X
//      10 ....... ALTIO .... ALTERNATE I/O SELECTION.. 0 ...... UART communicates using U1TX
//                                                                  and U1RX I/O pins
//      9 ...................... RESERVED ................ 0 ...... X
//      8 ...................... RESERVED ................ 0 ...... X
//      7 ........ WAKE ..... WAKE-UP ON START BIT .... 0 ...... Wake-up disabled
//      6 ........ LPBACK ... LOOPBACK MODE SELECT .... 0 ...... Loopback mode disabled
//      5 ........ ABAUD .... AUTO BAUD ENABLE ......... 0 ...... Input to capture module from
//                                                                  ICx pin
//      4 ...................... UNIMPLEMENTED ........... 0 ...... X
//      3 ...................... UNIMPLEMENTED ........... 0 ...... X
//      2 ........ PDSEL<1> . PARITY AND DATA SELECTION 0 ...... 8-bit data
//      1 ........ PDSEL<0>                             0         No parity
//      0 ........ STSEL .... STOP SELECTION .......... 0 ...... 1 Stop bit

//-----------------------------------------------------------------
//      U1STA:          UART1 STATUS AND CONTROL REGISTER            U1STA = 0x8400
//-----------------------------------------------------------------
//      BIT       NAME             FUNCTION              VALUE      DESCRIPTION
//-----------------------------------------------------------------
//      15 ....... UTXISEL .. TRANSMISSION INTERRUPT .. 1 ...... Interrupt when transmit
//                                                                  buffer is EMPTY
//      14 ..................... UNIMPLEMENTED ........... 0 ...... X
//      13 ..................... UNIMPLEMENTED ........... 0 ...... X
//      12 ..................... UNIMPLEMENTED ........... 0 ...... X
//      11 ....... UTXBRK ... TRANSMIT BREAK .......... 0 ...... U1TX pin operates normally
//      10 ....... UTXEN .... TRANSMIT ENABLE ......... 1 ...... UART transmitter ENABLED
//      9 ........ UTXBF .... TRANSMIT BUFFER FULL .... 0 ...... (READ ONLY)
//      8 ........ TRMT ..... TRANSMIT SHIFT REG EMPTY  0 ...... (READ ONLY)
//      7 ........ URXISEL<1> RECEIVE INTERRUPT MODE .. 0 ...... Interrupt flag set when
//                                                                  character is received
//      6 ........ URXISEL<0>                           0
//      5 ........ ADDEN .... ADDRESS CHARCTER DETECT . 0 ...... Address detect mode disabled
//      4 ........ RIDLE .... RECEIVER IDLE ........... 0 ...... (READ ONLY)
//      3 ........ PERR ..... PARITY ERROR STATUS ..... 0 ...... (READ ONLY)
//      2 ........ FERR ..... FRAMING ERROR STAUS ..... 0 ...... (READ ONLY)
//      1 ........ OERR ..... RX BUFFER OVERRUN ERROR . 0 ...... (READ ONLY)
//      0 ........ URXDA .... RX BUFFER DATA AVAILABLE. 0 ...... (READ ONLY)
//-----------------------------------------------------------------
```

```
void    InitUART1(void)
{
        U1MODE  =       0x8000;
        U1STA   =       0x8400;
        U1BRG   = ((FCY/16)/BAUD)-1;          //      Sets the baud rate to 9600 baud

        INTCON1bits.NSTDIS = 1;               //      Disable nested interrupts
}
//=============================================================================
//      INTERRUPT SERVICE ROUTINE (ISR):      _U1TXInterrupt
//=============================================================================
//      DESCRIPTION:          Clears the transmit interrupt has occurred flag
//-----------------------------------------------------------------------------
void    __attribute__((__interrupt__)) _U1TXInterrupt(void)
{
        IFS0bits.U1TXIF = 0;          //      Clear transmit interrupt has occurred flag
}
//=============================================================================
```

## *ADC Testing – TestADCPotentiometer.c*

```
//=====================================================================
//              12 BIT ADC TESTING - DC Voltage Input
//=====================================================================
//      BY:             ROBYN VERRINDER
//                      Parts of this program were adapted from the
//                      Example Code for UART module on the dsPICDEM Low
//                      Cost Starter Board,    ADC12_UART.c (D'Souza, S., 2003) [68].
//      DATE:           06 MAY 2006
//=====================================================================
//      PROCESSOR:      dsPIC30F6014A
//      BOARD:          VIBRATION DATA LOGGING PROTOTYPE BOARD
//      IDE:            MPLAB IDE v7.30
//      COMPILER:       C30 C COMPILER
//=====================================================================
//      DESCRIPTION:    This program tests the functioning of the 12-bit Analogue
//                      to Digital Converter (ADC) Module of the dsPIC30F6014A.
//                      on the Vibration Data Logging Prototype Board.
//
//                      A DC voltage (0-3.3V) is produced by a potentiometer
//                      connected across the AVDD supply.
//
//                      The data is outputted via the UART1 module to a PC.
//                      HyperTerminal is used to view the data.
//
//                      This code was mainly adapted from the example code for the UART module
//                      on the dsPICDEM Low Cost Starter Board [68].
//                      The InitUART, DelayNmSec and ISR for the UART module
//                      were written by Microchip. The InitADC12 and
//                      SendtoUART routines were adapted from their code, while the
//                      asciiVoltsConv and asciiConv routines were written by Robyn Verrinder.
//
//                      Register descriptions are as in dsPIC30F Family Reference Manual
//                      Section 18: 12 bit A/D Convertor and Section 19: UART [63]
//=====================================================================
//      XTAL FREQ (Fxtal):      10MHz
//      MIPS:                   20MIPS  (XT PLLx8 Mode)
//      TCY:                    50ns
//      Tconv:                  22.4us  (14Tad)
//      Tad:                    1.6us   (calculated value using ADCS<5:0> bits)
//      ADC Resolution (bits): 12-bits
//      ADC Resolution (mV):    0.806 mV (for a 3.3V supply)
//=====================================================================
#include                "p30f6014a.h"                   //      (Sinha, P., 2005) [64].
#include                "dsPIC30F6014APins.h"
#include                "common.h"
//---------------------------------------------------------------------
//      dsPIC30F6014A CONFIGURATION SETTINGS
//---------------------------------------------------------------------
//      CLOCK SWITCHING AND MONITOR ..... Sw Disabled, Mon Disabled
//      OSCILLATOR ...................... XT w/PLL 8x
//      WATCHDOG TIMER .................. Disabled
//      MASTER CLEAR ENABLE ............. Enabled
//      POR TIMER VALUE ................. 64ms
//      GENERAL CODE SEGMENT CODE PROTECT ..... Disabled
//---------------------------------------------------------------------
_FOSC(CSW_FSCM_OFF & XT_PLL8);
_FWDT(WDT_OFF);
_FBORPOR(MCLR_EN & PWRT_64);
_FGS(CODE_PROT_OFF);
//---------------------------------------------------------------------
//      FUNCTION DEFINITIONS
//---------------------------------------------------------------------
void    InitADC12(void);
void    InitUART1(void);

void    SendtoUART();
void    DelayNmSec(unsigned int N);

void    asciiDecConv(const unsigned int value);
char    asciiConv(unsigned int value);
void    __attribute__((__interrupt__)) _U1TXInterrupt(void);
```

```
//-------------------------------------------------------------------
//      ASCII BUFFERS FOR SERIAL DATA
//-------------------------------------------------------------------
//      BYTE         NAME          VALUE
//-------------------------------------------------------------------
//      7 ........ DATA<0> ........ 0x30 ........ Units
//      6 ........ '.' ............ 0x2E ........ Decimal Point
//      5 ........ DATA<2> ........ 0x31 ........ Tenths
//      4 ........ DATA<3> ........ 0x32 ........ Hundredths
//      3 ........ DATA<4> ........ 0x33 ........ Thousands
//      2 ........ 'V' ............ 0x56 ........ V (Volts)
//      1 ........ LF ............. 0x0A ........ New Line
//      0 ........ CR ............. 0x0D ........ Character Return
//-------------------------------------------------------------------

unsigned char  OutData[TX_BUFFERSIZE] = {0x30,0x2E,0x31,0x32,0x56,0x0A,0x0D};

//-------------------------------------------------------------------
//      GLOBAL VARIABLES
//-------------------------------------------------------------------

unsigned char                 txCount;

int                           ADCValue;

unsigned char                 units;
unsigned char                 tenths;
unsigned char                 hundredths;

//-------------------------------------------------------------------
//      MAIN FUNCTION
//-------------------------------------------------------------------

int     main(void)
{
        InitUART1();
        InitADC12();

        while(1)                                    //      Do forever
        {
                ADCON1bits.SAMP  = 1;               //      Start sampling
                DelayNmSec(20);                     //      for 20ms
                ADCON1bits.SAMP  = 0;               //      Stop sampling

                while(!IFS0bits.ADIF)               //      Wait for conversion to finish
                {
                        /* NULL STATEMENT */;
                }

                ADCValue = ADCBUF0;                 //      Load 16-bit ADCValue with the
                                                    //      result of the conversion
                SendtoUART();                       //      Send the data via UART1 to PC
                                                    //      running HyperTerminal
        }

}
//===================================================================
//      PUBLIC FUNCTION:        InitUART1       (D'Souza, S., 2003) [68].
//===================================================================
//      FUNCTION CALL:          InitUART1();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//-------------------------------------------------------------------
//      DESCRIPTION:            Initialises the UART1 module on the dsPIC30F6014A.
//                              The value of all the registers are laid out below
//-------------------------------------------------------------------
//      BAUD RATE:              57600 bps
//      DATA SIZE:              8 Bits
//      PARITY:                 No parity
//      STOP BIT:               1 stop bit
//      FLOW CONTROL:           None
```

```
//-------------------------------------------------------------------
//      U1MODE:         UART1 MODE REGISTER             U1MODE = 0x8000
//-------------------------------------------------------------------
//      BIT          NAME            FUNCTION          VALUE      DESCRIPTION
//-------------------------------------------------------------------
//      15 ....... UARTEN ... UART ENABLE BIT ......... 1 ...... UART is ENABLED
//      14 ..................... UNIMPLEMENTED ........... 0 ...... X
//      13 ....... USIDL .... STOP IN IDLE MODE ....... 0 ...... Operates in IDLE mode
//      12 ..................... UNIMPLEMENTED ........... 0 ...... X
//      11 ..................... RESERVED ................ 0 ...... X
//      10 ....... ALTIO .... ALTERNATE I/O SELECTION.. 0 ...... UART communicates using U1TX
//                                                                  and U1RX I/O pins
//      9 ..................... RESERVED ................ 0 ...... X
//      8 ..................... RESERVED ................ 0 ...... X
//      7 ........ WAKE ..... WAKE-UP ON START BIT .... 0 ...... Wake-up disabled
//      6 ........ LPBACK ... LOOPBACK MODE SELECT .... 0 ...... Loopback mode disabled
//      5 ........ ABAUD .... AUTO BAUD ENABLE ........ 0 ...... Input to capture module from
//                                                                  ICx pin
//      4 ..................... UNIMPLEMENTED ........... 0 ...... X
//      3 ..................... UNIMPLEMENTED ........... 0 ...... X
//      2 ........ PDSEL<1> . PARITY AND DATA SELECTION 0 ...... 8-bit data
//      1 ........ PDSEL<0>                             0         No parity
//      0 ........ STSEL .... STOP SELECTION .......... 0 ...... 1 Stop bit
//-------------------------------------------------------------------
//      U1STA:          UART1 STATUS AND CONTROL REGISTER            U1STA = 0x0000
//-------------------------------------------------------------------
//      BIT          NAME            FUNCTION          VALUE      DESCRIPTION
//-------------------------------------------------------------------
//      15 ....... UTXISEL .. TRANSMISSION INTERRUPT .. 0 ...... Interrupt when a character is
//                                                                  transferred to shift register
//      14 ..................... UNIMPLEMENTED ........... 0 ...... X
//      13 ..................... UNIMPLEMENTED ........... 0 ...... X
//      12 ..................... UNIMPLEMENTED ........... 0 ...... X
//      11 ....... UTXBRK ... TRANSMIT BREAK .......... 0 ...... U1TX pin operates normally
//      10 ....... UTXEN .... TRANSMIT ENABLE ......... 0 ...... UART transmitter NOT ENABLED
//      9 ........ UTXBF .... TRANSMIT BUFFER FULL .... 0 ...... (READ ONLY)
//      8 ........ TRMT ..... TRANSMIT SHIFT REG EMPTY  0 ...... (READ ONLY)
//      7 ........ URXISEL<1> RECEIVE INTERRUPT MODE .. 0 ...... Interrupt flag set when
//                                                                  character is received
//      6 ........ URXISEL<0>                           0
//      5 ........ ADDEN .... ADDRESS CHARCTER DETECT . 0 ...... Address detect mode disabled
//      4 ........ RIDLE .... RECEIVER IDLE ........... 0 ...... (READ ONLY)
//      3 ........ PERR ..... PARITY ERROR STATUS ..... 0 ...... (READ ONLY)
//      2 ........ FERR ..... FRAMING ERROR STAUS ..... 0 ...... (READ ONLY)
//      1 ........ OERR ..... RX BUFFER OVERRUN ERROR . 0 ...... (READ ONLY)
//      0 ........ URXDA .... RX BUFFER DATA AVAILABLE. 0 ...... (READ ONLY)
//-------------------------------------------------------------------

void    InitUART1(void)
{
        U1MODE  =       0x8000;
        U1STA   =       0x0000;

        U1BRG   = ((FCY/16)/BAUD)-1;          //      Sets the baud rate

        INTCON1bits.NSTDIS = 1;               //      Disable nested interrupts
}

//===================================================================
//      PUBLIC FUNCTION:        InitADC12
//===================================================================
//      FUNCTION CALL:          InitADC12();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//-------------------------------------------------------------------
//      DESCRIPTION:            Initialises the 12 bit Analogue to Digital Converter
//                              All the registers are set as laid out below
//-------------------------------------------------------------------
//      ADCON1:         A/D     CONTROL REGISTER 1              ADCON1 = 0x0000
//-------------------------------------------------------------------
//      BIT       NAME            FUNCTION              VALUE      DESCRIPTION
//-------------------------------------------------------------------
//      15 ...... ADON ...... A/D OPERATING MODE ....... 0 ...... ADC is OFF
//      14 ..................... UNIMPLEMENTED ........... 0 ...... X
//      13 ...... ADSIDL .... STOP IN IDLE MODE ........ 0 ...... Operates in IDLE mode
//      12 ..................... UNIMPLEMENTED ........... 0 ...... X
//      11 ..................... UNIMPLEMENTED ........... 0 ...... X
```

```
//      10 ................. UNIMPLEMENTED ........... 0 ...... X
//      9 ........ FORM<1> .. DATA OUTPUT FORMAT ..... 0 ...... Unsigned Integer
//      8 ........ FORM<0>                             0        (DOUT = 0000 dddd dddd dddd)
//      7 ........ SSRC<2> .. CONVERSION TRIGGER SOURCE  0 ...... Clearing SAMP bit ends
//      6 ........ SSRC<1>                             0        sampling and starts
//      5 ........ SSRC<0>                             0        conversion
//      4 ................. UNIMPLEMENTED ...........  0 ...... X
//      3 ................. UNIMPLEMENTED ........... 0 ...... X
//      2 ........ ASAM ..... A/D SAMPLE AUTO START ...  0 ...... Sampling begins when SAMP bit
//                                                               is set
//      1 ........ SAMP ..... A/D SAMPLE ENABLE .......  0 ...... SAMP bit is auto set by ASAM
//                                                               setting
//      0 ........ DONE ..... A/D CONVERSION STATUS ...  0 ...... Automatically cleared and set
//                                                               by MCU
//-------------------------------------------------------------------
//      ADCON2:         A/D     CONTROL REGISTER 2            ADCON2 = 0x0000
//-------------------------------------------------------------------
//      BIT       NAME            FUNCTION                VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ...... VCFG<2> ... VOLTAGE REF CONFIG ...... 0 ...... A/D VREFH = AVDD
//      14 ...... VCFG<1>                               0        A/D VREFL = AVSS
//      13 ...... VCFG<0>                               0
//      12 ................. RESERVED ................ 0
//      11 ................. UNIMPLEMENTED ........... 0 ...... X
//      10 ....... CSCNA .... SCAN INPUTS ............ 0 ...... Do not scan inputs
//      9 ................. UNIMPLEMENTED ........... 0 ...... X
//      8 ................. UNIMPLEMENTED ........... 0 ...... X
//      7 ........ BUFS ..... BUFFER FILL STATUS ..... 0 ...... Auto set (1 = A/D is filling
//                                                               0x8-0xF find data
//                                                               in 0x0-0x7; 0 = A/D is
//                                                               filling 0x0-0x7 find
//                                                               data in 0x8-0xF)
//      6 ................. UNIMPLEMENTED ........... 0 ...... X
//      5 ........ SMPI<3> .. SAMP-CONV SEQ/INTERRUPT . 0 ...... Interrupts at the completion
//      4 ........ SMPI<2>                              0        of the conversion sequence
//      3 ........ SMPI<1>                              0        for each sample/convert
//                                                               sequence
//      2 ........ SMPI<0>                              0
//      1 ........ BUFM ..... BUFFER MODE SELECT ..... 0 ...... Buffer configured as one 16-
//                                                               word buffer
//      0 ........ ALTS ..... ALTERNATE INPUT SAMPLE .. 0 ...... Always use MUX A input
//                                                               multiplexer settings
//-------------------------------------------------------------------
//      ADCON3:         A/D     CONTROL REGISTER 3            ADCON3 = 0x0002
//-------------------------------------------------------------------
//      BIT       NAME            FUNCTION                VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ................. UNIMPLEMENTED ........... 0 ...... X
//      14 ................. UNIMPLEMENTED ........... 0 ...... X
//      13 ................. UNIMPLEMENTED ........... 0 ...... X
//      12 ....... SAMC<4> .. AUTO SAMPLE TIME ........ 0 ...... SAMC = 0Tad
//      11 ........SAMC<3>                              0
//      10 ........SAMC<2>                              0
//      9 .........SAMC<1>                              0
//      8 .........SAMC<0>                              0
//      7 ........ ADRC ..... A/D CONVERSION CLK SOURCE 0 ...... Clock derived from system clk
//      6 ................. UNIMPLEMENTED ........... 0 ...... X
//      5 ........ ADCS<5> .. A/D CONVERSION CLOCK .... 0 ...... TCY/2(ADCS<5:0> + 1) = 1.5TCY
//      4 ........ ADCS<4>                              0
//      3 ........ ADCS<3>                              0
//      2 ........ ADCS<2>                              0
//      1 ........ ADCS<1>                              1
//      0 ........ ADCS<0>                              0
//-------------------------------------------------------------------
//      ADCHS:          A/D     INPUT SELECT REGISTER         ADCHS = 0x000B
//-------------------------------------------------------------------
//      BIT         NAME            FUNCTION                VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ................. UNIMPLEMENTED ........... 0 ...... X
//      14 ................. UNIMPLEMENTED ........... 0 ...... X
//      13 ................. UNIMPLEMENTED ........... 0 ...... X
//      12 ....... CH0NB .... CH0- INPUT SELECT MUXB .. 0 ...... NOT USED
//      11 ....... CH0NB<3> . CH0+ INPUT SELECT MUXB .. 0 ...... NOT USED
//      10 ....... CH0NB<2>                             0
//      9 ........ CH0NB<1>                             0
//      8 ........ CH0NB<0>                             0
//      7 ................. UNIMPLEMENTED ........... 0 ...... X
```

