The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

# A Handheld DSP Based Vibration Analyzer

Samuel Isaac Ginsberg

University of Cape Town

A dissertation submitted to the Department of Electrical Engineering, University of Cape Town, in fulfillment of the requirements for the degree of Master of Science in Engineering

Cape Town, August 2001

# Declaration

I declare that this dissertation is my own unaided work. It is being submitted for the degree of Master of Science in Engineering in the University of Cape Town. It has not been submitted before for any degree or examination in any other university.

Signature of Author

Cape Town
August 17, 2001

University of Cape Town

# Abstract

This dissertation investigates the design and implementation of a hand held vibration analyzer for use on electrical rotating machinery. The analyzer gathers data from an accelerometer and can present either acceleration, velocity or displacement information. Any information can be presented to the user in either the time or frequency domain. Numerical measurements can be made on the readings and readings can be stored onto a CompactFlash memory card. The instrument features its own type of file system and data storage metaformats. Facilities exist for the instrument to upload data to a computer or to download information from a computer. The instruments is made ergonomic by means of an extensive menu and hotkey system and by means of automated route tracking facilities.

University of Cape Town

# Acknowledgments

I am very grateful to my family who have encouraged me at every step during this project. My parents in particular have supported me in every way possible and without them this work would not have been possible.

I thank my supervisor, Professor Tapson for proposing the project and for supervising it.

I thank Dr Norman Morrison for his ideas and suggestions regarding the Fast Fourier Transform, and for the time that he spent discussing the instrument's mathematics with me.

The technical staff in the department made many valuable suggestions and gave useful help. These people include Mr Stephen Schrire, Mr Martin Lennon and Mr Paul Daniels.

I am grateful to my close friends for their patience and support. In particular Sven Shepstone was a constant source of encouragement and support.

University of Cape Town

# Table of Contents





University of Cape Town

University of Cape Town

University of Cape Town

University of Cape Town

University of Cape Town

University of Cape Town

# List Of Figures



University of Cape Town

University of Cape Town

# Chapter 1

# Introduction

## 1.1 Maintenance Strategies

Machines need maintenance. There are various different maintenance strategies, some of which offer distinct advantages over the others. These strategies can be divided into two basic groups.

### 1.1.1 Unplanned Maintenance

Unplanned maintenance is the most common strategy [31, p2]. Machinery is run until it fails and is not capable of running any more. Repairs are then carried out. This form of maintenance is suited to some forms of machinery but is sub-optimal in other cases. Where a system gives no prior indication of failure, and failure is rare and randomly distributed in time, this is the most applicable strategy [42, p6]. In the case of systems which fail at regular intervals, or in the case of systems which exhibit symptoms of failure before breakdown, this strategy results in unnecessary downtime, unscheduled loss of production and subsequent financial loss.

This form of maintenance generally requires only rudimentary diagnostic and monitoring equipment as it implies that no maintenance is carried out until chronic (usually obvious) machinery failure occurs.

University of Cape Town

### 1.1.2 Planned Maintenance

Planned maintenance is maintenance which occurs with forethought and according to a predetermined plan [42, p7]. The use of planned maintenance does not obviate the need for unplanned maintenance, with its disadvantages, but will help to reduce these problems.

Two major planned maintenance strategies are scheduled maintenance and condition based maintenance [42, p8]. Scheduled maintenance takes place on a regular basis, either in terms of time or in terms of machine operation, for example mileage covered in vehicle maintenance. Condition based maintenance occurs when a machine in some way indicates that maintenance would be advantageous.

Planned maintenance allows plant managers to schedule downtime in an opportunistic way. It also reduces the cost of maintenance in many cases [31, p3].

## 1.2 The Need and Requirements for Condition Monitoring Equipment

In order to implement a condition based maintenance scheme some means of monitoring machinery condition needs to be available. The monitoring system must give an objective and repeatable measure of the machine's condition, and these measurements must be taken on a regular basis.

The monitoring equipment should be capable of detecting the onset of failure modes, and it should give its warning indication as far in advance of catastrophic failure as possible. The monitoring system should detect as wide a range of potential problems as possible.

The condition monitor would be more useful if it could play an active role in the diagnosis of the problem. A system which only informs the user of a problem is useful, but not as useful as a system which provides the user with information which may lead to speedy diagnosis.

University of Cape Town

## 1.3 The Validity of Vibration Analysis as a Predictive System

There are very many machine parameters that can be measured, but the vibration signature of moving machinery, and in particular rotating machinery, contains more information than any other parameter [15, p1].

Vibration readings can indicate the presence of electrical and mechanical problems [15, p92] By indicating the onset of mechanical failure, such as bearing wear, vibration analysis allows the system or the user to predict that future failure is imminent.

Excessive vibration is also the cause of other problems. In extreme cases vibration has been known to physically tear machinery apart [15, pxii]. In these instances the vibration is a very good predictor of failure as it is itself the cause.

Vibration analysis is an established means of diagnosing many common machine problems. Some of these problems include bearing failure, machine imbalance, bent or damaged shafts and damaged rotor bars in induction machines [15, p88]. Many other problems are cited in the references [15,22,38].

## 1.4 The Need for a Portable Vibration Analyzer

Vibration monitoring is useful on a very wide range of machines. Not all of these machines would warrant the expense of their own dedicated vibration monitoring system. There are also plants in which no online system is available but where vibration measurements need to be taken. These applications call for a portable vibration measurement system. A portable system is also of use in conducting feasibility studies. In this case the results from the portable analyzer would be used to prove that a fully online system of vibration monitoring would be beneficial and financially viable.

If a machine does have online monitoring then a portable instrument could still be a valuable supplement in fault diagnosis. The online system would report a fault, but it may not furnish sufficient information for diagnosis to be made. The portable instrument would be used to gather supplemental data for analysis and diagnosis.

University of Cape Town

In an overall maintenance plan a portable unit would prove invaluable. Technicians could take regular readings on the equipment to which they are assigned. This has the benefit that the technician would be more aware of the status of his machinery and its general characteristics. This would boost the level of maintenance and enhance plant productivity.

## 1.5 Literature Review

The literature that was consulted for this thesis is divided into several basic categories. The first category of literature that was consulted consisted of books on maintenance, condition monitoring, machines and vibration analysis [15,22,31,38,42]. These books served to define the need for an analyzer and the requirements for such an instrument. The books provided insight into the topic of vibration analysis in an industrial context and this information was of primary importance in ensuring that the ultimate product of the project is of real use.

The next stage in the research was centered around the mathematics and major algorithms used by the instrument [11,12,21,26]. These books were mostly focussed on implementing the Fast Fourier Transform efficiently and within the limits of the instrument.

There was a certain body of literature that was consulted throughout the system development. The datasheets and books provided by the manufacturers of the devices used in the system are an essential resource without which no development could occur. The major works in this category are those from Analog Devices [4,5,6,7,8,9], Altera [1,2,3], The CompactFlash and IDE specifications [14,39], Dallas Semiconductor [17], and Maxim Semiconductor [23,24,25]. There are also some extremely valuable resources on the internet. These include information on the use of graphical LCD modules [19], and information on interfacing to IDE based devices [18,36].

The handbooks for existing instruments of a similar nature were examined to gain insight into industry expectations [13].

University of Cape Town

## 1.6 Dissertation Preview

This dissertation discusses the analyzer at several levels. The hardware and software are dealt with in separate sets of chapters. For both hardware and software there are three distinct levels of design. Each of the three levels has a chapter dedicated to it. The levels consist of describing the requirements for the subsystems, discussing the subsystems in detail and finally detailing way in which the subsystems are integrated to form a system. The entire plan of development is then as follows:

This first chapter gives an introduction to the subject of maintenance with a focus on condition monitoring achieved through vibration analysis. The need for a portable vibration analyzer is also discussed.

Chapter 2 gives a brief glimpse into the science of vibration analysis. This chapter is not intended as a text on the subject, rather the discussion presented there is used as a guide to defining the requirements for the instrument. The requirements for the instrument as a unit, as derived from the theory presented in the discussion, are also delineated.

Chapter 3 discusses the requirements for the major hardware subsystems. The systems are discussed individually with enough detail to define the design parameters to be used. This chapter draws on theory presented in chapter 2.

Chapter 4 details the specifics of each and every hardware subsystem. Each system is discussed, component choice is explained and the final circuitry is presented in a simplified form, with each subsystem shown in isolation.

Chapter 5 explains how the individual hardware subsystems are integrated into a coherent whole. The communication between subsystems is discussed where relevant and the design of interconnecting circuitry is mentioned.

Chapter 6 is not directly involved in the final product, but it presents one of the major topics dealt with during system development. The subject is that of processor development systems and programmable logic development tools. An understanding of these tools is essential to the design, or modification, of an instrument such as this.

Chapter 7 is the first chapter to deal specifically with system software. This chapter defines the requirements for each and every software component in the system. This

University of Cape Town

chapter draws on information presented in chapter 2, as well as information presented in the chapters on the hardware.

Chapter 8 gives the specifics of the software subsystems. Each subsystem is considered in detail and in isolation. The design decisions behind each unit are explained and the algorithms used in the software are presented.

Chapter 9 explains the linkage between the software components. The software components' interaction is explained. This in itself requires the explanation of certain software aspects.

Chapter 10 examines the completed system and conclusions are drawn. The strengths and weaknesses of the instrument are considered to give an accurate assessment of the project's success.

Chapter 11 proposes improvements to the system. These are aimed at improving the system and correcting any deficiencies that may exist.

University of Cape Town

# Chapter 2

# Basics of Vibration Analysis

## 2.1 Causes and Characteristics of Vibration

University of Cape Town

An understanding of the causes of vibration is necessary to understand the characteristics of those vibrations. The characteristics of these vibrations in turn determine the requirements of the instrumentation needed to measure and display the vibrations.

In a simple electrical machine there is a rotating part, the rotor, and a stationary part, the stator. The rotor turns in bearings which are held in the stator. The stator holds electrical windings. The rotor itself consists of a shaft and on this shaft is the assembly which carries the rotor's electrical system, often rotor bars or wire windings. The rotor then rotates another piece of machinery, often through some form of coupling such as a belt drive or gearbox. The other machinery may be a pump, compressor or some other mechanical device.