```
//      6 ..................... UNIMPLEMENTED ........... 0 ...... X
//      5 ..................... UNIMPLEMENTED ........... 0 ...... X
//      4 ........ CH0NA .... CH0- INPUT SELECT MUXA .. 0 ...... NOT USED
//      3 ........ CH0NA<3> . CH0+ INPUT SELECT MUXA .. 1 ...... Channel 0 positive input is
//                                                                AN11
//      2 ........ CH0NA<2>                             0
//      1 ........ CH0NA<1>                             1
//      0 ........ CH0NA<0>                             1
//-------------------------------------------------------------------
//      ADPCFC:         A/D     PORT CONFIGURATION REGISTER          ADPCFC = 0xF7FF
//-------------------------------------------------------------------
//      BIT         NAME            FUNCTION            VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ....... PCFC<15> . ANALOG INPUT PIN CONFIG . 1 ...... X
//      14 ....... PCFG<14>                            1 ...... X
//      13 ....... PCFG<13>                            1 ...... X
//      12 ....... PCFG<12>                            1 ...... X
//      11 ....... PCFG<11>                            0 ...... MIC_ADC
//      10 ....... PCFG<10>                            1 ...... X
//      9  ....... PCFG<9>                             1 ...... X
//      8 ........ PCFG<8>                             1 ...... X
//      7 ........ PCFG<7>                             1 ...... X
//      6 ........ PCFG<6>                             1 ...... X
//      5 ........ PCFG<5>                             1 ...... X
//      4 ........ PCFG<4>                             1 ...... X
//      3 ........ PCFG<3>                             1 ...... X
//      2 ........ PCFG<2>                             1 ...... X
//      1 ........ PCFG<1>                             1 ...... PGC
//      0 ........ PCFG<0>                             1 ...... PGD
//-------------------------------------------------------------------
//      ADCSSL:         A/D     INPUT SCAN SELECT REGISTER           ADCSSL = 0x0000
//-------------------------------------------------------------------
//      BIT         NAME            FUNCTION            VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ....... CSSL<15> . A/D INPUT PIN SCAN SELECT 0 ...... X
//      14 ....... CSSL<14>                            0 ...... X
//      13 ....... CSSL<13>                            0 ...... X
//      12 ....... CSSL<12>                            0 ...... X
//      11 ....... CSSL<11>                            0 ...... MIC_ADC
//      10 ....... CSSL<10>                            0 ...... X
//      9  ....... CSSL<9>                             0 ...... X
//      8 ........ CSSL<8>                             0 ...... X
//      7 ........ CSSL<7>                             0 ...... X
//      6 ........ CSSL<6>                             0 ...... X
//      5 ........ CSSL<5>                             0 ...... X
//      4 ........ CSSL<4>                             0 ...... X
//      3 ........ CSSL<3>                             0 ...... X
//      2 ........ CSSL<2>                             0 ...... X
//      1 ........ CSSL<1>                             0 ...... PGC
//      0 ........ CSSL<0>                             0 ...... PGD
//-------------------------------------------------------------------

void            InitADC12(void)
{
        ADCON1 = 0x0000;
        ADCON2 = 0x0000;
        ADCON3 = 0x0002;

        ADCHS  = 0x000B;
        ADPCFG = 0xF7FF;
        ADCSSL = 0x0000;

        ADCON1bits.ADON = 1;            //      Turn ADC ON
}

//===================================================================
//      PUBLIC FUNCTION:        SendtoUART      (Adapted from D'Souza, S., 2003) [68].
//===================================================================
//      FUNCTION CALL: SendtoUART(adc_value);
//      ARGUMENTS:      None
//      RETURNS:        Nothing
//-------------------------------------------------------------------
//      DESCRIPTION:    This routine transmits the data acquired from the ADC
//                      via UART1 to the PC running HyperTerminal
//-------------------------------------------------------------------
```

```
void    SendtoUART()
{
        while (!U1STAbits.TRMT)                         //      Wait until the TSR is empty
        {
                /* NULL STATEMENT */;
        }

        U1STAbits.UTXEN = 0;                            //      Disable transmission

        ADCValue = ADCValue/0x000C;                     //      Multiply by 100

        asciiDecConv(ADCValue);
        OutData[0] = units;
        OutData[2] = tenths;
        OutData[3] = hundredths;

        txCount         = 0;                            //      Sets counter to 0
        IFS0bits.U1TXIF = 0;                            //      Clear TX flag
        IEC0bits.U1TXIE = 1;                            //      Enable interrupt

        U1STA   = 0x0000;                               //      Clear all status bits
        U1STAbits.UTXEN = 1;                            //      Start transmission
}

//=====================================================================
//      PUBLIC FUNCTION:        asciiVoltsConv
//=====================================================================
//      FUNCTION CALL: asciiVoltsConv(const unsigned int value);
//      ARGUMENTS:      Integer value
//      RETURNS:        Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:    This function converts a 12-bit binary word to a
//                      BCD ascii decimal value
//
//                      Resolution of this function is 1mV
//---------------------------------------------------------------------
void    asciiVoltsConv(const unsigned int value)
{
    int unitsInt          = 0;                          //      Initialise local variables
    int tenthsInt         = 0;
    int hundredthsInt     = 0;
    int thousandthsInt    = 0;

    int n               = value;

    if (n >= 0x04D9)                                    //      If there are 1Vs
    {
        unitsInt    = n/0x04D9;
        units       = asciiConv(unitsInt);              //      convert to an ascii char
        n           = n%0x04D9;             //  the remainder is ready for more conversion
    }
    if (n >= 0x007C)                                    //      If there are 100mVs
    {
        tenthsInt   = n/0x007C;
        tenths      = asciiConv(tenthsInt);             //      convert to an ascii char
        n           = n%0x007C;           // the remainder is ready for more conversion
    }
    if (n >= 0x000C)                                    //      If there are 10mVs
    {
        hundredthsInt  = n/0x000C;
        hundredths     = asciiConv(hundredthsInt);      //      convert to an ascii char
        n              = n%0x000C;        // the remainder is ready for more conversion
    }
    if (value >= 0x0001)                                //      If there are 1mVs
    {
        thousandthsInt = n/0x0001;
        thousandths    = asciiConv(thousandsInt);       //      convert to an ascii char
        n              = n%0x0001;        // the remainder is ready for more conversion
    }
}
```

```
//=====================================================================
//      PUBLIC FUNCTION:        asciiConv
//=====================================================================
//      FUNCTION CALL:          asciiConv(const unsigned int value);
//      ARGUMENTS:              Integer Value
//      RETURNS:                Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:            This function converts a 16-bit value to an 8-bit
//                              ascii character
//---------------------------------------------------------------------
char    asciiConv(const unsigned int value)
{
        char no;

        no = (char)(value & 0x0F);              //      Only select bits 0-3

        if      (no     > 9)                    //      If number is > 9
        {
                no = no + 0x37;                 //      Number will be (A = 0x41, B = 0x42, C
                                                //      = 0x43 etc )
        }
        else
        {
                no = no + 0x30;                 //      Number will be 0 - 9
        }

        return no;
}

//=====================================================================
//      PUBLIC FUNCTION:        DelayNmSec      (D'Souza, S., 2003) [68].
//=====================================================================
//      FUNCTION CALL:          DelayNmSec(unsigned int N);
//      ARGUMENTS:              Number of millisecond N
//      RETURNS:                Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:    This function creates a 1ms to 65.5ms delay. For
//                      a 1ms delay N=1 for a 30ms delay N=30 etc
//---------------------------------------------------------------------
void DelayNmSec(unsigned int N)
{
         unsigned int j;                        //  Counter j

        while(N--)                              //  Do while until N = 0
        {
                for(j=0;j < MILLISEC;j++)       //  This creates a 1 mS delay
                {
                        /* NULL STATEMENT */;
                }
        }
}

//=====================================================================
//      INTERRUPT SERVICE ROUTINE (ISR):        _U1TXInterrupt (D'Souza, S., 2003) [68].
//=====================================================================
//      DESCRIPTION:    This ISR clears the interrupt has occured flag and
//                                      then transmits a word of data
//---------------------------------------------------------------------
void    __attribute__((__interrupt__)) _U1TXInterrupt(void)
{
        IFS0bits.U1TXIF = 0;                            //      Clear transmit interrupt has
                                                        //      occured flag

        U1TXREG = (int)OutData[txCount++];              //      Write a single word to UART

        if(txCount == TX_BUFFERSIZE)
        {
                IEC0bits.U1TXIE = 0;                    //      Disable transmit interrupts since
                                                        //      all buffer values are transmitted
        }

}
//=====================================================================
```

## *ADC Testing – TestACC.c*

```
//=====================================================================
//                      ACCELEROMETER TESTING
//=====================================================================
//      BY:             ROBYN VERRINDER
//                      Parts of this program were adapted from the
//                      Example Code for the dsPICDEM Low Cost Starter Board
//                      ADC12_UART.c (D'Souza, S., 2003) [68] and
//                      the example code CE001_ADC_DSP_lib_FILTER ADC.c
//                      (Vasuki, H., 2005) [69].
//      DATE:           06 MAY 2006
//=====================================================================
//      PROCESSOR:      dsPIC30F6014A
//      BOARD:          VIBRATION DATA LOGGING PROTOTYPE BOARD
//      IDE:            MPLAB IDE v7.30
//      COMPILER:       C30 C COMPILER
//=====================================================================
//      DESCRIPTION:    This program tests the functioning of both the
//                      12-bit Analog to Digital Convertor (ADC) Module of
//                      the dsPIC30F6014A and the ADXL202E Accelerometer.
//
//                      The data is outputed via the UART1 module to a PC.
//                      Hyperterminal is used to view the data.
//
//                      Register descriptions are as in dsPIC30F Family Reference Manual
//                      Section 12: Timers, Section 18: 12 bit A/D Convertor and Section 19:
//                      UART [63].
//=====================================================================
//      XTAL FREQ (Fxtal):      10MHz
//      MIPS:                   20MIPS  (XT PLLx8 Mode)
//      TCY:                    50ns
//      Tconv:                  20us    (this value is greater than the 10us minimum)
//      Tad:                    1.45us  (calculated value using ADCS<5:0> bits)
//=====================================================================
//      Fs              =       32kHz
//      SAMC<4:0>       =       10101   (8)
//      ADCS<5:0>       =       111001  (57)
//---------------------------------------------------------------------
#include                "p30f6014a.h"           //      (Sinha, P., 2005) [64].
#include                <math.h>                //      Standard C Math header file
#include                "dsPIC30F6014APins.h"
#include                "common.h"
//---------------------------------------------------------------------
//      dsPIC30F6014A CONFIGURATION SETTINGS
//---------------------------------------------------------------------
//      CLOCK SWITCHING AND MONITOR ..... Sw Disabled, Mon Disabled
//      OSCILLATOR ...................... XT w/PLL 8x
//      WATCHDOG TIMER .................. Disabled
//      MASTER CLEAR ENABLE ............. Enabled
//      POR TIMER VALUE ................. 64ms
//      GENERAL CODE SEGMENT CODE PROTECT ..... Disabled
//---------------------------------------------------------------------
_FOSC(CSW_FSCM_OFF & XT_PLL8);
_FWDT(WDT_OFF);
_FBORPOR(MCLR_EN & PWRT_64);
_FGS(CODE_PROT_OFF);
//---------------------------------------------------------------------
//      FUNCTION DEFINITIONS
//---------------------------------------------------------------------

void    InitADC12(void);
void    InitUART1(void);

unsigned long RMSCalculation(unsigned int *dataPtr, int N);

void    SendDatatoUART1(unsigned long data);

void    asciiDecConv(const unsigned int value);
char    asciiConv(unsigned int value);

void     __attribute__((__interrupt__)) _U1TXInterrupt(void);
```

```
//-------------------------------------------------------------------
//      ASCII BUFFERS FOR SERIAL DATA
//-------------------------------------------------------------------
//      BYTE         NAME          VALUE
//-------------------------------------------------------------------
//      8 ........ DATA<3> ........ 0x30 ........ Units
//      7 ........ '.' ............ 0x2E ........ Decimal Point
//      6 ........ DATA<2> ........ 0x31 ........ Tenths
//      5 ........ DATA<1> ........ 0x32 ........ Hundredths
//      4 ........ DATA<0> ........ 0x33 ........ Thousands
//      3 ........ SPACE .......... 0x20
//      2 ........ 'V' ............ 0x43
//      1 ........ LF ............. 0x0A
//      0 ........ CR ............. 0x0D
//-------------------------------------------------------------------

unsigned char  OutData[TX_BUFFERSIZE] = {0x30,0x2E,0x31,0x32,0x33,0x20,0x56,0x0A,0x0D};

//-------------------------------------------------------------------
//      GLOBAL VARIABLES
//-------------------------------------------------------------------
unsigned int   *ADCbufferPtr;                       //      Pointer to ADCbuff
unsigned int   *ADCvaluePtr;                        //      Pointer to ADC Value Buffer
unsigned int   *ADCvalueXPtr;                       //      Pointer to ADC XValue Buffer
unsigned int   *ADCvalueYPtr;                       //      Pointer to ADC YValue Buffer

unsigned char  txCount;

int            RMS_valueX;
int            RMS_valueY;
unsigned char  ADC_bcd   = 0;

unsigned char  units;
unsigned char  tenths;
unsigned char  hundredths;
unsigned char  thousandths;


unsigned char  ADCValue[16];
unsigned char  ADCXValue[8];
unsigned char  ADCYValue[8];

//-------------------------------------------------------------------
//      MAIN FUNCTION
//-------------------------------------------------------------------

int            main(void)
{
       InitUART1();
       InitADC12();

       int i = 0;                              //      Initalise Counter

       ADCON1bits.ADON = 1;                    //      Turn the ADC Module ON

       while(1)                                //      Do forever
       {

              RMSValueX = 0;                   //      Reset RMS X value
              RMSValueY = 0;                   //      Reset RMS Y value
              ADCbufferPtr = &ADCBUF0;         //      Point to 1st value of ADC buffer
              ADCvaluePtr = &ADCValue[0];      //      Point to 1st value of ADC value buffer
              ADCvalueXPtr = &ADCValueX[0];    //      Point to 1st value in the ADC X buffer
              ADCvalueXPtr = &ADCValueY[0];    //      Point to 1st value in the ADC Y buffer

              IFS0bits.ADIF = 0;               //      Clear 16 samples have been converted flag

              ADCON1bits.ASAM = 1;             //      Sampling begins immediately after last
                                               //      conversion ends

              while(!IFS0bits.ADIF)            //      Wait for conversion to finish
              {
                     /* NULL STATEMENT */;
              }

              ADCON1bits.ASAM = 0;             //      Sampling begins when SAMP bit is set
```

```
            for (i=0; i<ADC_BUFFERSIZE; i++)
            {
                  *ADCvaluePtr++ = *ADCbufferPtr++ - OFFSET;   //      Subtract the offset
                                                               //      from each sample
            }

            for (i=0; i<ADC_BUFFERSIZE; i++)
            {
                  if ((i%2) == 0)
                  {
                        ADCValueX[i] = ADCValue[i];            //      Samples from ACCX
                  }
                  else
                  {
                        ADCValueY[i] = ADCValue[i];            //      Samples from ACCY
                  }
            }


            RMSValueX = RMSCalcultaion(ADCvalueXPtr, ADC_BUFFERSIZE);
            RMSValueY = RMSCalcultaion(ADCvalueYPtr, ADC_BUFFERSIZE);

            SendDatatoUART1(RMSValueX);                        //      Send the data via UART1
            SendDatatoUART1(RMSValueY);                        //      Send the data via UART1
      }
}
//=====================================================================
//      PUBLIC FUNCTION:        InitADC12
//=====================================================================
//      FUNCTION CALL:          InitADC12();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:            Initialises the 12 bit Analogue to Digital Converter
//                              All the registers are set as laid out below
//---------------------------------------------------------------------
//      SAMPLING RATE:          32 kHz
//---------------------------------------------------------------------
//      ADCON1:         A/D     CONTROL REGISTER 1              ADCON1 = 0x03E0
//---------------------------------------------------------------------
//      BIT       NAME              FUNCTION                VALUE       DESCRIPTION
//---------------------------------------------------------------------
//      15 ..... ADON ..... A/D OPERATING MODE ....... 0 ..... ADC is OFF
//      14 ................ UNIMPLEMENTED ............ 0 ..... X
//      13 ..... ADSIDL .... STOP IN IDLE MODE ....... 0 ..... Operates in IDLE mode
//      12 ................ UNIMPLEMENTED ............ 0 ..... X
//      11 ................ UNIMPLEMENTED ............ 0 ..... X
//      10 ................ UNIMPLEMENTED ............ 0 ..... X
//      9 ........ FORM<1> .. DATA OUTPUT FORMAT ..... 1 ..... Signed Fractional
//      8 ........ FORM<0>                             1       (DOUT = sddd dddd dddd 0000)
//      7 ........ SSRC<2> .. CONVERSION TRIGGER SOURCE 1 ..... Internal counter ends
//      6 ........ SSRC<1>                             1       sampling and starts
//      5 ........ SSRC<0>                             1       conversion (Auto convert)
//      4 ................ UNIMPLEMENTED ............ 0 ..... X
//      3 ................ UNIMPLEMENTED ............ 0 ..... X
//      2 ........ ASAM ..... A/D SAMPLE AUTO START ... 0 ..... Sampling begins when SAMP bit
//                                                             is set
//      1 ........ SAMP ..... A/D SAMPLE ENABLE ....... 0 ..... SAMP bit is auto set by ASAM
//                                                             setting
//      0 ........ DONE ..... A/D CONVERSION STATUS ... 0 ..... Automatically cleared and set
//                                                             by MCU
//---------------------------------------------------------------------
//      ADCON2:         A/D     CONTROL REGISTER 2              ADCON2 = 0x003D
//---------------------------------------------------------------------
//      BIT       NAME              FUNCTION              VALUE       DESCRIPTION
//---------------------------------------------------------------------
//      15 ..... VCFG<2> ... VOLTAGE REF CONFIG ..... 0 ..... A/D VREFH = AVDD
//      14 ..... VCFG<1>                              0       A/D VREFL = AVSS
//      13 ..... VCFG<0>                              0
//      12 ................ RESERVED ................ 0
//      11 ................ UNIMPLEMENTED ........... 0 ..... X
//      10 ....... CSCNA .... SCAN INPUTS ............ 0 ..... Do not scan inputs
//      9 ................. UNIMPLEMENTED ........... 0 ..... X
//      8 ................. UNIMPLEMENTED ........... 0 ..... X
//      7 ........ BUFS ..... BUFFER FILL STATUS ..... 0 ..... Auto set (1 = A/D is filling
//                                                             0x8-0xF find data
//                                                             in 0x0-0x7; 0 = A/D is
```

```
//                                                                  filling 0x0-0x7 find
//                                                                  data in 0x8-0xF)
//      6 ..... UNIMPLEMENTED ..... 0 ..... X
//      5 ..... SMPI<3> .. SAMP-CONV SEQ/INTERRUPT . 1 ..... Interrupts at the completion
//      4 ..... SMPI<2>                              1       of the conversion sequence
//      3 ..... SMPI<1>                              1       for each 16th sample/convert
//                                                           sequence
//      2 ..... SMPI<0>                              1
//      1 ..... BUFM ..... BUFFER MODE SELECT ..... 0 ..... Buffer configured as one 16-
//                                                           word buffer
//      0 ..... ALTS ..... ALTERNATE INPUT SAMPLE .. 1 ..... MUX A = Sample 1
//                                                           MUX B = Sample2
//-------------------------------------------------------------------
//      ADCON3:         A/D     CONTROL REGISTER 3              ADCON3 = 0x0A39
//-------------------------------------------------------------------
//      BIT         NAME            FUNCTION                VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ..... UNIMPLEMENTED ..... 0 ..... X
//      14 ..... UNIMPLEMENTED ..... 0 ..... X
//      13 ..... UNIMPLEMENTED ..... 0 ..... X
//      12 ..... SAMC<4> .. AUTO SAMPLE TIME ..... 1 ..... SAMC = 8Tad
//      11 .....SAMC<3>                              0
//      10 .....SAMC<2>                              1
//      9 .....SAMC<1>                               0
//      8 .....SAMC<0>                               1
//      7 ..... ADRC ..... A/D CONVERSION CLK SOURCE 0 ..... Clock derived from system clk
//      6 ..... UNIMPLEMENTED ..... 0 ..... X
//      5 ..... ADCS<5> .. A/D CONVERSION CLOCK .... 1 ..... TCY/2(ADCS<5:0> + 1) = 29TCY
//      4 ..... ADCS<4>                              1
//      3 ..... ADCS<3>                              1
//      2 ..... ADCS<2>                              0
//      1 ..... ADCS<1>                              0
//      0 ..... ADCS<0>                              1
//-------------------------------------------------------------------
//      ADCHS:          A/D     INPUT SELECT REGISTER           ADCHS = 0x0A09
//-------------------------------------------------------------------
//      BIT         NAME            FUNCTION                VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ..... UNIMPLEMENTED ..... 0 ..... X
//      14 ..... UNIMPLEMENTED ..... 0 ..... X
//      13 ..... UNIMPLEMENTED ..... 0 ..... X
//      12 ..... CH0NB .... CH0- INPUT SELECT MUXB .. 0 ..... NOT USED
//      11 ..... CH0NB<3> . CH0+ INPUT SELECT MUXB .. 1 ..... Channel 0 positive input is
//      10 ..... CH0NB<2>                            0 ..... A10
//      9 ..... CH0NB<1>                             1
//      8 ..... CH0NB<0>                             0
//      7 ..... UNIMPLEMENTED ..... 0 ..... X
//      6 ..... UNIMPLEMENTED ..... 0 ..... X
//      5 ..... UNIMPLEMENTED ..... 0 ..... X
//      4 ..... CH0NA .... CH0- INPUT SELECT MUXA .. 0 ..... NOT USED
//      3 ..... CH0NA<3> . CH0+ INPUT SELECT MUXA .. 1 ..... Channel 0 positive input is
//      2 ..... CH0NA<2>                             0       AN9
//      1 ..... CH0NA<1>                             1
//      0 ..... CH0NA<0>                             1
//-------------------------------------------------------------------
//      ADPCFC:         A/D     PORT CONFIGURATION REGISTER         ADPCFC = 0xF9FF
//-------------------------------------------------------------------
//      BIT         NAME            FUNCTION            VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ..... PCFC<15> . ANALOG INPUT PIN CONFIG .  1 ..... X
//      14 ..... PCFG<14>                              1 ..... X
//      13 ..... PCFG<13>                              1 ..... X
//      12 ..... PCFG<12>                              1 ..... X
//      11 ..... PCFG<11>                              1 ..... X
//      10 ..... PCFG<10>                              0 ..... ACC_Y
//      9  ..... PCFG<9>                               0 ..... ACC_X
//      8 ..... PCFG<8>                                1 ..... X
//      7 ..... PCFG<7>                                1 ..... X
//      6 ..... PCFG<6>                                1 ..... X
//      5 ..... PCFG<5>                                1 ..... X
//      4 ..... PCFG<4>                                1 ..... X
//      3 ..... PCFG<3>                                1 ..... X
//      2 ..... PCFG<2>                                1 ..... X
//      1 ..... PCFG<1>                                1 ..... PGC
//      0 ..... PCFG<0>                                1 ..... PGD
```

```
//-------------------------------------------------------------------
//      ADCSSL:         A/D     INPUT SCAN SELECT REGISTER          ADCSSL = 0x0000
//-------------------------------------------------------------------
//      BIT         NAME            FUNCTION            VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ..... CSSL<15> . A/D INPUT PIN SCAN SELECT 0 ..... X
//      14 ..... CSSL<14>                             0 ..... X
//      13 ..... CSSL<13>                             0 ..... X
//      12 ..... CSSL<12>                             0 ..... X
//      11 ..... CSSL<11>                             0 ..... X
//      10 ..... CSSL<10>                             0 ..... ACC_Y
//      9  ..... CSSL<9>                              0 ..... ACC_X
//      8 ..... CSSL<8>                               0 ..... X
//      7 ..... CSSL<7>                               0 ..... X
//      6 ..... CSSL<6>                               0 ..... X
//      5 ..... CSSL<5>                               0 ..... X
//      4 ..... CSSL<4>                               0 ..... X
//      3 ..... CSSL<3>                               0 ..... X
//      2 ..... CSSL<2>                               0 ..... X
//      1 ..... CSSL<1>                               0 ..... PGC
//      0 ..... CSSL<0>                               0 ..... PGD
//-------------------------------------------------------------------

void            InitADC12(void)
{
        ADCON1 = 0x03E0;
        ADCON2 = 0x003D;
        ADCON3 = 0x0A39;

        ADCHS  = 0x0A09;
        ADPCFG = 0xF9FF;
        ADCSSL = 0x0000;

        IFS0bits.ADIF  =        0;              //      Clear ADC interrupt has occurred flag
        IEC0bits.ADIE  =        0;              //      Disable ADC interrupts
}

//===================================================================
//      PUBLIC FUNCTION:        InitUART1
//===================================================================
//      FUNCTION CALL:          InitUART1();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//-------------------------------------------------------------------
//      DESCRIPTION:            Initialises the UART1 module on the dsPIC30F6014A.
//                              The value of all the registers are laid out below
//-------------------------------------------------------------------
//      BAUD RATE:              57600 bps
//      DATA SIZE:              8 Bits
//      PARITY:                 No parity
//      STOP BIT:               1 stop bit
//-------------------------------------------------------------------
//      U1MODE:         UART1 MODE REGISTER             U1MODE = 0x8000
//-------------------------------------------------------------------
//      BIT             NAME            FUNCTION            VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ..... UARTEN ... UART ENABLE BIT ......... 1 ..... UART is ENABLED
//      14 ..... UNIMPLEMENTED ..... 0 ..... X
//      13 ..... USIDL .... STOP IN IDLE MODE ..... 0 ..... Operates in IDLE mode
//      12 ..... UNIMPLEMENTED ..... 0 ..... X
//      11 ..... RESERVED ..... 0 ..... X
//      10 ..... ALTIO .... ALTERNATE I/O SELECTION.. 0 ..... UART communicates using U1TX
//                                                            and U1RX I/O pins
//      9 ..... RESERVED ..... 0 ..... X
//      8 ..... RESERVED ..... 0 ..... X
//      7 ..... WAKE ..... WAKE-UP ON START BIT .... 0 ..... Wake-up disabled
//      6 ..... LPBACK ... LOOPBACK MODE SELECT .... 0 ..... Loopback mode disabled
//      5 ..... ABAUD .... AUTO BAUD ENABLE ..... 0 ..... Input to capture module from
//                                                            ICx pin
//      4 ..... UNIMPLEMENTED ..... 0 ..... X
//      3 ..... UNIMPLEMENTED ..... 0 ..... X
//      2 ..... PDSEL<1> . PARITY AND DATA SELECTION 0 ..... 8-bit data
//      1 ..... PDSEL<0>                            0       No parity
//      0 ..... STSEL .... STOP SELECTION ..... 0 ..... 1 Stop bit
```

```
//-------------------------------------------------------------------
//      U1STA:          UART1 STATUS AND CONTROL REGISTER            U1STA = 0x0000
//-------------------------------------------------------------------
//      BIT          NAME            FUNCTION           VALUE      DESCRIPTION
//-------------------------------------------------------------------
//      15 ....... UTXISEL .. TRANSMISSION INTERRUPT .. 0 ...... Interrupt when a character is
//                                                                transferred to shift register
//      14 ..................... UNIMPLEMENTED ........... 0 ...... X
//      13 ..................... UNIMPLEMENTED ........... 0 ...... X
//      12 ..................... UNIMPLEMENTED ........... 0 ...... X
//      11 ....... UTXBRK ... TRANSMIT BREAK .......... 0 ...... U1TX pin operates normally
//      10 ....... UTXEN .... TRANSMIT ENABLE ......... 0 ...... UART transmitter NOT ENABLED
//      9 ........ UTXBF .... TRANSMIT BUFFER FULL .... 0 ...... (READ ONLY)
//      8 ........ TRMT ..... TRANSMIT SHIFT REG EMPTY  0 ...... (READ ONLY)
//      7 ........ URXISEL<1> RECEIVE INTERRUPT MODE .. 0 ...... Interrupt flag set when
//                                                                character is received
//      6 ........ URXISEL<0>                           0
//      5 ........ ADDEN .... ADDRESS CHARCTER DETECT . 0 ...... Address detect mode disabled
//      4 ........ RIDLE .... RECEIVER IDLE ........... 0 ...... (READ ONLY)
//      3 ........ PERR ..... PARITY ERROR STATUS ..... 0 ...... (READ ONLY)
//      2 ........ FERR ..... FRAMING ERROR STAUS ..... 0 ...... (READ ONLY)
//      1 ........ OERR ..... RX BUFFER OVERRUN ERROR . 0 ...... (READ ONLY)
//      0 ........ URXDA .... RX BUFFER DATA AVAILABLE. 0 ...... (READ ONLY)
//-------------------------------------------------------------------
void    InitUART1(void)
{
        U1MODE  =       0x8000;
        U1STA   =       0x0000;

        U1BRG   = ((FCY/16)/BAUD)-1;          //      Sets the baud rate to 57600 baud
}
//===================================================================
//      PUBLIC FUNCTION:        RMSCalculation
//===================================================================
//      FUNCTION CALL: RMSCalculation(*dataPtr,N);
//      ARGUMENTS:      unsigned int *dataPtr,
//                      int N
//      RETURNS:        unsigned long Result_RMS_calulation
//-------------------------------------------------------------------
//      DESCRIPTION:    This routine takes an array of length N and converts
//                      it into a single RMS value using the equation below
//
//                                          ┌──────────┐
//                                          │┌  N      ┐│
//                      RMSValue    =       ││  Σ [(ui)²] ││
//                                          ││ i=0      ││
//                                          ││ ──────── ││
//                                          √└    N     ┘│
//-------------------------------------------------------------------

unsigned int RMSCalculation(unsigned int *dataPtr, int N)
{
        int i = 0;                                      //      Initialise counter

        unsigned int  squared_value   = 0;              //      Stores the Squared value
        unsigned int  sum             = 0;              //      Stores sum
        unsigned int  average         = 0;              //      Stores average
        signed int    RMS_valueQ15    = 0;              //      Stores RMS value in Q1.15 format
        unsigned int  RMS_value       = 0;              //      Stores RMS value in unsigned
                                                        //      integer format

        for (i=0;i<N;i++)                               //      Convert all values in array
        {
                squared_value =  pow(*dataPtr++,2);     //      Square each value
                sum += squared_value;                   //      and sum them all together
        }

        average = sum/N;                        //      Divide the sum by the number of values
                                                //      to find the average
        RMS_valueQ15 = (int)sqrt(average);      //      Square root the average to find RMS value

        RMS_value = Q15toInteger(RMS_valueQ15);         //      Convert from Q1.15 to integer

        return RMS_value;
}
```

```
//=====================================================================
//      PUBLIC FUNCTION:       Q15toInteger
//=====================================================================
//      FUNCTION CALL: Q15toInteger(const signed int value);
//      ARGUMENTS:     Integer value
//      RETURNS:       Unsigned int converted_result
//---------------------------------------------------------------------
//      DESCRIPTION:   This function converts a 16-bit number in Q1.15
//                     data format to an unsigned integer
//
//                     Signed Fractional (Q1.15)
//                     (DOUT = sddd dddd dddd 0000)
//
//                     Unsigned Integer
//                     (DOUT = 0000 dddd dddd dddd)
//---------------------------------------------------------------------

unsigned int Q15toInteger(signed int value)
{
        value = value>>4;

        unsigned int value_bytes12 = 0;              //     Stores bytes 1 and 2
        unsigned int value_byte3   = 0;              //     Stores byte 3
        unsigned int integer_value = 0;              //     Stores the converted integer value

        value_bytes12 = value & 0x00FF;              //     Select bytes 1 and 2

        if (value >= 0x0800)                         //     If the sign bit is 1, CLEAR it
        {
                value_byte3 = value >> 8;
                value_byte3 -= 0x0008;
                value_byte3 = value_byte3 << 8;

                integer_value = value_bytes12 + value_byte3;
        }
        else if (value < 0x0800)                     //     If the sign bit is 0, SET it
                {
                value_byte3 = value >> 8;
                value_byte3 += 0x0008;
                value_byte3 = value_byte3 << 8;

                integer_value = value_bytes12 + value_byte3;
        }

        return integer_value;                        //     Return the converted value
}
```

```
//=====================================================================
//      PUBLIC FUNCTION:       asciiVoltsConv
//=====================================================================
//      FUNCTION CALL: asciiVoltsConv(const unsigned int value);
//      ARGUMENTS:     Integer value
//      RETURNS:       Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:   This function converts a 12-bit binary word to a
//                     BCD ascii decimal value
//
//                     Resolution of this function is 1mV
//---------------------------------------------------------------------
void    asciiVoltsConv(const unsigned int value)
{
    int unitsInt         = 0;                          //      Initialise local variables
    int tenthsInt        = 0;
    int hundredthsInt    = 0;
    int thousandthsInt   = 0;

    int n                = value;

    if (n >= 0x04D9)                                   //      If there are 1Vs
    {
        unitsInt    = n/0x04D9;
        units       = asciiConv(unitsInt);             //      convert to an ascii char
        n           = n%0x04D9;           //  the remainder is ready for more conversion
    }
    if (n >= 0x007C)                                   //      If there are 100mVs
    {
        tenthsInt     = n/0x007C;
        tenths        = asciiConv(tenthsInt);          //      convert to an ascii char
        n             = n%0x007C;           //  the remainder is ready for more conversion
    }
    if (n >= 0x000C)                                   //      If there are 10mVs
    {
        hundredthsInt  = n/0x000C;
        hundredths     = asciiConv(hundredthsInt);     //      convert to an ascii char
        n              = n%0x000C;         // the remainder is ready for more conversion
    }
    if (value >= 0x0001)                               //      If there are 1mVs
    {
        thousandthsInt = n/0x0001;
        thousandths    = asciiConv(thousandsInt);      //      convert to an ascii char
        n              = n%0x0001;         // the remainder is ready for more conversion
    }
}
//=====================================================================
//      PUBLIC FUNCTION:       asciiConv
//=====================================================================
//      FUNCTION CALL:         asciiConv(const unsigned int value);
//      ARGUMENTS:             Integer Value
//      RETURNS:               Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:           This function converts a 16-bit value to an 8-bit
//                             ascii character
//---------------------------------------------------------------------
char    asciiConv(const unsigned int value)
{
        char no;

        no = (char)(value & 0x0F);                     //      Only select bits 0-3

        if      (no     > 9)                           //      If number is > 9
        {
                no = no + 0x37;                        //  Number will be (A = 0x41, B = 0x42, C
                                                       //      = 0x43 etc )
        }
        else
        {
                no = no + 0x30;                        //      Number will be 0 - 9
        }

        return no;
}
```

```
//=================================================================
//      INTERRUPT SERVICE ROUTINE (ISR):      _U1TXInterrupt
//=================================================================
//      DESCRIPTION:    This ISR clears the interrupt has occurred flag and
//                      then transmits a word of data
//-----------------------------------------------------------------
void    __attribute__((__interrupt__)) _U1TXInterrupt(void)
{
        IFS0bits.U1TXIF = 0;                            //      Clear transmit interrupt has
                                                        //      occurred flag
        U1TXREG = (int)OutData[txCount++];              //      Write a single word to UART
        if(txCount == TX_BUFFERSIZE)
        {
                IEC0bits.U1TXIE = 0;                    //      Disable transmit interrupts since
        all buffer values
                                                        //      are transmitted
        }
}
//=================================================================
```

## *ADC Testing – TestMIC.c*