The electrical system may develop faults either in the rotor or the stator. Electrical problems usually exhibit themselves as multiples of the line frequency [15, p92]. Examples of electrical problems are broken rotor bars, damaged stator windings or loose laminations in a winding.

The mechanical system more commonly causes vibrations at frequencies related to the machine shaft's rotational speed [15, p90]. Typical mechanical problems are imbalance, damaged or bent shafts and, very often, worn or damaged bearings. These problems are

many and varied. Each mechanical component has its own set of pathologies, and some of these are listed in the references [15,22,38].

It is of particular importance to define the frequency range over which the instrument must measure, and the resolution that the analyzer must offer in the frequency domain. The lowest frequencies which may be encountered are probably those caused by oil whip in sleeve bearings [15, p92]. These frequencies are generally subharmonics of the shaft rotation frequency, often at a half or a third of the frequency. Cavitation may cause lower frequencies, but it often exhibits itself as broadband vibration noise. Various authorities disagree on the properties of cavitation vibrations [15,22,38].

On the upper end of the frequency range are those vibrations caused by bearings and gearboxes. Gearboxes may generate gear mesh frequency vibrations. The frequency of these vibrations is typically the product of shaft frequency and the number of teeth. This can result in vibrations at many kilohertz.

Bearings may vibrate at a frequency determined by multiplying the number of rolling elements and the shaft speed. This could easily result in vibrations at several kilohertz. An even higher frequency response is required to capture transient spikes in the time domain. These spikes are often caused by the impact as a roller element in a bearing rolls over a crack in one of the bearing races [31, p40]. In order to capture these impulses a bandwidth of tens of kilohertz (low frequency ultrasonic) is required.

The frequency resolution must be great enough to differentiate between vibrations at line frequency and those at shaft rotation frequency. Many induction motors run at very low slip and low speed and so this is a difficult requirement to meet.

Two other pieces of information that are of use in determining if a fault is present are the RMS level of vibration and the crest factor [31, p56]. These represent respectively the overall vibration level and the 'peakiness' of those vibrations.

In some cases vibration displacement is of use in diagnosis, in other case velocity is most useful and sometimes acceleration yields the most information [31,p52].

A machine may vibrate naturally even when no problem is present. Part of the value in vibration monitoring is in mapping a machine's vibration trends over time. In general noting the deterioration in a machine's vibrations yields better predictive data than simply

University of Cape Town

examining the machine in a single state.

## 2.2 General Requirements for Vibration Analysis

From the above discussion several points arise. The first criteria is that of low end frequency response. The line frequency is typically 50Hz, and the shaft frequency is lower than this, often by factors of four or more, in multi pole machines. The oil whip frequency is often 1/3 of this frequency, and so the instrument must measure frequencies down to 4Hz or lower.

The high end frequency response requirement is more difficult to define. The instrument's range should extend into the low ultrasonic range, but more specific information is not readily available. Similar instruments were examined [13] and it was found that the industrially accepted upper limit is 20kHz. This figure was taken as the design requirement.

The frequency resolution is also difficult to define precisely. There is an inherent compromise between frequency response and resolution in the frequency domain, as defined by the size of the transform used to convert information into the frequency domain. The greatest resolution is needed at frequencies around the line frequency. This is where the absolute frequency differences are the smallest. Once again the industrially accepted instruments were examined for some indication of industry expectations. This examination showed that 4096 frequency components or "lines" was as good as any other instrument available. This figure was chosen as the design aim. The frequency resolution is further enhanced by having variable sample rates. The use of low sample rates allows the user to trade frequency response for frequency resolution. This compromise is useful as the greatest frequency resolution is required at frequencies far lower than the analyzer's maximum frequency.

Most vibration information is best analyzed in the frequency domain [15, p88]. This implies that the instrument must be able to display information in the frequency domain. The time domain representation is also important for some problems [16 p207] and so the instrument must also produce time domain plots.

University of Cape Town

The instrument should be able to immediately compute or measure the RMS and crest factor of a vibration and convey these to the user.

The analyzer must be able to measure displacement, velocity or acceleration. This should be achieved using only one type of transducer. This transducer could be either a velocity sensor or an accelerometer, but not a displacement transducer as this leads to unacceptable measurement errors [15, p62].

In order to track machine vibration trends a system must exist for storage of the machine's characteristics. In order to accurately track vibration trends the instrument must either store many readings for each machine, or it must store the readings temporarily and offer a facility to upload this data to long term storage. The readings should be date stamped to make trend analysis easy and error free.

University of Cape Town

# Chapter 3

# Hardware Requirements

## 3.1 Data Capture Specifications

The nature of the data that is being captured and the transducers used determine the specifications of the data capture system. In this case the transducer is an accelerometer which must capture vibration data.

The transducer can only translate acceleration and the user will require either acceleration, velocity or displacement measurements, therefore a means of conversion is required. Displacement is the time integral of velocity, and velocity is the time integral of acceleration. This implies that a system of two integrators will suffice, and a means of bypassing either neither, one or both integrators will provide the required measurements. This integration could theoretically be performed either in hardware or in software. The decision was made to use a hardware solution. This decision was based on the need for triggering. If the integration is to be carried out in software, then it would be very difficult to make the instrument trigger (start data capture) when reading velocity or displacement.

As explained above the instrument must analyze frequency components up to 20kHz. This means that the analog to digital converter must have a sample rate of at least 40kSa/s. Much of the vibration data may be at frequencies considerably lower than 20kHz, and thus the sample rate of the analog to digital converter must be user selectable. In order to avoid aliasing of high frequency vibration components the anti alias filter must be automatically selected to match the sample rate.

University of Cape Town

Sample acquisition should not consume excessive processor overhead as this will affect the system response to user input and may affect the instrument's reliability and sample rate accuracy. For this reason it is preferable to have a separate converter clock which can determine the sample rate.

It is preferable to have high resolution data acquisition in order that the quantization error does not affect the accuracy of the Fast Fourier Transform. Too low a resolution would result in noise artifacts [37] which may deceive the user into an incorrect diagnosis.

High resolution data conversion also allows the instrument to perform autoranging in software, thus eliminating the complexity of external autoranging hardware, and the errors associated with it. The resolution considerations and the relative costs of available converters resulted in a 16 bit converter being used.

## 3.2 Display Specifications

The display must be suitable for a portable system. This implies that it must be lightweight and robust. Low power consumption is also of importance. The display must be capable of displaying graphics and text and must be of an adequate size to accurately portray the data being measured. For this application a monochrome display is sufficient.

It was decided that backlighting is not an essential feature. Most industrial environments have adequate lighting to read the display without extra lighting. The display type used should allow for a backlight to be added if needed. This could then be a feature of later models.

In order to keep the hardware design as simple as possible the display must have a built in controller. This reduces the amount of custom hardware to be built and reduces development time.

These factors resulted in a graphic LCD module being selected. The display size is 128 horizontal pixels by 128 vertical pixels. The module has an integrated controller on board. This is a standard locally available module.

University of Cape Town

## 3.3 Memory Requirements

The most fundamental requirement in terms of memory is that there must be enough program storage memory to store the sophisticated software required by the analyzer, and there must be enough data storage memory to hold the large arrays needed for the data that is captured for analysis, as well as adequate working memory, which includes a large screen buffer.

The chapter on the basics of vibration analysis outlines the criteria for the size of the Fast Fourier Transform. The resolution of the data to be transformed is determined by the required transform accuracy and measurement range. The transform size is 8192 points, giving 4096 spectral lines, and the resolution is 16 bits, thus the system needs 8192 * 16 bits of memory for the transform alone. The transform outputs 4096 data points, each with 16 bits of resolution. Owing to the length of time needed to compute the transform it would be advantageous to allow the user to view, pan, zoom and make measurements on captured data while the transform is taking place. This requires that the output from the transform process must be temporarily stored in a location separate to those used by the transform. This requires a further 4096 * 16 bits of memory. The total memory requirement for data storage is in excess of 12 kwords, excluding working space. The minimum feasible amount of data storage was estimated at 16 kwords.

The program memory requirement was less well-defined than the data memory requirement. For this reason the only fixed requirement that was initially specified was an expandable memory system.

Because the program size was not determinable during the initial design phase it was decided that the non-volatile program storage should be extendible. This implies that a system with an external ROM integrated circuit would be most suitable. The ROM system should be as simple and cheap as possible without compromising the expandability of the system.

University of Cape Town

## 3.4 Processing Speed Requirements

The system processor is responsible for all the system functions as well as the mathematical functions required to perform the Fast Fourier Transform. The FFT itself is the most processor intensive task that the instrument must perform. Owing to the nature of the system (a portable field instrument) the FFT must be performed fast to enhance the user's productivity. Experiments were conducted on various processor platforms in order to determine the approximate level of computing speed needed to make the system usable. The execution times were compared with other instruments of this nature in order to determine what the industry expectations are. The final choice of speed was in the 30 MIPS (Million Instructions Per Second) range.

## 3.5 Vibration Data Storage

The instrument is a field based unit. Very often users will wish to take readings in the field and store them for later analysis. Owing to the remote locations in which this unit will be used it is desirable that in the course of a normal working day the user should not need to return to a computer to upload data. The unit should also be upgradeable so that users who are in the field for extended periods may store larger volumes of data. The ease of taking measurements means that it should be possible to take at least one reading every five minutes. This means that an industrious user may take one hundred readings per day. The size of the reading is identical to the size of the data captured by the instrument for the FFT. This condition ensures that there will be no loss of data when the storage facility is used. The size of each reading is 8192*2 bytes. An additional 512 bytes of data are used to store information about the reading. Thus the unit must store a minimum of 1.7 Megabytes of data, as well as providing storage for the file system used to organize the data. The overall vibration data storage requirement was estimated at 2 Megabytes.

The vibration data storage must be non-volatile for it to be of use. There are a number of

University of Cape Town

memory technologies that could be used. These include battery backed SRAM, Flash and Magnetic disks. Flash memory was chosen as the storage type of choice because of its high data density, inherent non-volatility and comparatively low cost. Flash memory also offers low power consumption which is essential in a portable instrument.