```
//=====================================================================
//              ELECTRET CONDENSOR MICROPHONE TESTING
//=====================================================================
//      BY:             ROBYN VERRINDER
//                      Parts of this program were adapted from the
//                      Example Code for the dsPICDEM Low Cost Starter Board
//                      ADC12_UART.c (D'Souza, S., 2003) [68] and
//                      the example code CE001_ADC_DSP_lib_FILTER ADC.c
//                      (Vasuki, H., 2005) [69].
//      DATE:           06 MAY 2006
//=====================================================================
//      PROCESSOR:      dsPIC30F6014A
//      BOARD:          VIBRATION DATA LOGGING PROTOTYPE BOARD
//      IDE:            MPLAB IDE v7.30
//      COMPILER:       C30 C COMPILER
//=====================================================================
//      DESCRIPTION:    This program tests the functioning of both the
//                      12-bit Analog to Digital Convertor (ADC) Module of
//                      the dsPIC30F6014A and the Electret Condensor Microphone
//                      which can be connected to the socket J2 on the Vibration
//                      Data Logging Prototype Board.
//
//                      The data is outputed via the UART1 module to a PC.
//                      Hyperterminal is used to view the data.
//
//                      The pin connections of the Electret Condesor Microphone (MK1)
//                      to the microcontroller are shown below.
//
//                      Register descriptions are as in dsPIC30F Family Reference Manual
//                      Section 12: Timers, Section 18: 12 bit A/D Convertor and Section 19:
//                      UART [63].
//=====================================================================
//      XTAL FREQ (Fxtal):      10MHz
//      MIPS:                   20MIPS  (XT PLLx8 Mode)
//      TCY:                    50ns
//      Tconv:                  20us    (this value is greater than the 10us minimum)
//      Tad:                    1.45us  (calculated value using ADCS<5:0> bits)
//=====================================================================                                    //
//      SAMPLING FREQUENCIES FOR DEVICES:
//---------------------------------------------------------------------
//      ELECTRET MICROPHONE:
//      fsound          =       20Hz - 20kHz
//      fmic            =       50Hz - 13kHz
//      let fmax        =       10kHz
//      fs              >=      20kHz
//      Fs              =       20kHz
//      SAMC<4:0>       =       10101   (21)
//      ADCS<5:0>       =       111001  (57)
//=====================================================================
//      MK1:    Electret Condensor Microphone
//=====================================================================
//      PIN OUTS:
//      1       =       GND ................. GROUND
//      2       =       SIGNAL .............. OUTPUT SIGNAL
//      3       =       BIAS ................ BIAS VOLTAGE 3.3V - 12V
//
//      PIN CONNECTIONS: (MK1 to dsPIC30F6014A)
//      GND                     =       AGND
//      SIGNAL (MIC_IN) -> U6 (PIN2) -> (PIN7) MIC_FILT -> JP6 -> MIC_ADC   = AN11 (PIN 30)
//      BIAS    (VIA R22)       =       AVDD
//
//      GAIN from MIC to dsPIC30F6014A ....... 2.2
//      OFFSET VALUE FROM OPAMP .............. 1.67V -> 0x0819
//---------------------------------------------------------------------
#include              "p30f6014a.h"          //      (Sinha, P., 2005) [64].
#include              <math.h>               //      Standard C Math header file
#include              "dsPIC30F6014APins.h"
#include              "common.h"
```

```
//-------------------------------------------------------------------
//      dsPIC30F6014A CONFIGURATION SETTINGS
//-------------------------------------------------------------------
//      CLOCK SWITCHING AND MONITOR ..... Sw Disabled, Mon Disabled
//      OSCILLATOR ..... XT w/PLL 8x
//      WATCHDOG TIMER ..... Disabled
//      MASTER CLEAR ENABLE ..... Enabled
//      POR TIMER VALUE ..... 64ms
//      GENERAL CODE SEGMENT CODE PROTECT ..... Disabled
//-------------------------------------------------------------------
_FOSC(CSW_FSCM_OFF & XT_PLL8);
_FWDT(WDT_OFF);
_FBORPOR(MCLR_EN & PWRT_64);
_FGS(CODE_PROT_OFF);
//-------------------------------------------------------------------
//      FUNCTION DEFINITIONS
//-------------------------------------------------------------------

void    InitADC12(void);
void    InitUART1(void);

unsigned long RMSCalculation(unsigned int *dataPtr, int N);

void    SendDatatoUART1(unsigned long data);

void    asciiDecConv(const unsigned int value);
char    asciiConv(unsigned int value);

void     __attribute__((__interrupt__)) _U1TXInterrupt(void);

//-------------------------------------------------------------------
//      ASCII BUFFERS FOR SERIAL DATA
//-------------------------------------------------------------------
//      BYTE         NAME          VALUE
//-------------------------------------------------------------------
//      8 ........ DATA<3> ........ 0x30 ........ Units
//      7 ........ '.' ............ 0x2E ........ Decimal Point
//      6 ........ DATA<2> ........ 0x31 ........ Tenths
//      5 ........ DATA<1> ........ 0x32 ........ Hundredths
//      4 ........ DATA<0> ........ 0x33 ........ Thousands
//      3 ........ SPACE ........... 0x20
//      2 ........ 'V' ............. 0x43
//      1 ........ LF .............. 0x0A
//      0 ........ CR .............. 0x0D
//-------------------------------------------------------------------

unsigned char  OutData[TX_BUFFERSIZE] = (0x30,0x2E,0x31,0x32,0x33,0x20,0x56,0x0A,0x0D);

//-------------------------------------------------------------------
//      GLOBAL VARIABLES
//-------------------------------------------------------------------
unsigned int   *ADCbufferPtr;                          //      Pointer to ADCbuff
unsigned int   *ADCvaluePtr;

unsigned char  txCount;

int            RMS_value;
unsigned char  ADC_bcd   = 0;

unsigned char  units;
unsigned char  tenths;
unsigned char  hundredths;
unsigned char  thousandths;

unsigned char  ADCValue[ADC_BUFFERSIZE];

//-------------------------------------------------------------------
//      MAIN FUNCTION
//-------------------------------------------------------------------

int            main(void)
(
       InitUART1();
       InitADC12();

       int i = 0;                              //      Initalise Counter
```

```
        ADCON1bits.ADON = 1;                    //      Turn the ADC Module ON

        while(1)                                //      Do forever
        {

                RMS_value = 0;                  //      Reset RMS value
                ADCbufferPtr = &ADCBUF0;        //      Point to the first value in the ADC
                                                //      buffer
                ADCvaluePtr = &ADCValue
                IFS0bits.ADIF = 0;              //      Clear 8 samples have been converted flag

                ADCON1bits.ASAM = 1;            //      Sampling begins immediately after last
                                                //      conversion ends

                while(!IFS0bits.ADIF)           //      Wait for conversion to finish
                {
                        /* NULL STATEMENT */;
                }

                ADCON1bits.ASAM = 0;            //      Sampling begins when SAMP bit is set

                for (i=0; i<ADC_BUFFERSIZE; i++)
                {
                        *ADCvaluePtr++ = *ADCbufferPtr++ - OFFSET;    //      Subtract offset
                                                                      //      from each sample
                }

                RMS_value = RMSCalcultaion(ADCvaluePtr, ADC_BUFFERSIZE);

                SendDatatoUART1(RMS_value);                   //      Send the data via UART1 to
                                                              //      PC running HyperTerminal
        }
}

//=================================================================
//      PUBLIC FUNCTION:        InitADC12
//=================================================================
//      FUNCTION CALL:          InitADC12();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//-----------------------------------------------------------------
//      DESCRIPTION:            Initialises the 12 bit Analogue to Digital Converter
//                              All the registers are set as laid out below
//-----------------------------------------------------------------
//      SAMPLING RATE:          20 kHz
//-----------------------------------------------------------------
//      ADCON1:         A/D     CONTROL REGISTER 1              ADCON1 = 0x03E0
//-----------------------------------------------------------------
//      BIT       NAME            FUNCTION                VALUE       DESCRIPTION
//-----------------------------------------------------------------
//      15 ..... ADON ..... A/D OPERATING MODE ..... 0 ..... ADC is OFF
//      14 ..... ..... UNIMPLEMENTED ..... 0 ..... X
//      13 ..... ADSIDL .... STOP IN IDLE MODE ..... 0 ..... Operates in IDLE mode
//      12 ..... ..... UNIMPLEMENTED ..... 0 ..... X
//      11 ..... ..... UNIMPLEMENTED ..... 0 ..... X
//      10 ..... ..... UNIMPLEMENTED ..... 0 ..... X
//      9 ..... FORM<1> .. DATA OUTPUT FORMAT ..... 1 ..... Signed Fractional
//      8 ..... FORM<0>                             1       (DOUT = sddd dddd dddd 0000)
//      7 ..... SSRC<2> .. CONVERSION TRIGGER SOURCE 1 ..... Internal counter ends
//      6 ..... SSRC<1>                             1       sampling and starts
//      5 ..... SSRC<0>                             1       conversion (Auto convert)
//      4 ..... ..... UNIMPLEMENTED ..... 0 ..... X
//      3 ..... ..... UNIMPLEMENTED ..... 0 ..... X
//      2 ..... ASAM ..... A/D SAMPLE AUTO START ... 0 ..... Sampling begins when SAMP bit
//                                                          is set
//      1 ..... SAMP ..... A/D SAMPLE ENABLE ..... 0 ..... SAMP bit is auto set by ASAM
//                                                          setting
//      0 ..... DONE ..... A/D CONVERSION STATUS ... 0 ..... Automatically cleared and set
//                                                          by MCU
//-----------------------------------------------------------------
//      ADCON2:         A/D     CONTROL REGISTER 2              ADCON2 = 0x0020
//-----------------------------------------------------------------
//      BIT       NAME            FUNCTION                VALUE       DESCRIPTION
//-----------------------------------------------------------------
//      15 ..... VCFG<2> ... VOLTAGE REF CONFIG ..... 0 ..... A/D VREFH = AVDD
//      14 ..... VCFG<1>                            0       A/D VREFL = AVSS
//      13 ..... VCFG<0>                            0
```

```
//      12 ................. RESERVED ............... 0
//      11 ................. UNIMPLEMENTED .......... 0 ...... X
//      10 ....... CSCNA .... SCAN INPUTS ............ 0 ...... Do not scan inputs
//      9 .................. UNIMPLEMENTED .......... 0 ...... X
//      8 .................. UNIMPLEMENTED .......... 0 ...... X
//      7 ........ BUFS ..... BUFFER FILL STATUS ..... 0 ...... Auto set (1 = A/D is filling
//                                                              0x8-0xF find data
//                                                              in 0x0-0x7; 0 = A/D is
//                                                              filling 0x0-0x7 find
//                                                              data in 0x8-0xF)
//      6 .................. UNIMPLEMENTED .......... 0 ...... X
//      5 ........ SMPI<3> .. SAMP-CONV SEQ/INTERRUPT . 1 ...... Interrupts at the completion

//      4 ........ SMPI<2>                            0         of the conversion sequence
//      3 ........ SMPI<1>                            0         for each 8th sample/convert
//                                                              sequence
//      2 ........ SMPI<0>                            0
//      1 ........ BUFM ..... BUFFER MODE SELECT ..... 0 ...... Buffer configured as one 16-
//                                                              word buffer
//      0 ........ ALTS ..... ALTERNATE INPUT SAMPLE .. 0 ...... Always use MUX A input
//                                                              multiplexer settings
//-----------------------------------------------------------------
//      ADCON3:         A/D     CONTROL REGISTER 3              ADCON3 = 0x1539
//-----------------------------------------------------------------
//      BIT         NAME            FUNCTION                VALUE       DESCRIPTION
//-----------------------------------------------------------------
//      15 ................. UNIMPLEMENTED .......... 0 ...... X
//      14 ................. UNIMPLEMENTED .......... 0 ...... X
//      13 ................. UNIMPLEMENTED .......... 0 ...... X
//      12 ....... SAMC<4> .. AUTO SAMPLE TIME ........ 1 ...... SAMC = 21Tad
//      11 ........SAMC<3>                            0
//      10 ........SAMC<2>                            1
//      9 .........SAMC<1>                            0
//      8 .........SAMC<0>                            1
//      7 ........ ADRC ..... A/D CONVERSION CLK SOURCE 0 ...... Clock derived from system clk
//      6 .................. UNIMPLEMENTED .......... 0 ...... X
//      5 ........ ADCS<5> .. A/D CONVERSION CLOCK .... 1 ...... TCY/2(ADCS<5:0> + 1) = 29TCY
//      4 ........ ADCS<4>                            1
//      3 ........ ADCS<3>                            1
//      2 ........ ADCS<2>                            0
//      1 ........ ADCS<1>                            0
//      0 ........ ADCS<0>                            1
//-----------------------------------------------------------------
//      ADCHS:          A/D     INPUT SELECT REGISTER           ADCHS = 0x000B
//-----------------------------------------------------------------
//      BIT         NAME            FUNCTION                VALUE       DESCRIPTION
//-----------------------------------------------------------------
//      15 ................. UNIMPLEMENTED .......... 0 ...... X
//      14 ................. UNIMPLEMENTED .......... 0 ...... X
//      13 ................. UNIMPLEMENTED .......... 0 ...... X
//      12 ....... CH0NB .... CH0- INPUT SELECT MUXB .. 0 ...... NOT USED
//      11 ....... CH0NB<3> . CH0+ INPUT SELECT MUXB .. 0 ...... NOT USED
//      10 ....... CH0NB<2>                           0
//      9 ........ CH0NB<1>                           0
//      8 ........ CH0NB<0>                           0
//      7 .................. UNIMPLEMENTED .......... 0 ...... X
//      6 .................. UNIMPLEMENTED .......... 0 ...... X
//      5 .................. UNIMPLEMENTED .......... 0 ...... X
//      4 ........ CH0NA .... CH0- INPUT SELECT MUXA .. 0 ...... NOT USED
//      3 ........ CH0NA<3> . CH0+ INPUT SELECT MUXA .. 1 ...... Channel 0 positive input is
//                                                              AN11
//      2 ........ CH0NA<2>                           0
//      1 ........ CH0NA<1>                           1
//      0 ........ CH0NA<0>                           1

//-----------------------------------------------------------------
//      ADPCFC:         A/D     PORT CONFIGURATION REGISTER         ADPCFC = 0xF7FF
//-----------------------------------------------------------------
//      BIT         NAME            FUNCTION                VALUE       DESCRIPTION
//-----------------------------------------------------------------
//      15 ....... PCFC<15> . ANALOG INPUT PIN CONFIG . 1 ...... X
//      14 ....... PCFG<14>                           1 ...... X
//      13 ....... PCFG<13>                           1 ...... X
//      12 ....... PCFG<12>                           1 ...... X
//      11 ....... PCFG<11>                           0 ...... MIC_ADC
//      10 ....... PCFG<10>                           1 ...... X
//      9  ....... PCFG<9>                            1 ...... X
```

```
//      8 ........ PCFG<8>                              1 ...... X
//      7 ........ PCFG<7>                              1 ...... X
//      6 ........ PCFG<6>                              1 ...... X
//      5 ........ PCFG<5>                              1 ...... X
//      4 ........ PCFG<4>                              1 ...... X
//      3 ........ PCFG<3>                              1 ...... X
//      2 ........ PCFG<2>                              1 ...... X
//      1 ........ PCFG<1>                              1 ...... PGC
//      0 ........ PCFG<0>                              1 ...... PGD
//-------------------------------------------------------------------
//      ADCSSL:         A/D     INPUT SCAN SELECT REGISTER          ADCSSL = 0x0000
//-------------------------------------------------------------------
//      BIT         NAME            FUNCTION            VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ....... CSSL<15> . A/D INPUT PIN SCAN SELECT 0 ...... X
//      14 ....... CSSL<14>                             0 ...... X
//      13 ....... CSSL<13>                             0 ...... X
//      12 ....... CSSL<12>                             0 ...... X
//      11 ....... CSSL<11>                             0 ...... MIC_ADC
//      10 ....... CSSL<10>                             0 ...... X
//      9  ....... CSSL<9>                              0 ...... X
//      8 ........ CSSL<8>                              0 ...... X
//      7 ........ CSSL<7>                              0 ...... X
//      6 ........ CSSL<6>                              0 ...... X
//      5 ........ CSSL<5>                              0 ...... X
//      4 ........ CSSL<4>                              0 ...... X
//      3 ........ CSSL<3>                              0 ...... X
//      2 ........ CSSL<2>                              0 ...... X
//      1 ........ CSSL<1>                              0 ...... PGC
//      0 ........ CSSL<0>                              0 ...... PGD
//-------------------------------------------------------------------

void            InitADC12(void)
{
        ADCON1 = 0x03E0;
        ADCON2 = 0x0020;
        ADCON3 = 0x1539;

        ADCHS  = 0x000B;
        ADPCFG = 0xF7FF;
        ADCSSL = 0x0000;

        IFS0bits.ADIF  =        0;              //      Clear ADC interrupt has occurred flag
        IEC0bits.ADIE  =        0;              //      Disable ADC interrupts
}

//===================================================================
//      PUBLIC FUNCTION:        InitUART1
//===================================================================
//      FUNCTION CALL:          InitUART1();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//-------------------------------------------------------------------
//      DESCRIPTION:            Initialises the UART1 module on the dsPIC30F6014A.
//                              The value of all the registers are laid out below
//-------------------------------------------------------------------
//      BAUD RATE:              57600 bps
//      DATA SIZE:              8 Bits
//      PARITY:                 No parity
//      STOP BIT:               1 stop bit
//-------------------------------------------------------------------
//      U1MODE:         UART1 MODE REGISTER             U1MODE = 0x8000
//-------------------------------------------------------------------
//      BIT           NAME            FUNCTION            VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ....... UARTEN ... UART ENABLE BIT ......... 1 ...... UART is ENABLED
//      14 ................. UNIMPLEMENTED ............ 0 ...... X
//      13 ....... USIDL .... STOP IN IDLE MODE ....... 0 ...... Operates in IDLE mode
//      12 ................. UNIMPLEMENTED ............ 0 ...... X
//      11 ................. RESERVED ................. 0 ...... X
//      10 ....... ALTIO .... ALTERNATE I/O SELECTION.. 0 ...... UART communicates using U1TX
//                                                                 and U1RX I/O pins
//      9 ................. RESERVED ................. 0 ...... X
//      8 ................. RESERVED ................. 0 ...... X
//      7 ........ WAKE ..... WAKE-UP ON START BIT .... 0 ...... Wake-up disabled
//      6 ........ LPBACK ... LOOPBACK MODE SELECT .... 0 ...... Loopback mode disabled
//      5 ........ ABAUD .... AUTO BAUD ENABLE ........ 0 ...... Input to capture module from
```

```
//                                                                    ICx pin
//
//      4 ................. UNIMPLEMENTED ........... 0 ...... X
//      3 ................. UNIMPLEMENTED ........... 0 ...... X
//      2 ........ PDSEL<1> . PARITY AND DATA SELECTION 0 ...... 8-bit data
//      1 ........ PDSEL<0>                            0        No parity
//      0 ........ STSEL .... STOP SELECTION .......... 0 ...... 1 Stop bit
//-------------------------------------------------------------------
//      U1STA:          UART1 STATUS AND CONTROL REGISTER              U1STA = 0x0000
//-------------------------------------------------------------------
//      BIT          NAME            FUNCTION           VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ....... UTXISEL .. TRANSMISSION INTERRUPT .. 0 ...... Interrupt when a character is
//                                                                  transferred to shift register
//      14 ................. UNIMPLEMENTED ........... 0 ...... X
//      13 ................. UNIMPLEMENTED ........... 0 ...... X
//      12 ................. UNIMPLEMENTED ........... 0 ...... X
//      11 ....... UTXBRK ... TRANSMIT BREAK .......... 0 ...... U1TX pin operates normally
//      10 ....... UTXEN .... TRANSMIT ENABLE ......... 0 ...... UART transmitter NOT ENABLED
//      9 ........ UTXBF .... TRANSMIT BUFFER FULL .... 0 ...... (READ ONLY)
//      8 ........ TRMT ..... TRANSMIT SHIFT REG EMPTY  0 ...... (READ ONLY)
//      7 ........ URXISEL<1> RECEIVE INTERRUPT MODE .. 0 ...... Interrupt flag set when
//                                                                  character is received
//      6 ........ URXISEL<0>                          0
//      5 ........ ADDEN .... ADDRESS CHARCTER DETECT . 0 ...... Address detect mode disabled
//      4 ........ RIDLE .... RECEIVER IDLE ........... 0 ...... (READ ONLY)
//      3 ........ PERR ..... PARITY ERROR STATUS ..... 0 ...... (READ ONLY)
//      2 ........ FERR ..... FRAMING ERROR STAUS ..... 0 ...... (READ ONLY)
//      1 ........ OERR ..... RX BUFFER OVERRUN ERROR . 0 ...... (READ ONLY)
//      0 ........ URXDA .... RX BUFFER DATA AVAILABLE. 0 ...... (READ ONLY)
//-------------------------------------------------------------------
void    InitUART1(void)
{
        U1MODE  =       0x8000;
        U1STA   =       0x0000;

        U1BRG   = ((FCY/16)/BAUD)-1;          //      Sets the baud rate to 57600 baud
}

//===================================================================
//      PUBLIC FUNCTION:        RMSCalculation
//===================================================================
//      FUNCTION CALL: RMSCalculation(*dataPtr,N);
//      ARGUMENTS:      unsigned int *dataPtr,
//                      int N
//      RETURNS:        unsigned long Result_RMS_calulation
//-------------------------------------------------------------------
//      DESCRIPTION:    This routine takes an array of length N and converts
//                      it into a single RMS value using the equation below
//
//                                                  ___________
//                                                 |┌  N        ┐
//                      RMSValue  =                ||  Σ [(ui)²] |
//                                                 || i=0       |
//                                                 ||  ―――――――  |
//                                                √|└      N    ┘
//-------------------------------------------------------------------
unsigned int RMSCalculation(unsigned int *dataPtr, int N)
{
        int i = 0;                                      //      Initialise counter

        unsigned int  squared_value   = 0;              //      Stores the Squared value
        unsigned int  sum             = 0;              //      Stores sum
        unsigned int  average         = 0;              //      Stores average
        signed int    RMS_valueQ15    = 0;              //      Stores RMS value in Q1.15 format
        unsigned int  RMS_value       = 0;              //      Stores RMS value in unsigned
                                                        //      integer format

        for (i=0;i<N;i++)                               //      Convert all values in array
        {
                squared_value =  pow(*dataPtr++,2);     //      Square each value
                sum += squared_value;                   //      and sum them all together
        }

        average = sum/N;                                //      Divide the sum by the number of
                                                        //      values
                                                        //      to find the average
        RMS_valueQ15 = (int)sqrt(average);              //      Square root the average to find
                                                        //      the RMS value
```

```
        RMS_value = Q15toInteger(RMS_valueQ15);       //      Convert from Q1.15 to integer

        return RMS_value;
}

//=============================================================================
//      PUBLIC FUNCTION:        Q15toInteger
//=============================================================================
//      FUNCTION CALL: Q15toInteger(const signed int value);
//      ARGUMENTS:      Integer value
//      RETURNS:        Unsigned int converted_result
//-----------------------------------------------------------------------------
//      DESCRIPTION:    This function converts a 16-bit number in Q1.15
//                      data format to an unsigned integer
//
//                      Signed Fractional (Q1.15)
//                      (DOUT = sddd dddd dddd 0000)
//
//                      Unsigned Integer
//                      (DOUT = 0000 dddd dddd dddd)
//-----------------------------------------------------------------------------

unsigned int Q15toInteger(signed int value)
{
        value = value>>4;

        unsigned int value_bytes12 = 0;               //      Stores bytes 1 and 2
        unsigned int value_byte3   = 0;               //      Stores byte 3
        unsigned int integer_value = 0;               //      Stores the converted integer value

        value_bytes12 = value & 0x00FF;               //      Select bytes 1 and 2

        if (value >= 0x0800)                          //      If the sign bit is 1, CLEAR it
        {
                value_byte3 = value >> 8;
                value_byte3 -= 0x0008;
                value_byte3 = value_byte3 << 8;

                integer_value = value_bytes12 + value_byte3;
        }
        else if (value < 0x0800)                      //      If the sign bit is 0, SET it
                {
                value_byte3 = value >> 8;
                value_byte3 += 0x0008;
                value_byte3 = value_byte3 << 8;

                integer_value = value_bytes12 + value_byte3;
        }

        return integer_value;                         //      Return the converted value
}
```

```
//=====================================================================
//      PUBLIC FUNCTION:        asciiVoltsConv
//=====================================================================
//      FUNCTION CALL: asciiVoltsConv(const unsigned int value);
//      ARGUMENTS:      Integer value
//      RETURNS:        Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:    This function converts a 12-bit binary word to a
//                      BCD ascii decimal value
//
//                      Resolution of this function is 1mV
//---------------------------------------------------------------------
void    asciiVoltsConv(const unsigned int value)
{
    int unitsInt          = 0;                          //      Initialise local variables
    int tenthsInt         = 0;
    int hundredthsInt     = 0;
    int thousandthsInt    = 0;

    int n               = value;

    if (n >= 0x04D9)                                    //      If there are 1Vs
    {
        unitsInt    = n/0x04D9;
        units       = asciiConv(unitsInt);              //      convert to an ascii char
        n           = n%0x04D9;           //  the remainder is ready for more conversion
    }
    if (n >= 0x007C)                                    //      If there are 100mVs
    {
        tenthsInt     = n/0x007C;
        tenths        = asciiConv(tenthsInt);           //      convert to an ascii char
        n             = n%0x007C;         //  the remainder is ready for more conversion
    }
    if (n >= 0x000C)                                    //      If there are 10mVs
    {
        hundredthsInt  = n/0x000C;
        hundredths     = asciiConv(hundredthsInt);      //      convert to an ascii char
        n              = n%0x000C;        // the remainder is ready for more conversion
    }
    if (value >= 0x0001)                                //      If there are 1mVs
    {
        thousandthsInt = n/0x0001;
        thousandths    = asciiConv(thousandsInt);       //      convert to an ascii char
        n              = n%0x0001;        // the remainder is ready for more conversion
    }
}
//=====================================================================
//      PUBLIC FUNCTION:        asciiConv
//=====================================================================
//      FUNCTION CALL:          asciiConv(const unsigned int value);
//      ARGUMENTS:              Integer Value
//      RETURNS:                Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:            This function converts a 16-bit value to an 8-bit
//                              ascii character
//---------------------------------------------------------------------
char    asciiConv(const unsigned int value)
{
        char no;

        no = (char)(value & 0x0F);                      //      Only select bits 0-3

        if      (no     > 9)                            //      If number is > 9
        {
                no = no + 0x37;                         //  Number will be (A = 0x41, B = 0x42, C
                                                        //      = 0x43 etc )
        }
        else
        {
                no = no + 0x30;                         //      Number will be 0 - 9
        }

        return no;
}
```

```
//=====================================================================
//      INTERRUPT SERVICE ROUTINE (ISR):      _U1TXInterrupt
//=====================================================================
//      DESCRIPTION:    This ISR clears the interrupt has occurred flag and
//                      then transmits a word of data
//---------------------------------------------------------------------
void    __attribute__((__interrupt__)) _U1TXInterrupt(void)
{
        IFS0bits.U1TXIF = 0;                            //      Clear transmit interrupt has
                                                        //      occurred flag
        U1TXREG = (int)OutData[txCount++];              //      Write a single word to UART
        if(txCount == TX_BUFFERSIZE)
        {
                IEC0bits.U1TXIE = 0;                    //      Disable transmit interrupts since
        all buffer values
                                                        //      are transmitted
        }
}
//=====================================================================
```

## *ADC Testing – TestMAX6608.c*

```
//=====================================================================
//              MAX6608 LOW-VOLTAGE ANALOG TEMPERATURE SENSOR TESTING
//=====================================================================
//      BY:             ROBYN VERRINDER
//                      Parts of this program were adapted from the
//                      Example Code for UART Module on the dsPICDEM
//                      Low Cost Starter Board
//                      ADC12_UART.c (D'Souza, S., 2003) [68].
//      DATE:           07 MAY 2006
//=====================================================================
//      PROCESSOR:      dsPIC30F6014A
//      BOARD:          VIBRATION DATA LOGGING PROTOTYPE BOARD
//      IDE:            MPLAB IDE v7.30
//      COMPILER:       C30 C COMPILER
//=====================================================================
//      DESCRIPTION:    This program tests the functioning of both the
//                      12-bit Analog to Digital Convertor (ADC) Module of
//                      the dsPIC30F6014A and the Low voltage analog temperature
//                      sensor (MAX6608) onboard the Vibration Data Logging
//                      Prototype Board. Jumper JP7 must be connected.
//
//                      The temperature format is °C
//
//                      The data is outputed via the UART1 module to a PC.
//                      Hyperterminal is used to view the data. Jumpers on JP8 must be
//                      connected.
//
//                      The pin connections of the MAX6608 (U7) to the microntroller
//                      are shown below. [46].
//
//                      This code was mainly adapted from the example code for the UART module
//                      on the dsPICDEM Low Cost Starter Board (D'Souza, S., 2003) [68].
//                      The InitUART, DelayNmSec and ISR for the UART module
//                      were written by Microchip. The InitADC12 and
//                      SendtoUART routines were adapted from their code, while the
//                      asciiVoltsConv and asciiConv routines were written by Robyn Verrinder.
//
//                      Register descriptions are as in dsPIC30F Family Reference Manual
//                      Section 18: 12 bit A/D Convertor and Section 19: UART
//                      Microchip Technology Incorporated Reference Manual, 2003a) [63].
//=====================================================================
//      XTAL FREQ (Fxtal):      10MHz
//      MIPS:                   20MIPS  (XT PLLx8 Mode)
//      TCY:                    50ns
//      ADC Resolution (bits): 12-bits
//      ADC Resolution (mV):    0.806 uV (for a 3.3V supply)
//=====================================================================
//      U7:     MAX6608 =       Low-Voltage Analog Temperature Sensor
//=====================================================================
//      PIN OUTS:
//                      1       =       NC ................. NOT CONNECTED
//                      2       =       GND ................ GROUND
//                      3       =       A .................. MUST BE CONNECTED TO GND
//                      4       =       VCC ................ SUPPLY INPUT
//                      5       =       OUT ................ TEMPERATURE SENSOR OUTPUT
//
//      PIN CONNECTIONS: (MAX6608 to dsPIC30F6014A)
//                      VCC     =       AVDD
//                      GND     =       AGND
//                      A       =       AGND
//                      OUT (TEMP) -> JP7 -> TEMP_ADC =       AN8 (PIN 27)
//                      T(°C) = (VOUT - 500mV) / 10mV/°C
//---------------------------------------------------------------------
#include              "p30f6014a.h"          //      (Sinha, P., 2005) [64].
#include              "dsPIC30F6014APins.h"
#include              "common.h"
//---------------------------------------------------------------------
//      dsPIC30F6014A CONFIGURATION SETTINGS
//---------------------------------------------------------------------
//      CLOCK SWITCHING AND MONITOR ..... Sw Disabled, Mon Disabled
//      OSCILLATOR ...................... XT w/PLL 8x
//      WATCHDOG TIMER .................. Disabled
//      MASTER CLEAR ENABLE ............. Enabled
```

```
//      POR TIMER VALUE ......................... 64ms
//      GENERAL CODE SEGMENT CODE PROTECT ......... Disabled
//---------------------------------------------------------------------
_FOSC(CSW_FSCM_OFF & XT_PLL8);
_FWDT(WDT_OFF);
_FBORPOR(MCLR_EN & PWRT_64);
_FGS(CODE_PROT_OFF);
//---------------------------------------------------------------------
//      FUNCTION DEFINITIONS
//---------------------------------------------------------------------

void    InitADC12(void);
void    InitUART1(void);

void    SendtoUART(void);
void     DelayNmSec(unsigned int N);

void    asciiDecConv(const unsigned int value);
char    asciiConv(unsigned int value);

void     __attribute__((__interrupt__)) _U1TXInterrupt(void);

//---------------------------------------------------------------------
//      ASCII BUFFERS FOR SERIAL DATA
//---------------------------------------------------------------------
//      BYTE         NAME          VALUE
//---------------------------------------------------------------------
//      7 ........ DATA<3> ........ 0x30 ........ Thousands
//      6 ........ DATA<2> ........ 0x31 ........ Hundreds
//      5 ........ DATA<1> ........ 0x32 ........ Tens
//      4 ........ DATA<0> ........ 0x30 ........ Units
//      3 ........ ' ' ............ 0x20 ........ Space
//      2 ........ 'C' ............ 0x43 ........ C (degrees Celcius)
//      1 ........ LF ............. 0x0A ........ New Line
//      0 ........ CR ............. 0x0D ........ Character Return
//---------------------------------------------------------------------

unsigned char  OutData[TX_BUFFERSIZE] = (0x30,0x31,0x32,0x33,0x20,0x43,0x0A,0x0D);

//---------------------------------------------------------------------
//      GLOBAL VARIABLES
//---------------------------------------------------------------------

unsigned char  txCount;

int            ADCValue = 0;

unsigned char  units;
unsigned char  tens;
unsigned char  hundreds;
unsigned char  thousands;

//---------------------------------------------------------------------
//      MAIN FUNCTION
//---------------------------------------------------------------------
int     main(void)
{
        InitUART1();
        InitADC12();

        while(1)                                    //      Do forever
        {
                ADCON1bits.SAMP  = 1;               //      Start sampling
                DelayNmSec(20);                     //      for 20ms
                ADCON1bits.SAMP  = 0;               //      Stop sampling

                while(!IFS0bits.ADIF)               //      Wait for conversion to finish
                {
                        /* NULL STATEMENT */;
                }

                ADCValue = ADCBUF0;                 //      Load 16-bit ADCValue with the
                                                    //      result of the conversion
                SendtoUART();                       //      Send the data via UART1 to PC
                                                    //      running HyperTerminal
        }
}
```

```
//=====================================================================
//      PUBLIC FUNCTION:        InitUART1       (D'Souza, S., 2003) [68].
//=====================================================================
//      FUNCTION CALL:          InitUART1();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:            Initialises the UART1 module on the dsPIC30F6014A.
//                              The value of all the registers are laid out below
//---------------------------------------------------------------------
//      BAUD RATE:              57600 bps
//      DATA SIZE:              8 Bits
//      PARITY:                 No parity
//      STOP BIT:               1 stop bit
//      FLOW CONTROL:           None
//---------------------------------------------------------------------
//      U1MODE:         UART1 MODE REGISTER             U1MODE = 0x8000
//---------------------------------------------------------------------
//      BIT          NAME           FUNCTION          VALUE       DESCRIPTION
//---------------------------------------------------------------------
//      15 ....... UARTEN ... UART ENABLE BIT ......... 1 ...... UART is ENABLED
//      14 ................. UNIMPLEMENTED ........... 0 ...... X
//      13 ....... USIDL .... STOP IN IDLE MODE ....... 0 ...... Operates in IDLE mode
//      12 ................. UNIMPLEMENTED ........... 0 ...... X
//      11 ................. RESERVED ................ 0 ...... X
//      10 ....... ALTIO .... ALTERNATE I/O SELECTION.. 0 ...... UART communicates using U1TX
//                                                               and U1RX I/O pins
//      9 ................. RESERVED ................ 0 ...... X
//      8 ................. RESERVED ................ 0 ...... X
//      7 ........ WAKE ..... WAKE-UP ON START BIT .... 0 ...... Wake-up disabled
//      6 ........ LPBACK ... LOOPBACK MODE SELECT .... 0 ...... Loopback mode disabled
//      5 ........ ABAUD .... AUTO BAUD ENABLE ........ 0 ...... Input to capture module from
//                                                               ICx pin
//      4 ................. UNIMPLEMENTED ........... 0 ...... X
//      3 ................. UNIMPLEMENTED ........... 0 ...... X
//      2 ........ PDSEL<1> . PARITY AND DATA SELECTION 0 ...... 8-bit data
//      1 ........ PDSEL<0>                           0         No parity
//      0 ........ STSEL .... STOP SELECTION .......... 0 ...... 1 Stop bit
//---------------------------------------------------------------------
//      U1STA:          UART1 STATUS AND CONTROL REGISTER           U1STA = 0x0000
//---------------------------------------------------------------------
//      BIT          NAME           FUNCTION          VALUE       DESCRIPTION
//---------------------------------------------------------------------
//      15 ....... UTXISEL .. TRANSMISSION INTERRUPT .. 0 ...... Interrupt when a character is
//                                                               transferred to shift register
//      14 ................. UNIMPLEMENTED ........... 0 ...... X
//      13 ................. UNIMPLEMENTED ........... 0 ...... X
//      12 ................. UNIMPLEMENTED ........... 0 ...... X
//      11 ....... UTXBRK ... TRANSMIT BREAK .......... 0 ...... U1TX pin operates normally
//      10 ....... UTXEN .... TRANSMIT ENABLE ......... 0 ...... UART transmitter NOT ENABLED
//      9 ........ UTXBF .... TRANSMIT BUFFER FULL .... 0 ...... (READ ONLY)
//      8 ........ TRMT ..... TRANSMIT SHIFT REG EMPTY  0 ...... (READ ONLY)
//      7 ........ URXISEL<1> RECEIVE INTERRUPT MODE .. 0 ...... Interrupt flag set when
//                                                               character is received
//      6 ........ URXISEL<0>                          0
//      5 ........ ADDEN .... ADDRESS CHARCTER DETECT . 0 ...... Address detect mode disabled
//      4 ........ RIDLE .... RECEIVER IDLE ........... 0 ...... (READ ONLY)
//      3 ........ PERR ..... PARITY ERROR STATUS ..... 0 ...... (READ ONLY)
//      2 ........ FERR ..... FRAMING ERROR STAUS ..... 0 ...... (READ ONLY)
//      1 ........ OERR ..... RX BUFFER OVERRUN ERROR . 0 ...... (READ ONLY)
//      0 ........ URXDA .... RX BUFFER DATA AVAILABLE. 0 ...... (READ ONLY)
//---------------------------------------------------------------------
void    InitUART1(void)
{
        U1MODE  =       0x8000;
        U1STA   =       0x0000;

        U1BRG   = ((FCY/16)/BAUD)-1;          //      Sets the baud rate to 9600 baud

        INTCON1bits.NSTDIS = 1;                       //      Disable nested interrupts
}
```