Flash memory is available in many formats. These include individual integrated circuits, Flash drives and an array of different memory cards for use in the consumer market. The Flash cards offer the highest density, lowest cost and best availability. They are also readily removable from the instrument should the user wish to upgrade the capacity or archive data. For these reasons the card format was chosen. There are many different types of cards available and a decision was made to use the CompactFlash compatible cards. This choice was based on the popularity of CompactFlash in embedded industrial applications, as well as its compatibility with existing magnetic disk systems, which should ensure its continued availability. The CompactFlash is an open standard which means that data on interfacing to the cards is readily available. All the advantages of a multiple sourced device apply to the CompactFlash cards.

The smallest CompactFlash available holds 16 Megabytes of data. This is well in excess of requirements and is inexpensive enough to make it a realistic memory option.

University of Cape Town

## 3.6 Data Retrieval From the Instrument

The instrument is capable of capturing and storing large amounts of data. In order for this data to be fully utilized it is essential that a means of transfer exists between the instrument and a personal computer.

The fact that the CompactFlash card is removable suggests that data may be transferred by removing the card from the instrument and plugging it into a reader which is connected to the computer. This is disadvantageous as it requires an extra CompactFlash reader, and this expense may not be justified. In addition an external reader has driver software implications, which in turn lead to compatibility problems. For these reasons it was decided that an alternative means of transfer would be required.

The instrument uses a serial data link to move data to the computer. The transfer uses an RS232 compatible link. The link must carry large amounts of data, and so the link speed is 57600 baud. This is the fastest achievable transfer speed within the constraints of the hardware setup.

Using a 57600 baud link a single vibration reading will be transferred in (8192*16+256*16)/57600 = 2.3 seconds. This is acceptable in practice.

## 3.7 Physical Requirements

The instrument is to be used as a portable unit in an industrial environment. The unit must be easy to use so as not to hamper the user with cumbersome operations.

The instrument must be battery powered. Because of the large amounts of circuitry to be powered the unit must use high energy density rechargeable batteries. Nickel Metal Hydride (NiMH) batteries were chosen because of their high energy density and excellent charge/discharge characteristics.

The instrument as a whole must be rugged to withstand the rigors of use on an industrial plant. This implies a high quality casing, robust connectors, industrial grade keypad and secure mounting of all system components within the casing.

The casing must be big enough to accommodate all of the circuitry and the batteries. There must be a facility to mount the large display unit and a suitable surface to mount the keypad. The case must be shaped ergonomically so that users may use the instrument with ease and in comfort. The casing chosen was the DATEC control L case from OKW cases [30]. This case includes a desk mounted docking station which can include the serial link to a computer and battery charger circuitry.

The keypad used is a membrane type. These keypads are liquid resistant and mechanically rugged. They can be custom designed for the casing, thus contributing to the instrument's aesthetics.

University of Cape Town

# Chapter 4

# Hardware Subsystems

## 4.1 Analog Signal Processing

As explained above the instrument needs to have a set of two switchable integrators to produce the required displacement and velocity readings. The signals to control the integrators must come from the digital system that controls the instrument. The switching is achieved with two low cost analog switches [27]. The input circuitry can be AC coupled (capacitively coupled) as the input signals will have a zero mean. This vastly reduces the effect of offsets on the integrator stages. The basic input conditioning system is as follows:

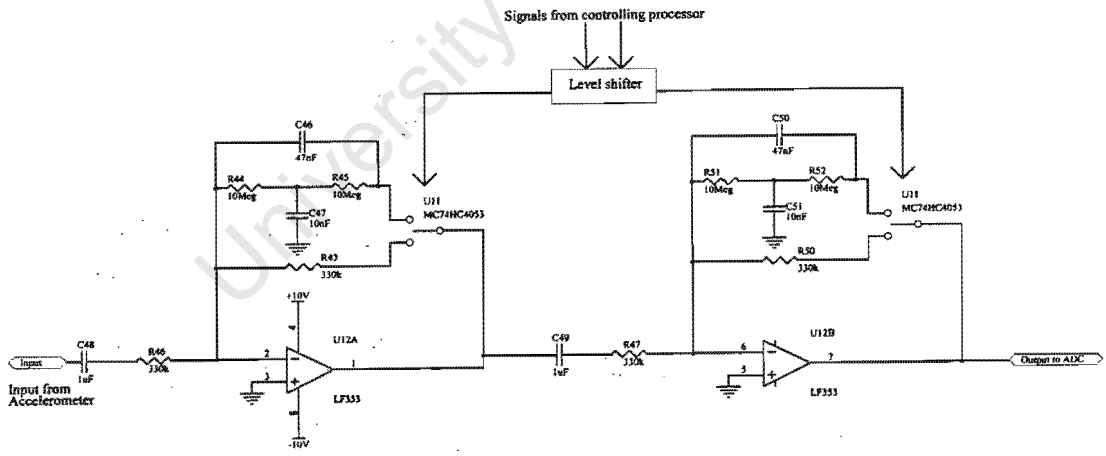



*Figure 1: Analog preprocessing circuitry showing the two switchable integrator stages*

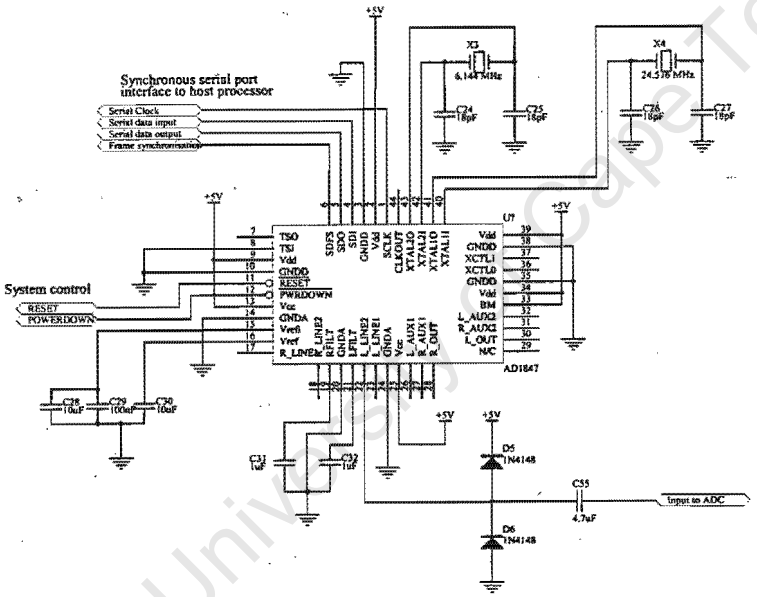
University of Cape Town

## 4.2 Analog to Digital Converter System

The Analog to Digital Converter (ADC) system must sample data periodically and send the results to the system processor. It must include a tunable anti aliasing filter and settable sample rate with a maximum throughput of at least 40kSa/s. The resolution must be 16 bits.

The device chosen was the Analog Devices AD1847 Audio CODEC. The CODEC includes the anti alias filter, it can generate periodic interrupts, and interfaces to the main system processor through a synchronous serial port. The device needs few external components.

The basic analog to digital converter system is as follows:




University of Cape Town

*Figure 2: Analog to Digital Converter System*

In order to obtain adequate noise performance, split analog and digital power supplies were used. This prevents digital noise breakthrough from contaminating the very sensitive inputs to the ADC.

The two diodes, D5 and D6 are clamp diodes to prevent damage to the inputs of the ADC should the input signal level go outside acceptable levels. This is a real possibility because of the higher supply voltage used by the preceding conditioning circuitry.
The two crystals, X3 and X4 determine the sample rate of the system and are critical to the accuracy of frequency measurements of the system. These particular frequencies were chosen to give suitable sample rates. From the CODEC datasheet [9] the available division factors are as follows:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3072 | 2560 | 1536 | 896 | 768 | 512 | 448 | 384 |

*Figure 3: Available division rates for the AD1847 CODEC*

The desired sample rates are:

| | | | | | |
|---|---|---|---|---|---|
| 2 kSa/s | 4 kSa/s | 8 kSa/s | 16 kSa/s | 32 kSa/s | 48 kSa/s |

*Figure 4: Desired sample rates for the instrument*

These constraints led to the selection of 6.144MHz and 24.576MHz as the crystal frequencies. These frequencies are standards, and the crystals are readily available. The following table gives the exact derivation of the sample rates from the crystal frequencies and ratios.

| ***Sample rate*** | ***Crystal selected*** | ***Division ratio*** |
|---|---|---|
| 2 kSa/s | 6.144 MHz | 3072 |
| 4 kSa/s | 6.144 MHz | 1536 |
| 8 kSa/s | 6.144 MHz | 768 |
| 16 kSa/s | 6.144 MHz | 384 |
| 32 kSa/s | 24.576 MHz | 768 |
| 48 kSa/s | 24.576 MHz | 512 |

*Figure 5: Division Ratios used to derive sample rates from the system crystals*

University of Cape Town

The connections and protocols used to interface the system processor to the CODEC will be discussed later in this document.

The various filter and bypass capacitors used here are the exact values specified by Analog Devices in their application note [9] and their EZ-KIT lite reference manual [6].

## 4.3 Digital Signal Processor System

When the processor system requirements were investigated, several possibilities were available in terms of the system processor architecture. The first option was to use a conventional microprocessor and to add external memory, peripherals and system support. Another option was to use a network of microcontrollers, with one controller handling the user interface, another controller for the mathematical computation etc. etc. The third option was to use a Digital Signal Processor (DSP) microchip. The DSP option was seen as the simplest to implement and the most cost effective. Digital Signal Processors are available in 16 and 32 bit bus widths, some have floating point capabilities and most have integrated memory systems, and they are available with a very wide range of performance.

It was decided that a 16 bit fixed point processor with a speed of approximately 30 MIPS would be adequate. Various options were assessed, covering all of the major manufacturers.

The Analog Devices' ADSP2181 was chosen for the project. The reason for the choice is that the processor has large amounts of integrated memory, more than other processors in its class, it has a very simple interface to a standard ROM for non volatile program storage and it has enough processor speed to satisfy the speed requirements. The processor busses are made available externally for attachment of peripherals and expansion of the memory system, should that be required. The on-board peripheral set is well suited to the system's needs. The development tools were also more affordable than some of the competing processor options.