```
//=============================================================
//      PUBLIC FUNCTION:        InitADC12
//=============================================================
//      FUNCTION CALL:          InitADC12();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//-------------------------------------------------------------
//      DESCRIPTION:            Intialises the 12 bit Analogue to Digital Converter
//                              All the registers are set as laid out below
//-------------------------------------------------------------
//      ADCON1:         A/D     CONTROL REGISTER 1              ADCON1 = 0x0000
//-------------------------------------------------------------
//      BIT       NAME            FUNCTION                VALUE       DESCRIPTION
//-------------------------------------------------------------
//      15 ..... ADON ..... A/D OPERATING MODE ..... 0 ..... ADC is OFF
//      14 ................ UNIMPLEMENTED ........... 0 ..... X
//      13 ..... ADSIDL .... STOP IN IDLE MODE ..... 0 ..... Operates in IDLE mode
//      12 ................ UNIMPLEMENTED ........... 0 ..... X
//      11 ................ UNIMPLEMENTED ........... 0 ..... X
//      10 ................ UNIMPLEMENTED ........... 0 ..... X
//      9 ........ FORM<1> .. DATA OUTPUT FORMAT ..... 0 ..... Unsigned Integer
//      8 ........ FORM<0>                            0       (DOUT = 0000 dddd dddd dddd)
//      7 ........ SSRC<2> .. CONVERSION TRIGGER SOURCE 0 ..... Clearing SAMP bit ends
//      6 ........ SSRC<1>                            0       sampling and starts
//      5 ........ SSRC<0>                            0       conversion
//      4 ................. UNIMPLEMENTED ........... 0 ..... X
//      3 ................. UNIMPLEMENTED ........... 0 ..... X
//      2 ........ ASAM ..... A/D SAMPLE AUTO START ... 0 ..... Sampling begins when SAMP bit
//                                                              is set
//      1 ........ SAMP ..... A/D SAMPLE ENABLE ..... 0 ..... SAMP bit is auto set by ASAM
//                                                              setting
//      0 ........ DONE ..... A/D CONVERSION STATUS ... 0 ..... Automatically cleared and set
//                                                              by MCU
//-------------------------------------------------------------
//      ADCON2:         A/D     CONTROL REGISTER 2              ADCON2 = 0x0000
//-------------------------------------------------------------
//      BIT       NAME            FUNCTION                VALUE       DESCRIPTION
//-------------------------------------------------------------
//      15 ..... VCFG<2> ... VOLTAGE REF CONFIG ..... 0 ..... A/D VREFH = AVDD
//      14 ..... VCFG<1>                              0       A/D VREFL = AVSS
//      13 ..... VCFG<0>                              0
//      12 ................ RESERVED ................ 0
//      11 ................ UNIMPLEMENTED ........... 0 ..... X
//      10 ....... CSCNA .... SCAN INPUTS ............ 0 ..... Do not scan inputs
//      9 ................. UNIMPLEMENTED ........... 0 ..... X
//      8 ................. UNIMPLEMENTED ........... 0 ..... X
//      7 ........ BUFS ..... BUFFER FILL STATUS ..... 0 ..... Auto set (1 = A/D is filling
//                                                              0x8-0xF find data
//                                                              in 0x0-0x7; 0 = A/D is
//                                                              filling 0x0-0x7 find
//                                                              data in 0x8-0xF)
//      6 ................. UNIMPLEMENTED ........... 0 ..... X
//      5 ........ SMPI<3> .. SAMP-CONV SEQ/INTERRUPT . 0 ..... Interrupts at the completion
//      4 ........ SMPI<2>                            0       of the conversion sequence
//      3 ........ SMPI<1>                            0       for each sample/convert
//                                                              sequence
//      2 ........ SMPI<0>                            0
//      1 ........ BUFM ..... BUFFER MODE SELECT ..... 0 ..... Buffer configured as one 16-
//                                                              word buffer
//      0 ........ ALTS ..... ALTERNATE INPUT SAMPLE .. 0 ..... Always use MUX A input
//                                                              multiplexer settings
//-------------------------------------------------------------
//      ADCON3:         A/D     CONTROL REGISTER 3              ADCON3 = 0x0002
//-------------------------------------------------------------
//      BIT         NAME            FUNCTION                VALUE       DESCRIPTION
//-------------------------------------------------------------
//      15 ................ UNIMPLEMENTED ........... 0 ..... X
//      14 ................ UNIMPLEMENTED ........... 0 ..... X
//      13 ................ UNIMPLEMENTED ........... 0 ..... X
//      12 ....... SAMC<4> .. AUTO SAMPLE TIME ....... 0 ..... SAMC = 0Tad
//      11 ........SAMC<3>                            0
//      10 ........SAMC<2>                            0
//      9 .........SAMC<1>                            0
//      8 .........SAMC<0>                            0
//      7 ........ ADRC ..... A/D CONVERSION CLK SOURCE 0 ..... Clock derived from system clk
//      6 ................. UNIMPLEMENTED ........... 0 ..... X
//      5 ........ ADCS<5> .. A/D CONVERSION CLOCK .... 0 ..... TCY/2(ADCS<5:0> + 1) = 1.5TCY
```

```
//      4 ........ ADCS<4>                                  0
//      3 ........ ADCS<3>                                  0
//      2 ........ ADCS<2>                                  0
//      1 ........ ADCS<1>                                  1
//      0 ........ ADCS<0>                                  0
//-----------------------------------------------------------------------
//      ADCHS:          A/D     INPUT SELECT REGISTER          ADCHS = 0x0008
//-----------------------------------------------------------------------
//      BIT         NAME            FUNCTION            VALUE       DESCRIPTION
//-----------------------------------------------------------------------
//      15 ................... UNIMPLEMENTED ........... 0 ...... X
//      14 ................... UNIMPLEMENTED ........... 0 ...... X
//      13 ................... UNIMPLEMENTED ........... 0 ...... X
//      12 ....... CH0NB .... CH0- INPUT SELECT MUXB .. 0 ...... NOT USED
//      11 ....... CH0NB<3> . CH0+ INPUT SELECT MUXB .. 0 ...... NOT USED
//      10 ....... CH0NB<2>                             0
//      9 ........ CH0NB<1>                             0
//      8 ........ CH0NB<0>                             0
//      7 ................... UNIMPLEMENTED ........... 0 ...... X
//      6 ................... UNIMPLEMENTED ........... 0 ...... X
//      5 ................... UNIMPLEMENTED ........... 0 ...... X
//      4 ........ CH0NA .... CH0- INPUT SELECT MUXA .. 0 ...... NOT USED
//      3 ........ CH0NA<3> . CH0+ INPUT SELECT MUXA .. 1 ...... Channel 0 positive input is
//                                                              AN8
//      2 ........ CH0NA<2>                             0
//      1 ........ CH0NA<1>                             1
//      0 ........ CH0NA<0>                             1
//-----------------------------------------------------------------------
//      ADPCFC:         A/D     PORT CONFIGURATION REGISTER          ADPCFC = 0xFEFF
//-----------------------------------------------------------------------
//      BIT         NAME            FUNCTION            VALUE       DESCRIPTION
//-----------------------------------------------------------------------
//      15 ....... PCFC<15> . ANALOG INPUT PIN CONFIG . 1 ...... X
//      14 ....... PCFG<14>                             1 ...... X
//      13 ....... PCFG<13>                             1 ...... X
//      12 ....... PCFG<12>                             1 ...... X
//      11 ....... PCFG<11>                             1 ...... X
//      10 ....... PCFG<10>                             1 ...... X
//      9  ....... PCFG<9>                              1 ...... X
//      8 ........ PCFG<8>                              0 ...... TEMP_ADC
//      7 ........ PCFG<7>                              1 ...... X
//      6 ........ PCFG<6>                              1 ...... X
//      5 ........ PCFG<5>                              1 ...... X
//      4 ........ PCFG<4>                              1 ...... X
//      3 ........ PCFG<3>                              1 ...... X
//      2 ........ PCFG<2>                              1 ...... X
//      1 ........ PCFG<1>                              1 ...... PGC
//      0 ........ PCFG<0>                              1 ...... PGD
//-----------------------------------------------------------------------
//      ADCSSL:         A/D     INPUT SCAN SELECT REGISTER          ADCSSL = 0x0000
//-----------------------------------------------------------------------
//      BIT         NAME            FUNCTION            VALUE       DESCRIPTION
//-----------------------------------------------------------------------
//      15 ....... CSSL<15> . A/D INPUT PIN SCAN SELECT 0 ...... X
//      14 ....... CSSL<14>                             0 ...... X
//      13 ....... CSSL<13>                             0 ...... X
//      12 ....... CSSL<12>                             0 ...... X
//      11 ....... CSSL<11>                             0 ...... X
//      10 ....... CSSL<10>                             0 ...... X
//      9  ....... CSSL<9>                              0 ...... X
//      8 ........ CSSL<8>                              0 ...... TEMP_ADC
//      7 ........ CSSL<7>                              0 ...... X
//      6 ........ CSSL<6>                              0 ...... X
//      5 ........ CSSL<5>                              0 ...... X
//      4 ........ CSSL<4>                              0 ...... X
//      3 ........ CSSL<3>                              0 ...... X
//      2 ........ CSSL<2>                              0 ...... X
//      1 ........ CSSL<1>                              0 ...... PGC
//      0 ........ CSSL<0>                              0 ...... PGD
//-----------------------------------------------------------------------

void            InitADC12(void)
{
        ADCON1 = 0x0000;
        ADCON2 = 0x0000;
        ADCON3 = 0x0002;
```

```
        ADCHS  = 0x0008;
        ADPCFG = 0xFEFF;
        ADCSSL = 0x0000;

        ADCON1bits.ADON = 1;          //      Turn ADC ON
}

//=====================================================================
//      PUBLIC FUNCTION:        SendtoUART      (Adapted from D'Souza, S., 2003) [68].
//=====================================================================
//      FUNCTION CALL:          SendtoUART();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:    This routine transmits the data acquired from the ADC
//                      via UART1 to the PC running Hyperterminal
//---------------------------------------------------------------------

void    SendtoUART(void)
{
        while (!U1STAbits.TRMT)                         //      Wait until the TSR is empty
        {
                /* NULL STATEMENT */;
        }

        U1STAbits.UTXEN = 0;                            //      Disable transmission

        ADCValue = (ADCValue - 0x026C)/0x000C;          //      T = (Vo - 500mV)/10mV

        asciiDecConv(ADCValue);

        OutData[0] = thousands;
        OutData[1] = hundreds;
        OutData[2] = tens;
        OutData[3] = units;

        txCount         = 0;                            //      Sets counter to 0
        IFS0bits.U1TXIF = 0;                            //      Clear TX flag
        IEC0bits.U1TXIE = 1;                            //      Enable interrupt

        U1STA                   = 0x0000;               //      Clear all status bits
        U1STAbits.UTXEN = 1;                            //      Start transmission
}

//=====================================================================
//      PUBLIC FUNCTION:        asciiDecConv
//=====================================================================
//      FUNCTION CALL:          asciiDecConv(const unsigned int value);
//      ARGUMENTS:              Integer value
//      RETURNS:                Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:    This function converts a 12-bit binary word to a
//                      BCD ascii decimal value
//---------------------------------------------------------------------

void    asciiDecConv(const unsigned int value)
{
        int unitsInt        = 0;                        //      Initialise local variables
        int     tensInt         = 0;
        int     hundredsInt     = 0;
        int     thousandsInt    = 0;

        int n               = value;

    if (n >= 1000)                                      //      If there are 1000s
    {
        thousandsInt    = n/1000;
        thousands       = asciiConv(thousandsInt);      //      convert to an ascii char
        n               = n%1000;           // the remainder is ready for more conversion
    }
    if (n >= 100)                                       //      If there are 100s
    {
        hundredsInt     = n/100;
        hundreds        = asciiConv(hundredsInt);       //      convert to an ascii char
        n               = n%100;            //  the remainder is ready for more conversion
    }
    if (n >= 10)                                        //      If there are 10s
```

```
    {
        tensInt         = n/10;
        tens            = asciiConv(tensInt);       //      convert to an ascii char
        n               = n%10;         //      the remainder is ready for more conversion
    }
    if (value >= 1)                                                                     //
        If there are 1s
    {
        unitsInt        = n/1;
        units           = asciiConv(unitsInt);      //      convert to an ascii char
        n               = n%1;          //      the remainder is ready for more conversion
    }
}

//=====================================================================
//      PUBLIC FUNCTION:        asciiConv
//=====================================================================
//      FUNCTION CALL:          asciiConv(const unsigned int value);
//      ARGUMENTS:              Integer Value
//      RETURNS:                Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:    This function converts a 16-bit value to an 8-bit
//                      ascii character
//---------------------------------------------------------------------

char    asciiConv(const unsigned int value)
{
        char no;

        no = (char)(value & 0x0F);                      //      Only select bits 0-3

        if      (no     > 9)                            //      If number is > 9
        {
                no = no + 0x37;                         //  Number will be (A = 0x41, B = 0x42, C
                                                        //      = 0x43 etc )
        }
        else
        {
                no = no + 0x30;                         //      Number will be 0 - 9
        }

        return no;
}

//=====================================================================
//      PUBLIC FUNCTION:        DelayNmSec      (D'Souza, S., 2003) [68].
//=====================================================================
//      FUNCTION CALL:          DelayNmSec(unsigned int N);
//      ARGUMENTS:              Number of millisecond N
//      RETURNS:                Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:    This function creates a 1ms to 65.5ms delay. For
//                      a 1ms delay N=1 for a 30ms delay N=30 etc
//---------------------------------------------------------------------
void DelayNmSec(unsigned int N)
{
         unsigned int j;                        //  Counter j

        while(N--)                              //  Do while until N = 0
        {
                for(j=0;j < MILLISEC;j++)       //  This creates a 1 mS delay
                {
                        /* NULL STATEMENT */;
                }
        }
}
```

```
//=====================================================================
//      INTERRUPT SERVICE ROUTINE (ISR):      _U1TXInterrupt (D'Souza, S., 2003) [REF******].
//=====================================================================
//      DESCRIPTION:   This ISR clears the interrupt has occured flag and
//                                  then transmits a word of data
//---------------------------------------------------------------------
void    __attribute__((__interrupt__)) _U1TXInterrupt(void)
{
        IFS0bits.U1TXIF = 0;                    //      Clear transmit interrupt has occured flag

        U1TXREG = (int)OutData[txCount++];              //      Write a single word to UART

        if(txCount == TX_BUFFERSIZE)
        {
                IEC0bits.U1TXIE = 0;                    //      Disable transmit interrupts since
                                                        //      all buffer values are transmitted
        }

}
//=====================================================================
```

## *Memory Testing – TestMEMORYSTATUS.c*

```
//=====================================================================
//      AT45DB321B 32-MBIT DATAFLASH STATUS REGISTER TESTING
//=====================================================================
//      BY:             ROBYN VERRINDER
//                      Parts of this program were adapted from the source code
//                      to test the validity of the dataflash chip
//                      (Stowe, G., 2006) [65].
//      DATE:           10 MAY 2006
//=====================================================================
//      PROCESSOR:      dsPIC30F6014A
//      BOARD:          VIBRATION DATA LOGGING PROTOTYPE BOARD
//      IDE:            MPLAB IDE v7.30
//      COMPILER:       C30 C COMPILER
//=====================================================================
//      DESCRIPTION:    This program tests the Atmel 32-Mbit Dataflash,
//                      AT45DB321B (U9) onboard the Vibration Data Logging
//                      Prototype Board. The dataflash chip is connected
//                      to the dsPIC30F6014A via one of the Serial Peripheral
//                      Interface (SPI1) modules on board the MCU.
//
//                      The AT45DB321B has an onboard status register. The
//                      contents of this register are sent to the dsPIC30F6014A
//                      and are compared to the expected value[65]
//                      If the Dataflash chip is intact LED2 is switch ON if it is
//                      not intact LED4 is switched ON.
//
//                      Parts of this code were adapted from code written for
//                      SPI2 module on board the dsPIC30Fxxxx chip which was attached
//                      to a Atmel 64-Mbit Dataflash chip AT45DB642 [65].
//                      The DataflashWorking routine was written by Stowe, G. while
//                      the SendReceiveByte,InitSPI1 and StatusofDataflash were adapted
//                      from his code. Main and InitPorts routines were written by Robyn
//                      Verrinder.
//
//                      Register descriptions are as in dsPIC30F Family Reference Manual
//                      Section 19: UART and Section 20: Serial Peripheral Interface (SPI)
//                      [63].
//=====================================================================
//      U9:  AT45DB321B =       32-Mbit Serial Data Flash
//=====================================================================
//      CHIP:           AT45DB321B 32-megabit DataFlash Chip
//      INTERFACE:      SPI (Serial Peripheral Interface)
//=====================================================================
//      MEMORY SIZE:    32 MbitS
//      BYTES/PAGE:     528 bytes/page
//      PAGES/BLOCK:    8 blocks/page
//      NO OF PAGES:    8192 pages
//      NO OF BLOCKS:   1024 blocks
//=====================================================================
//      PIN OUTS:
//      1       =       GND ....... GROUND ............... SGND
//      2       =       NC ........ NOT CONNECTED......... X
//      3       =       NC ........ NOT CONNECTED ........ X
//      4       =       CS ........ CHIP SELECT .......... LOW = Selected, HIGH = Deselected
//      5       =       SCK ....... SERIAL CLOCK ......... Can be clocked up to 20Mhz
//      6       =       SI ........ SERIAL INPUT ......... Input only, used to shift data into
//                                                         device
//      7       =       SO ........ SERIAL OUTPUT ........ Output only, used to shift data out
//                                                         of device
//      8       =       NC ........ NOT CONNECTED ........ X
//      9       =       NC ........ NOT CONNECTED ........ X
//      10      =       NC ........ NOT CONNECTED ........ X
//      11      =       NC ........ NOT CONNECTED ........ X
//      12      =       NC ........ NOT CONNECTED ........ X
//      13      =       NC ........ NOT CONNECTED ........ X
//      14      =       NC ........ NOT CONNECTED ........ X
//      15      =       NC ........ NOT CONNECTED ........ X
//      16      =       NC ........ NOT CONNECTED ........ X
//      17      =       NC ........ NOT CONNECTED ........ X
//      18      =       NC ........ NOT CONNECTED ........ X
//      19      =       NC ........ NOT CONNECTED ........ X
//      20      =       NC ........ NOT CONNECTED ........ X
```