The development board that was used during system development was the Analog Devices EZ-KIT lite board. This board incorporates the DSP itself, as well as the

University of Cape Town

CODEC and ROM. The kit also has a monitor program which allows the program to be run to be downloaded to the DSP via the serial port of a PC. This was used in the earlier stages of development, until the program became too large for the monitor to accept.
Included in the EZ-KIT is a set of diagrams showing how to integrate the ADSP2181 into a system. These diagrams were only slightly modified to form the processing core of the instrument.
The diagram for the processor system is given in Appendix D1.

## 4.4 Display System

The requirements for the system's display were outlined in the previous chapter. The exact display module that was chosen was the MGLS128128 made by Varitronix. This display has a built in controller, the T6963C made by Toshiba. This is a popular controller system for such displays. The display requires eight data lines, which are unidirectional in this application, and four unidirectional control lines. In addition there is a reset line which must receive a power up pulse to reset the module. The display also requires a -15 volt variable supply to set the contrast ratio. There is a control line to select the text font, either 6*8 or 8*8 pixels. In this instrument all the text will be the same size, 6*8, and so the font select line was tied to the positive supply.
The data for the display was not readily available. The display unit's manufacturer, Varitronix, only offers a mechanical diagram. They do not distribute any information on interfacing to the unit. The display controller is made by Toshiba. They do have a datasheet for their controller [41], but this was not readily understandable. Eventually details were found on a web site [19,20] and this information was used as the major source of information on the display.
In the first stage of experimentation the display was connected to the parallel port of a computer and driver software was written. In the next stage the display was connected to the processor bus via a port which was built out of discrete logic. This was soon found to be too cumbersome and it was replaced with a programmable logic device.
The connections between the programmable logic and the display are shown in the circuit

University of Cape Town

diagram in appendix D.1.

## 4.5 Keypad System

The keypad needs to have 16 keys in order to provide an effective user interface. The first experimental implementations of the keypad used a seventeen line interface in which each button was connected to one of the processor's data bus lines and the seventeenth line was common to all the switches. It was soon found that this was inconvenient as it required too much interconnection to the rest of the system.

The final version used a keypad arranged as a matrix. The eight lines needed to connect to the keypad are connected to the eight bidirectional port lines that are present on the processor.

Some buttons on the keypad, notably the 'quit' or 'menu' button have extremely high priority. These buttons must initiate system action immediately, and so they are connected to the processor's interrupt request line via a system of OR gates.

The keypad shares one of its lines with the real time clock module.

The keypad detail is given here:

University of Cape Town

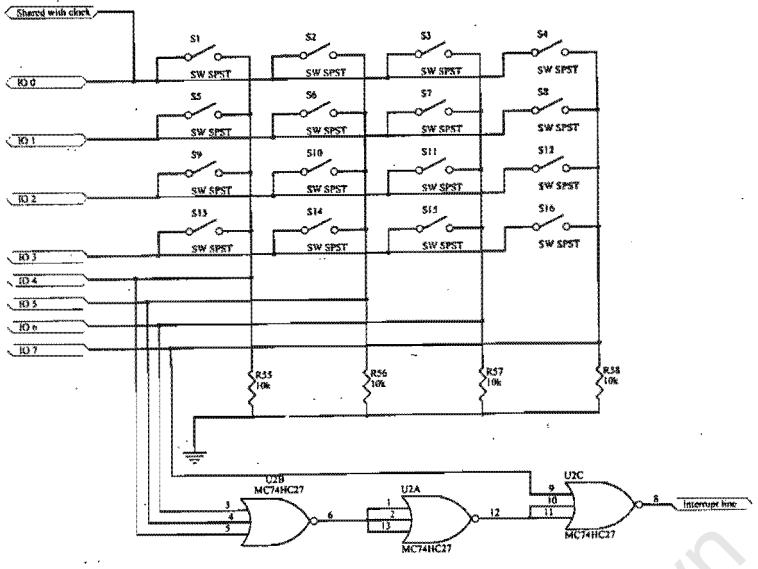



*Figure 6: Keypad arrangement*

## 4.6 CompactFlash System

The CompactFlash (CF) is used to store large amounts of vibration data.

The card itself has three alternative protocols. These are the PC Card Memory Mode, PC Card I/O Mode and the True IDE Mode. The True IDE Mode is an emulation of the ATA3 IDE standard used by magnetic disk drives.

True IDE mode was chosen as the protocol because ready made hardware is available to experiment on. This mode is also the best documented as it is covered in all the documentation on ATA3 and IDE interfacing.

The CF is arranged in a 16 bit wide format, i.e. all data accesses occur over a 16 bit bus. The data bus is therefore a bidirectional 16 bit bus.

The control of the CF is achieved over 8 lines. The use of these lines will be discussed later in this document. The hardware required for the control of the Flash is therefore 8 output lines.

University of Cape Town

In the first experimental stage of work the CF was interfaced to a computer parallel port and software was written in C++ to access the memory. This was partially successful and the next stage was undertaken. In the next stage of experimentation an 8255 programmable input output (PIO) device was used. This was interfaced to a microcontroller and software was written to interface to the card. These experiments were successful. The PIO was found to be too slow to interface to the high speed processor bus [29], and so the interface was achieved using high speed programmable logic. The programmable logic was interfaced to the processor's address and data busses. The CompactFlash interface is placed in the processor's peripheral address space.
The circuitry which was implemented in the programmable logic device is shown in appendix D.4. The circuitry to connect the CompactFlash to the programmable logic device is shown in appendix D.1.