```
//      21      =       NC ........ NOT CONNECTED ........ X
//      22      =       NC ........ NOT CONNECTED ........ X
//      23      =       RDY/BUSY .. READY/BUSY ........... Output pin - LOW = Device is BUSY
//      24      =       RESET ..... CHIP RESET ........... LOW = reset chip, HIGH = normal
//                                                         operation
//      25      =       WP ........ HARDWARE PAGE WRITE PROTECT LOW = cannot reprogramme 1st
//                                                                 256 pages of main mem, HIGH =
//                                                                 can programme
//      26      =       NC ........ NOT CONNECTED ........ X
//      27      =       NC ........ NOT CONNECTED ........ X
//      28      =       VCC ....... POWER SUPPLY ......... VDD
//
//      PIN CONNECTIONS: (AT45DB321B to dsPIC30F6014A)
//       VCC      =  VDD
//       GND      =  SGND
//       SCK      =  SCK1 (PIN 45)
//       SI       =  SDO1 (PIN 43)
//       SO       =  SDI1 (PIN 44)
//       CS       =  RD8  (PIN 54)
//       RDY/BUSY =  RD9  (PIN 55)
//       RESET    =  RD10 (PIN 56)
//       WP       =  RD11 (PIN 57)
//-------------------------------------------------------------------
//      AT45DB321B STATUS REGISTER - 8 BIT  = 0xB4
//-------------------------------------------------------------------
//      BIT7    =       RDY/BUSY ............... LOW = busy, HIGH = not busy
//      BIT6    =       COMP ................... LOW = data in main memory matches buffer, HIGH
//                                                    = at least 1 bit of data in main memory
//                                                    does not match data in buffer
//      BIT5    =       1 ...................... Bits 5-2 = device density
//      BIT4    =       1
//      BIT3    =       0
//      BIT2    =       1
//      BIT1    =       X
//      BIT0    =       X
//-------------------------------------------------------------------
#include                "p30f6014a.h"           //      (Sinha, P., 2005) [64].
#include                "dsPIC30F6014APins.h"
#include                "AT45DB321B.h"
#include                "common.h"
//-------------------------------------------------------------------
//      dsPIC30F6014A CONFIGURATION SETTINGS
//-------------------------------------------------------------------
//      CLOCK SWITCHING AND MONITOR ............... Sw Disabled, Mon Disabled
//      OSCILLATOR ................................ XT w/PLL 8x
//      WATCHDOG TIMER ............................ Disabled
//      MASTER CLEAR ENABLE ....................... Enabled
//      POR TIMER VALUE ........................... 64ms
//      GENERAL CODE SEGMENT CODE PROTECT ......... Disabled
//-------------------------------------------------------------------
_FOSC(CSW_FSCM_OFF & XT_PLL8);
_FWDT(WDT_OFF);
_FBORPOR(MCLR_EN & PWRT_64);
_FGS(CODE_PROT_OFF);
//-------------------------------------------------------------------
//      FUNCTION DEFINITIONS
//-------------------------------------------------------------------
void    InitSPI1(void);
void    InitPorts(void);

unsigned char  SendReceiveByte(unsigned char data);

unsigned char  StatusofDataflash(void);
unsigned int   DataflashWorking(void);

//-------------------------------------------------------------------
//      GLOBAL VARIABLES
//-------------------------------------------------------------------

unsigned char  dataflash_is_working;                //      Dataflash is working flag

//-------------------------------------------------------------------
//      MAIN FUNCTION
//-------------------------------------------------------------------
```

```
int     main(void)
{

        InitPorts();            //      Initialises Ports
        InitSPI1();             //      Initialises the SPI1 module

        dataflash_is_working = DataflashWorking();   //      Tests to see if the status
                                                     //      register value is correct

        while(1)                                     //      Do forever
        {
                RESET   =       1;                   //      Normal Operation


                if      (dataflash_is_working==1)    //      If the dataflash chip is working
                {
                        LED2 = 1;                    //      Turn LED2 ON
                }
                else if (dataflash_is_working==0)    //      If the dataflash chip is NOT
                                                     //      working
                {
                        LED4 = 1;                    //      Turn LED4 ON
                }
        }
}
//=============================================================
//      PUBLIC FUNCTION:        InitPorts
//=============================================================
//      FUNCTION CALL: InitPorts();
//      ARGUMENTS:      None
//      RETURNS:        Nothing
//-------------------------------------------------------------
//      DESCRIPTION:    Initialises the LED ports as OUTPUTs The values of all the registers
//                      are laid out below
//-------------------------------------------------------------
//      PORT D:         PIN DEFINITIONS                 TRISD  = 0x020E         LATD = 0x0100
//-------------------------------------------------------------
//              PIN                             USE             TRISD                   LATD
//      -----------------------------------------------------
//              RD0     ...........     OUTPUT ..... 0 ..... 0 ..... X
//              RD1     ...........     INPUT ..... 1 ..... 0 ..... SEND
//              RD2     ...........     INPUT ..... 1 ..... 0 ..... RECORD
//              RD3     ...........     INPUT ..... 1 ..... 0 ..... ERASE
//              RD4     ...........     OUTPUT ..... 0 ..... 0 ..... X
//              RD5     ...........     OUTPUT ..... 0 ..... 0 ..... X
//              RD6     ...........     OUTPUT ..... 0 ..... 0 ..... X
//              RD7     ...........     OUTPUT ..... 0 ..... 0 ..... X
//              RD8     ...........     OUTPUT ..... 0 ..... 1 ..... CS
//              RD9     ...........     INPUT ..... 1 ..... 0 ..... R_B
//              RD10    ...........     OUTPUT ..... 0 ..... 0 ..... RESET
//              RD11    ...........     OUTPUT ..... 0 ..... 0 ..... WP
//              RD12    ...........     OUTPUT ..... 0 ..... 0 ..... X
//              RD13    ...........     OUTPUT ..... 0 ..... 0 ..... X
//              RD14    ...........     OUTPUT ..... 0 ..... 0 ..... X
//              RD15    ...........     OUTPUT ..... 0 ..... 0 ..... X
//-------------------------------------------------------------
//      PORT F:         PIN DEFINITIONS         TRISF  = 0x0084  LATF = 0x0000
//-------------------------------------------------------------
//              PIN                             USE             TRISF           LATF
//      -----------------------------------------------------
//              RF0     ...........     OUTPUT ..... 0 ..... 0 ..... X
//              RF1     ...........     OUTPUT ..... 0 ..... 0 ..... X
//              RF2     ...........     INPUT ..... 1 ..... 0 ..... RX_OUT
//              RF3     ...........     OUTPUT ..... 0 ..... 0 ..... TX_IN
//              RF4     ...........     OUTPUT ..... 0 ..... 0 ..... X
//              RF5     ...........     OUTPUT ..... 0 ..... 0 ..... X
//              RF6     ...........     OUTPUT ..... 0 ..... 0 ..... SCK
//              RF7     ...........     INPUT ..... 1 ..... 0 ..... SO
//              RF8     ...........     OUTPUT ..... 0 ..... 0 ..... SI
//              RF9     ...........     NOT I/O ..... 0 ..... 0 ..... X
//              RF10    ...........     NOT I/O ..... 0 ..... 0 ..... X
//              RF11    ...........     NOT I/O ..... 0 ..... 0 ..... X
//              RF12    ...........     NOT I/O ..... 0 ..... 0 ..... X
//              RF13    ...........     NOT I/O ..... 0 ..... 0 ..... X
//              RF14    ...........     NOT I/O ..... 0 ..... 0 ..... X
//              RF15    ...........     NOT I/O ..... 0 ..... 0 ..... X
//-------------------------------------------------------------
```

```
void    InitPorts(void)
{
        TRISD   =       0x020E;                 //      Port D I/O direction
        TRISF   =       0x0084;                 //      Port F I/O direction
        TRISC   =       0x0000;                 //      Port C I/O direction
        TRISG   =       0x0000;                 //      Port G I/O direction

        PORTC   =       0x0000;                 //      Port C pins levels set to LOW
        PORTG   =       0x0000;                 //      Port G pins levels set to LOW
        PORTD   =       0x0F00;                 //      Port D latch settings set to 0
                                                //      (except CS, WP, RESET and
                                                //      READY/BUSY)
        PORTF   =       0x0000;                 //      Port F latch settings set to 0
}
//======================================================================
//      PUBLIC FUNCTION:        InitSPI1
//======================================================================
//      FUNCTION CALL:          InitSPI1();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//----------------------------------------------------------------------
//      DESCRIPTION:            Initialises the SPI1 module on the dsPIC30F6014A
//                              The values of all the registers are laid out below
//      FSCLK:                  FCY/(PRIMARY PRESCALE x SECONDARY PRESCALE)
//                              1.67MHz
//----------------------------------------------------------------------
//      SPI1CON:        SPI1 CONTROL REGISTER           SPI1CON = 0x0020
//----------------------------------------------------------------------
//      BIT         NAME            FUNCTION            VALUE       DESCRIPTION
//----------------------------------------------------------------------
//      15 ..... UNIMPLEMENTED ..... 0 ..... X
//      14 ..... FRMEN .... FRAMED SPI SUPPORT ..... 0 ..... Framed SPI support DISABLED
//      13 ..... SPIFSD ... FRAME SYNC PUL DIRECTION  0 ..... Frame sync pulse output
//                                                            (Master)
//      12 ..... UNIMPLEMENTED ..... 0 ..... X
//      11 ..... DISSDO ... DISABLE SDO1 PIN ..... 0 ..... SDO1 pin is controlled by the
//                                                            module
//      10 ..... MODE16 ... WORD/BYTE COMMS ..... 0 ..... Communication is 8 bits wide
//      9 ..... SMP ..... SPI DATA INPUT SAMP PHASE 0 ..... Input data sampled at middle
//                                                            of data output time
//      8 ..... CKE ..... SPI CLOCK EDGE SELECT ... 0 ..... Serial output data changes on
//                                                            transition from IDLE CLK to
//                                                            ACTIVE CLK (SPIMODE0)
//      7 ..... SSEN ..... SLAVE SELECT ENABLE ..... 0 ..... SS pin not used by module.
//      6 ..... CKP ..... SPI CLOCK POLARITY SELECT 0 ..... IDLE state for CLK is a HIGH
//      5 ..... MSTEN .... MASTER MODE ENABLE ..... 1 ..... Master mode
//      4 ..... SPRE<2> .. SECONDARY PRESCALE ..... 0 ..... Secondary Prescale 8:1
//      3 ..... SPRE<1>                             0
//      2 ..... SPRE<0>                             0
//      1 ..... PPRE<1>... PRIMARY PRESCALE ..... 0 ..... Primary Prescale 64:1
//      0 ..... PPRE<0>                             0
//----------------------------------------------------------------------
//      SPI1STAT:       SPI STATUS AND CONTROL REGISTER       SPI1STAT = 0x8000
//----------------------------------------------------------------------
//      BIT                 NAME            FUNCTION                VALUE       DESCRIPTION
//----------------------------------------------------------------------
//      15 ..... SPIEN .... SPI ENABLE ..... 1 ..... SPI1 is DISABLED
//      14 ..... UNIMPLEMENTED ..... 0 ..... X
//      13 ..... SPISIDL .. STOP IN IDLE MODE ..... 0 ..... Operates in IDLE mode
//      12 ..... UNIMPLEMENTED ..... 0 ..... X
//      11 ..... UNIMPLEMENTED ..... 0 ..... X
//      10 ..... UNIMPLEMENTED ..... 0 ..... X
//      9 ..... UNIMPLEMENTED ..... 0 ..... X
//      8 ..... UNIMPLEMENTED ..... 0 ..... X
//      7 ..... UNIMPLEMENTED ..... 0 ..... X
//      6 ..... SPIROV ... RECEIVE OVERFLOW FLAG ... 0 ..... No overflow has occured(READ
//                                                            ONLY)
//      5 ..... UNIMPLEMENTED ..... 0 ..... X
//      4 ..... UNIMPLEMENTED ..... 0 ..... X
//      3 ..... UNIMPLEMENTED ..... 0 ..... X
//      2 ..... UNIMPLEMENTED ..... 0 ..... X
//      1 ..... SPITBF ... SPI TRANSMIT BUFFER FULL  0 ..... Transmit started, SPI1TXB is
//                                                            EMPTY (READ ONLY)
//      0 ..... SPIRBF ... SPI RECEIVE BUFFER FULL.. 0 ..... Receive is not complete,
//                                                            SPI1RXB is EMPTY (READ ONLY)
//----------------------------------------------------------------------
```

```
void    InitSPI1(void)
{
        CS      =       1;                      //      Disable AT45DB321B
        RESET   =       1;                      //      Normal Operation
        WP      =       1;                      //      Can write to Dataflash chip

        SPI1CON =       0x0020;
        SPI1STAT|=      0x8000;                 //      Enable device and leave status of other
                                                //      bits
        SPI1STAT&=      0xFFBF;                 //      SPIROV flag = 0; leaves all other bits
                                                //      the same

        IFS0bits.SPI1IF = 0;                    //      Clears SPI interrupt has occurred flag
        IEC0bits.SPI1IE = 0;                    //      Disables SPI interrupts
}

//=====================================================================
//      PUBLIC FUNCTION:        SendReceiveByte
//=====================================================================
//      FUNCTION CALL: SendReceiveByte(data);
//      ARGUMENTS:      unsigned char data
//      RETURNS:        unsigned char byte_received
//---------------------------------------------------------------------
//      DESCRIPTION:    Sends a 8bit byte to the AT45DB321B 32Mbit Data
//                      flash chip via the SPI data interface and returns
//                      the received byte.
//---------------------------------------------------------------------

unsigned char  SendReceiveByte(unsigned char data)
{

        SPI1BUF = data;                         //      Load 1 data byte into SPI1BUF -> SPI1TXB
                                                //      -> SPI1SR
        SPI1STAT &= 0xA003;                     //      Leaves SPIEN, SPITBF and SPIRBF bits same

        while (SPI1STATbits.SPITBF)             //      Wait until transfer is complete
        {
                /* NULL STATEMENT */;
        }
        while (!SPI1STATbits.SPIRBF)            //      Wait until receive is complete
        {
                /* NULL STATEMENT */;
        }

        return SPI1BUF;                         //      Returns the received byte

}
```

```
//=============================================================
//      PUBLIC FUNCTION:       StatusofDataflash
//=============================================================
//      FUNCTION CALL: StatusofDataflash();
//      ARGUMENTS:     None
//      RETURNS:       unsigned char status_register
//-------------------------------------------------------------
//      DESCRIPTION:   This function returns the value of status register
//                     on the Atmel 32-Mbit Dataflash AT45DB321B.
//-------------------------------------------------------------

unsigned char  StatusofDataflash(void)
{
       unsigned char status_register;              //     Value of the status regsister

       CS = 0;                                     //     ENABLE AT45DB321B Dataflash chip

       SendReceiveByte(CMD_READ_STATUSREG);        //     Send read status register command
       status_register = SendReceiveByte(0x00);    //     Dummy write to recieve status
                                                   //     register value

       CS = 1;                                     //     DISABLE AT45DB321B Dataflash chip

       return status_register;                     //     Returns the value of the status
                                                   //     register
}

//=============================================================
//      PUBLIC FUNCTION:       DataflashWorking
//=============================================================
//      FUNCTION CALL: DataflashWorking();
//      ARGUMENTS:     None
//      RETURNS:       unsigned char dataflash_working (either TRUE or FALSE)
//-------------------------------------------------------------
//      DESCRIPTION:   This function determines whether the value of the
//                     dataflash status register is the expected value or not
//-------------------------------------------------------------

unsigned int DataflashWorking(void)
{
       unsigned char status;                       //     Initialise local varible to store
                                                   //     status register contents
       unsigned int device_good;                   //     Initialise device is good flag

       status = StatusofDataflash();               //     Store value of status register
                                                   //     Selects the device density bits
                                                   //      32-Mbit = 1101

       if      (status == 0xB4)                    //     If device is working the status
                                                   //     register is 10110100
       {
              device_good = 1;
       }
       else
       {
              device_good = 0;
       }
       return device_good;
}

//-------------------------------------------------------------
```

## *Memory Testing – TestMEMORY.c*

```
//=====================================================================
//              AT45DB321B 32-MBIT DATAFLASH MEMORY TESTING
//=====================================================================
//      BY:             ROBYN VERRINDER
//                      Parts of this program were adapted from the source code
//                      to test the validity of the dataflash chip
//                      (Stowe, G., 2006) [65].
//
//      DATE:           10 MAY 2006
//=====================================================================
//      PROCESSOR:      dsPIC30F6014A
//      BOARD:          VIBRATION DATA LOGGING PROTOTYPE BOARD
//      IDE:            MPLAB IDE v7.30
//      COMPILER:       C30 C COMPILER
//=====================================================================
//      DESCRIPTION:    This program tests the Atmel 32-Mbit Dataflash,
//                      AT45DB321B (U9) onboard the Vibration Data Logging
//                      Prototype Board. The dataflash chip is connected
//                      to the dsPIC30F6014A via one of the Serial Peripheral
//                      Interface (SPI1) modules on board the MCU.
//
//                      A buffer containing 20 'A' characters is written
//                      to the dataflash. The data is sent back to the
//                      dsPIC30F6014A processor where it is compared to the
//                      sent value. If they are equal the write/read operation
//                      took place correctly and LED2 is turned ON. If they
//                      are not equal the write/read operation failed.
//
//                      Parts of this code were adapted from code written for
//                      SPI2 module on board the dsPIC30Fxxxx chip which was attached
//                      to a Atmel 64-Mbit Dataflash chip AT45DB642 [65].
//                      The InitSPI1, SendReceiveByte,WritetoBuffer ReadfromDataflash routines
//                      were adapted from code written by Stowe, G [65]. Main and InitPorts
//                      routines were written by Robyn Verrinder.
//
//                      Register descriptions are as in dsPIC30F Family Reference Manual
//                      Section 19: UART and Section 20: Serial Peripheral Interface (SPI)
//                      [63].
//=====================================================================
//      U9:  AT45DB321B =       32-Mbit Serial Data Flash
//=====================================================================
//      CHIP:           AT45DB321B 32-megabit DataFlash Chip
//      INTERFACE:      SPI (Serial Peripheral Interface)
//=====================================================================
//      MEMORY SIZE:    32 MbitS
//      BYTES/PAGE:     528 bytes/page
//      PAGES/BLOCK:    8 blocks/page
//      NO OF PAGES:    8192 pages
//      NO OF BLOCKS:   1024 blocks
//=====================================================================
#include                "p30f6014a.h"           //      (Sinha, P., 2005) [64].
#include                "dsPIC30F6014APins.h"
#include                "AT45DB321B.h"
#include                "common.h"
//---------------------------------------------------------------------
//      dsPIC30F6014A CONFIGURATION SETTINGS
//---------------------------------------------------------------------
//      CLOCK SWITCHING AND MONITOR ..... Sw Disabled, Mon Disabled
//      OSCILLATOR ...................... XT w/PLL 8x
//      WATCHDOG TIMER .................. Disabled
//      MASTER CLEAR ENABLE ............. Enabled
//      POR TIMER VALUE ................. 64ms
//      GENERAL CODE SEGMENT CODE PROTECT ..... Disabled
//---------------------------------------------------------------------
_FOSC(CSW_FSCM_OFF & XT_PLL8);
_FWDT(WDT_OFF);
_FBORPOR(MCLR_EN & PWRT_64);
_FGS(CODE_PROT_OFF);
```

```
//-------------------------------------------------------------------
//      FUNCTION DEFINITIONS
//-------------------------------------------------------------------
void    InitUART1(void);
void    InitSPI1(void);
void    InitPorts(void);

unsigned char  SendReceiveByte(unsigned char data);

void    EraseDataflash(void);
void    WritetoBuffer(unsigned char buffer, unsigned short address,
                      unsigned char data[], unsigned short length);
void    ReadfromBuffer(unsigned char buffer, unsigned short address,
                      unsigned char *returned_data, unsigned short length);
//-------------------------------------------------------------------
//      BUFFER CONSTANTS
//-------------------------------------------------------------------
#define         BUFFER1         0x01
#define         BUFFER2         0x02
//-------------------------------------------------------------------
//      GLOBAL FLAGS
//-------------------------------------------------------------------
unsigned int    mem_is_clear = 0;               //      Main memory is empty flag
unsigned int    mem_is_full     = 0;            //      Main memory is full flag
//-------------------------------------------------------------------
//      GLOBAL VARIABLES
//-------------------------------------------------------------------
unsigned short length = 20;                     //      Length of data array
unsigned short address= 0x0000;                 //      Address in buffer

unsigned char  d[20];                           //      Value to be transmitted
unsigned char  read_data[20];                   //      Read value

unsigned char *rdptr;                           //      Pointer to Read Values
//-------------------------------------------------------------------
//      MAIN FUNCTION
//-------------------------------------------------------------------
int     main(void)
{
        InitPorts();                            //      Initialises Ports
        InitSPI1();                             //      Initialises the SPI1 module
        EraseDataflash();                       //      Erase the Dataflash Chip

        int i;                                  //      Initialise counter i

        for (i=0;i<length;i++)
        {
                d[i] = 'A';
        }

        rdptr = &read_data[0];

        unsigned char  x = d[10];

        WritetoBuffer(BUFFER1,address,d,length);
        ReadfromBuffer(BUFFER1,address,*rdptr,length);

        unsigned char  y = read_data[10];

        while(!READY_BUSY)                      //      Check to see if chip is busy
        {
                /* NULL STATEMENT */;
        }

        while(1)
        {
                if (y == x)
                {
                        LED2 = 1;
                }
                else
                {
                        LED4 = 1;
                }
        }
}
```

```
//=====================================================================
//      PUBLIC FUNCTION:        InitPorts
//=====================================================================
//      FUNCTION CALL: InitPorts();
//      ARGUMENTS:      None
//      RETURNS:        Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:    Initialises the LED ports as OUTPUTs The values of all the registers
//                      are laid out below
//---------------------------------------------------------------------
//      PORT D:         PIN DEFINITIONS                 TRISD  = 0x020E         LATD = 0x0100
//---------------------------------------------------------------------
//              PIN                             USE                 TRISD                   LATD
//      ------------------------------------------------------------
//              RD0     ..........      OUTPUT ..... 0 ..... 0 ..... X
//              RD1     ..........      INPUT ..... 1 ..... 0 ..... SEND
//              RD2     ..........      INPUT ..... 1 ..... 0 ..... RECORD
//              RD3     ..........      INPUT ..... 1 ..... 0 ..... ERASE
//              RD4     ..........      OUTPUT ..... 0 ..... 0 ..... X
//              RD5     ..........      OUTPUT ..... 0 ..... 0 ..... X
//              RD6     ..........      OUTPUT ..... 0 ..... 0 ..... X
//              RD7     ..........      OUTPUT ..... 0 ..... 0 ..... X
//              RD8     ..........      OUTPUT ..... 0 ..... 1 ..... CS
//              RD9     ..........      INPUT ..... 1 ..... 0 ..... R_B
//              RD10    ..........      OUTPUT ..... 0 ..... 0 ..... RESET
//              RD11    ..........      OUTPUT ..... 0 ..... 0 ..... WP
//              RD12    ..........      OUTPUT ..... 0 ..... 0 ..... X
//              RD13    ..........      OUTPUT ..... 0 ..... 0 ..... X
//              RD14    ..........      OUTPUT ..... 0 ..... 0 ..... X
//              RD15    ..........      OUTPUT ..... 0 ..... 0 ..... X
//---------------------------------------------------------------------
//      PORT F:         PIN DEFINITIONS         TRISF  = 0x0084  LATF = 0x0000
//---------------------------------------------------------------------
//              PIN                     USE             TRISF           LATF
//      ------------------------------------------------------------
//              RF0     ..........      OUTPUT ..... 0 ..... 0 ..... X
//              RF1     ..........      OUTPUT ..... 0 ..... 0 ..... X
//              RF2     ..........      INPUT ..... 1 ..... 0 ..... RX_OUT
//              RF3     ..........      OUTPUT ..... 0 ..... 0 ..... TX_IN
//              RF4     ..........      OUTPUT ..... 0 ..... 0 ..... X
//              RF5     ..........      OUTPUT ..... 0 ..... 0 ..... X
//              RF6     ..........      OUTPUT ..... 0 ..... 0 ..... SCK
//              RF7     ..........      INPUT ..... 1 ..... 0 ..... SO
//              RF8     ..........      OUTPUT ..... 0 ..... 0 ..... SI
//              RF9     ..........      NOT I/O ..... 0 ..... 0 ..... X
//              RF10    ..........      NOT I/O ..... 0 ..... 0 ..... X
//              RF11    ..........      NOT I/O ..... 0 ..... 0 ..... X
//              RF12    ..........      NOT I/O ..... 0 ..... 0 ..... X
//              RF13    ..........      NOT I/O ..... 0 ..... 0 ..... X
//              RF14    ..........      NOT I/O ..... 0 ..... 0 ..... X
//              RF15    ..........      NOT I/O ..... 0 ..... 0 ..... X
//---------------------------------------------------------------------
void    InitPorts(void)
{
        TRISD   =       0x020E;                 //      Port D I/O direction
        TRISF   =       0x0084;                 //      Port F I/O direction
        TRISC   =       0x0000;                 //      Port C I/O direction
        TRISG   =       0x0000;                 //      Port G I/O direction

        PORTC   =       0x0000;                 //      Port C pins levels set to LOW
        PORTG   =       0x0000;                 //      Port G pins levels set to LOW
        PORTD   =       0x0F00;                 //      Port D latch settings set to 0
                                                //      (except CS, WP, RESET and
                                                //      READY/BUSY)
        PORTF   =       0x0000;                 //      Port F latch settings set to 0
}
//=====================================================================
//      PUBLIC FUNCTION:        InitSPI1
//=====================================================================
//      FUNCTION CALL:          InitSPI1();
//      ARGUMENTS:              None
//      RETURNS:                Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:            Initialises the SPI1 module on the dsPIC30F6014A
//                              The values of all the registers are laid out below
//      FSCLK:                  FCY/(PRIMARY PRESCALE x SECONDARY PRESCALE)
//                              1.67MHz
```

```
//-------------------------------------------------------------------
//      SPI1CON:        SPI1 CONTROL REGISTER          SPI1CON = 0x0020
//-------------------------------------------------------------------
//      BIT         NAME            FUNCTION            VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ................. UNIMPLEMENTED ........... 0 ...... X
//      14 ....... FRMEN .... FRAMED SPI SUPPORT ..... 0 ...... Framed SPI support DISABLED
//      13 ....... SPIFSD ... FRAME SYNC PUL DIRECTION  0 ...... Frame sync pulse output
//                                                              (Master)
//      12 ................. UNIMPLEMENTED ........... 0 ...... X
//      11 ....... DISSDO ... DISABLE SDO1 PIN ........ 0 ...... SDO1 pin is controlled by the
//                                                               module
//      10 ....... MODE16 ... WORD/BYTE COMMS ......... 0 ...... Communication is 8 bits wide
//      9 ........ SMP ...... SPI DATA INPUT SAMP PHASE 0 ...... Input data sampled at middle
//                                                               of data output time
//      8 ........ CKE ...... SPI CLOCK EDGE SELECT ... 0 ...... Serial output data changes on
//                                                              transition from IDLE CLK to
//                                                              ACTIVE CLK (SPIMODE0)
//      7 ........ SSEN ..... SLAVE SELECT ENABLE ..... 0 ...... SS pin not used by module.
//      6 ........ CKP ...... SPI CLOCK POLARITY SELECT 0 ...... IDLE state for CLK is a HIGH
//      5 ........ MSTEN .... MASTER MODE ENABLE ...... 1 ...... Master mode
//      4 ........ SPRE<2> .. SECONDARY PRESCALE ...... 0 ...... Secondary Prescale 8:1
//      3 ........ SPRE<1>                              0
//      2 ........ SPRE<0>                              0
//      1 ........ PPRE<1>... PRIMARY PRESCALE ........ 0 ...... Primary Prescale 64:1
//      0 ........ PPRE<0>                              0
//-------------------------------------------------------------------
//      SPI1STAT:       SPI STATUS AND CONTROL REGISTER       SPI1STAT = 0x8000
//-------------------------------------------------------------------
//      BIT               NAME          FUNCTION                  VALUE       DESCRIPTION
//-------------------------------------------------------------------
//      15 ....... SPIEN .... SPI ENABLE ............... 1 ...... SPI1 is DISABLED
//      14 ................. UNIMPLEMENTED ........... 0 ...... X
//      13 ....... SPISIDL .. STOP IN IDLE MODE ....... 0 ...... Operates in IDLE mode
//      12 ................. UNIMPLEMENTED ........... 0 ...... X
//      11 ................. UNIMPLEMENTED ........... 0 ...... X
//      10 ................. UNIMPLEMENTED ........... 0 ...... X
//      9 .................. UNIMPLEMENTED ........... 0 ...... X
//      8 .................. UNIMPLEMENTED ........... 0 ...... X
//      7 .................. UNIMPLEMENTED ........... 0 ...... X
//      6 ........ SPIROV ... RECEIVE OVERFLOW FLAG ... 0 ...... No overflow has occured(READ
//                                                              ONLY)
//      5 .................. UNIMPLEMENTED ........... 0 ...... X
//      4 .................. UNIMPLEMENTED ........... 0 ...... X
//      3 .................. UNIMPLEMENTED ........... 0 ...... X
//      2 .................. UNIMPLEMENTED ........... 0 ...... X
//      1 ........ SPITBF ... SPI TRANSMIT BUFFER FULL  0 ...... Transmit started, SPI1TXB is
//                                                               EMPTY (READ ONLY)
//      0 ........ SPIRBF ... SPI RECEIVE BUFFER FULL.. 0 ...... Receive is not complete,
//                                                               SPI1RXB is EMPTY (READ ONLY)
//-------------------------------------------------------------------
void    InitSPI1(void)
{
        CS      =       1;                      //      Disable AT45DB321B
        RESET   =       1;                      //      Normal Operation
        WP      =       1;                      //      Can write to Dataflash chip

        SPI1CON =       0x0020;
        SPI1STAT|=      0x8000;                 //      Enable device and leave status of other
                                                //      bits
        SPI1STAT&=      0xFFBF;                 //      SPIROV flag = 0; leaves all other bits
                                                //      the same
        IFS0bits.SPI1IF = 0;                    //      Clears SPI interrupt has occurred flag
        IEC0bits.SPI1IE = 0;                    //      Disables SPI interrupts
)
```

```
//=====================================================================
//      PUBLIC FUNCTION:       SendReceiveByte
//=====================================================================
//      FUNCTION CALL: SendReceiveByte(data);
//      ARGUMENTS:     unsigned char data
//      RETURNS:       unsigned char byte_received
//---------------------------------------------------------------------
//      DESCRIPTION:   Sends a 8bit byte to the AT45DB321B 32Mbit Data
//                     flash chip via the SPI data interface and returns
//                     the received byte.
//---------------------------------------------------------------------

unsigned char  SendReceiveByte(unsigned char data)
{

       SPI1BUF = data;                         //      Load 1 data byte into SPI1BUF -> SPI1TXB
                                               //      -> SPI1SR
       SPI1STAT &= 0xA003;                     //      Leaves SPIEN, SPITBF and SPIRBF bits same

       while (SPI1STATbits.SPITBF)             //      Wait until transfer is complete
       {
              /* NULL STATEMENT */;
       }
       while (!SPI1STATbits.SPIRBF)            //      Wait until receive is complete
       {
              /* NULL STATEMENT */;
       }

       return SPI1BUF;                         //      Returns the received byte

}
```

```
//=====================================================================
//      PUBLIC FUNCTION:       Erase
//=====================================================================
//      FUNCTION CALL: Erase();
//      ARGUMENTS:     None
//      RETURNS:       Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:   Erases entire data flash, block by block by making
//                     all values 1
//
//      ERASE INSTRUCTION FORMAT:
//                     CCCC CCCC RRAA AAAA AAAX XXXX XXXX XXXX
//                     8bit command, 2bits reserved, 9bit address, 13 don't care bits
//
//      MEMORY SIZE:   32 MbitS
//      BYTES/PAGE:    528
//      PAGES/BLOCK:   8
//      NO OF PAGES:   8192
//      NO OF BLOCKS:  1024
//---------------------------------------------------------------------

void    Erase(void)
{
        ERASE_LED = 1;                                  //      Turn erase led ON

        static unsigned int block_counter = 0;          //      Number of blocks erased counter

        mem_is_clear = 1;                               //      Set main memory is empty flag
        mem_is_full  = 0;                               //      Clear main memory is full flag

        while(block_counter < 1024)                     //      Do until all the pages have been
                                                        //      erased
        {
                CS = 0;                                 //      ENABLE AT45DB321B Dataflash chip

                SendReceiveByte(CMD_ERASE_BLOCK);       //      Send erase block command
                SendReceiveByte((char)(block_counter >> 3)); //      Puts address in correct
                                                        //      erase instruction
                                                        //      format
                SendReceiveByte((char)(block_counter << 5));
                SendReceiveByte(0x00);                  //      Don't care bits

                CS = 1;                                 //      DISABLE AT45DB321B Dataflash chip

                block_counter++;                        //      Increment block counter

                while(!READY_BUSY)
                {
                        /* NULL STATEMENT */;           //      Wait until the chip has completed
                                                        //      the erase
                }
        }

        ERASE_LED = 0;                                  //      Turn erase led OFF
}
```

```
//=====================================================================
//      PUBLIC FUNCTION:       WritetoBuffer
//=====================================================================
//      FUNCTION CALL: WritetoBuffer(buffer, address, *data, offset, length);
//      ARGUMENTS:     unsigned char buffer
//                     unsigned short address
//                     unsigned char *data
//                     unsigned short offset
//                     unsigned short length
//      RETURNS:       Nothing
//---------------------------------------------------------------------
//      DESCRIPTION:   Writes a data from an offset in a data array of a
//                     specified length to the AT45DB321B 32Mbit Data Flash
//                     Chip's buffer at an address
//
//      WRITE TO BUFFER COMMAND FORMAT:
//                     CCCC CCCC XXXX XXXX XXXX XXAA AAAA AAAA DDDD DDDD
//                     8bit command, 14bits dont care, 10bit address, 8bits of data
//
//---------------------------------------------------------------------

void    WritetoBuffer(unsigned char buffer, unsigned short address,
                              unsigned char data[], unsigned short length)
{
        RECORD_LED = 1;                                 //      Turn ON the Record led

        unsigned int i;
        while(!READY_BUSY)                              //      Check to see if chip is busy
        {
                /* NULL STATEMENT */;
        }

//      WRITES DATA TO BUFFER 1

        CS = 0;                                         //      ENABLE AT45DB321B Dataflash chip

        if      (buffer = BUFFER1)                      //      If Buffer is 1
        {
                SendReceiveByte(CMD_WRITE_BUFFER1);     //      Then send write to buffer 1
                                                        //      command

        }
        else
        {
                SendReceiveByte(CMD_WRITE_BUFFER2);     //      Else send write to buffer 2
                                                        //      command
        }

                SendReceiveByte(0x00);                  //      Don't care bits
                SendReceiveByte(address>> 8);           //      Don't care bits and 1st 2 bits of
                                                        //      address
                SendReceiveByte(address);               //      Buffer address

        for (i=0;i<length;i++)
        {
                SendReceiveByte(data[i]);
        }

        CS = 1;                                         //      DISABLE AT45DB321B Dataflash chip


        RECORD_LED = 0;                                 //      Turn OFF the Record led
}
```

```
//=============================================================
//      PUBLIC FUNCTION:       ReadfromBuffer
//=============================================================
//      FUNCTION CALL: ReadfromBuffer(buffer, address, *data, offset, length);
//      ARGUMENTS:     unsigned char buffer
//                     unsigned int address
//                     unsigned char *data
//                     unsigned int length
//      RETURNS:       Nothing
//-------------------------------------------------------------
//      DESCRIPTION:   Transfers the contents of the active buffer
//                     of the AT45DB321B Dataflash via the UART module
//                     to a PC running Hyperterminal
//
//      READ FROM ACTIVE BUFFER COMMAND FORMAT:
//                     CCCC CCCC XXXX XXXX XXXX XXAA AAAA AAAA XXXX XXXX

//                     8bit command, 14bits dont care, 10bit address, 8bits don't care
//-------------------------------------------------------------

void    ReadfromBuffer(unsigned char buffer, unsigned int address,
                       unsigned char *returned_data, unsigned int length)

{
        SEND_LED = 1;                           //      Turn ON the Send led

        unsigned int i;

        while(!READY_BUSY)                      //      Check to see if chip is busy
        {
                /* NULL STATEMENT */;
        }

//      READS DATA FROM BUFFER 1

        CS = 0;                                 //      ENABLE AT45DB321B Dataflash chip

        if      (buffer = BUFFER1)              //      If Buffer is 1
        {
                SendReceiveByte(CMD_READ_BUFFER1);      //      Then send read from
                                                        //      buffer 1 command
        }
        else
        {
                SendReceiveByte(CMD_READ_BUFFER2);      //      Else send read from
                                                        //      buffer 2 command
        }


                SendReceiveByte(0x00);          //      Don't care bits
                SendReceiveByte(address>> 8); //      Don't care bits and 1st 2 bits of address
                SendReceiveByte(address);       //      Buffer address
                SendReceiveByte(0x00);          //      Don't care bits

        for (i=0;i<length;i++)
        {
                *returned_data++ = SendReceiveByte(0x00);
        }

        CS = 1;                                 //      DISABLE AT45DB321B Dataflash chip

        SEND_LED = 0;                           //      Turn OFF the Send led

}
```

**UNIVERSITY OF CAPE TOWN**
**FACULTY OF ENGINEERING & THE BUILT ENVIRONMENT**

## LETTER OF INTENTION TO SUBMIT MASTERS DISSERTATION FOR GRADUATION JUNE / DECEMBER

Date:

**TO:**

Ms E Nobin – email address – Estelle.Nobin@uct.ac.za

Faculty of Engineering & the Built Environment
University of Cape Town
Private Bag
RONDEBOSCH, 7701

Fax No: (021) 650 3782

**Re: SUBMISSION OF MASTERS THESIS**

This is to inform you that I intend submitting my Masters thesis for graduation in

| | | | |
|---|---|---|---|
| **June / December :** | | **YEAR:** | |

| | | | | |
|---|---|---|---|---|
| **Title, Name & Surname:** | | | | |
| **Address:** | | | | |
| | | | | |
| | | | | |
| | | | | |
| **Telephone nos:** | *Home:* | | *Work:* | |
| | *Cell:* | | *Fax:* | |
| **Email address:** | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| **Student No:** | | | | | | |
| **Degree Code:** | | | | | | |
| **Thesis title:** | | | | | | |
| | | | | | | |
| **Department:** | | | | | | |
| **Supervisor/s:** | | | | | | |
| **180 credit dissertation or 120 credit dissertation** | | | | | | |
| **Currently Registered: Tick** | **YES** | | **NO** | | | |
| **Are you resubmitting your dissertation for examination? Tick** | | | **YES** | | **NO** | |

Yours sincerely
Estelle Nobin