## 4.7 Real Time Clock

The real time clock must satisfy some demanding criteria. It must keep time even when the instrument is turned off, and preferably even if the main batteries are discharged. It must provide accurate time and date information. The clock must have extremely low power consumption so that it does not drain its backup power source too fast.
The power requirements indicate that the system processor will not be suitable as a real time clock emulator. Thus external circuitry is needed. The integrated circuit that was chosen is the Dallas Semiconductor DS1302. This is an eight pin device with a serial interface. The device has built in power management to reduce the amount of external circuitry required. The clock runs off the main power supply when the instrument is turned on, and off its own battery or 1F Supercap when the system is turned off. There is a facility on the clock to charge a Supercap or rechargeable battery when the main power is available.
The timebase for the real time clock is a 32.768kHz crystal. The low frequency of operation aids in ensuring very low power consumption. The device consumes less than 300nA in standby mode [17].

University of Cape Town

The Clock also has 31 bytes of battery backed user RAM. This is useful in storing system settings, such as power up defaults.
The clock circuit is given here:

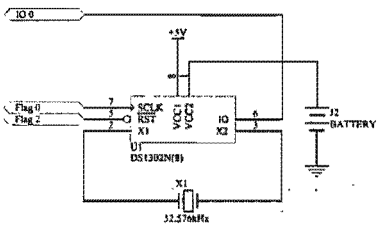



*Figure 7: Real Time Clock circuitry*

The clock has 3 interface lines to the system processor. One of these lines is a reset line, another line is a serial clock line and the third line is a data line. The reset line is used to initiate a data transfer to or from the clock. The reset line is unidirectional and it is driven from a flag pin on the processor. The serial clock line is used to synchronize the data transfer. This line is also unidirectional and it is also driven from a processor flag pin. The data line is bidirectional and it is connected to an input/output line on the processor. This line is shared with the keypad.

## 4.8 Serial Port Interface

The instrument needs an RS232 compatible serial port in order to communicate with a computer. The system processor does not have a UART on board, but it does have flag pins that may be used to emulate a UART. Level translation from the system's logic level to the RS232 line levels is needed. This level translation is achieved using a standard MAX232 RS232 compatible line driver integrated circuit.
The host computer needs some way of determining if the instrument is connected to the port and turned on. This is achieved with a loopback connection which is made via the MAX232 line driver. This implies that if the device is unplugged or switched off the

University of Cape Town

loopback will be broken. The RS232 lines used in the loopback are RTS (Request To Send) and CTS (Clear To Send).

The circuit diagram of the serial link is given below. The configuration of the MAX232 is derived from the datasheet [23].

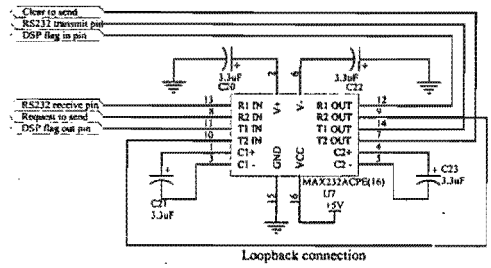



*Figure 8: Serial port interface circuitry*

## 4.9 Power Supply System

The instrument is portable and it must therefore be battery powered. The current consumption is 0.35A at a supply voltage of 5V.

The comparatively high current consumption and the cost of batteries makes the usage of rechargeable cells essential. The battery chemistry of choice is Nickel Metal Hydride (NiMH) because of its high capacity and excellent charge/discharge characteristics. These batteries do not suffer from a memory effect which is a further advantage. The nominal voltage of each cell is 1.2V.

The power supply to the electronic systems must be regulated to 5V ± 5%. The voltage regulator used is a low dropout type. This allows the instrument to use five NiMH cells instead of the six cells that would be needed otherwise. The battery life of the five cells with the low dropout regulator is almost identical to the battery life using six cells and a standard dropout regulator. The regulator must be capable of sourcing the high current needed by the circuitry.

University of Cape Town

negative voltage variable between 0V and -15V at a current of 1mA for its contrast adjust pin, and the analog processing circuitry requires ±10V at 3mA for the integrator circuitry. The analog to digital converter requires a filtered +5V supply at 70mA for its analog circuitry. Large amounts of decoupling are required because of the high speed logic which is powered from the 5V rail.

The low dropout regulator used is the LM2941 from National Semiconductor. The circuit used is very similar to the application diagram given in the datasheet from the manufacturers [28].

The LM2941 is housed in a five pin TO220 package and is capable of dissipating at least 1W without extra heatsinking, which is enough power so that it will not overheat. The regulator also features reverse battery protection, which is important in ensuring instrument ruggedness.

The -15V rail for the LCD module is generated using an inverting switching regulator from Maxim Semiconductor, the MAX637. The circuit and component values used are the same as those recommended in the Maxim datasheet [25].

The ±10V rails for the integrators used in the analog preprocessing were generated with a Maxim Semiconductor MAX680 charge pump integrated circuit. The values used are identical to the values recommended in the datasheet [24].

The complete power supply schematic is shown here:

University of Cape Town

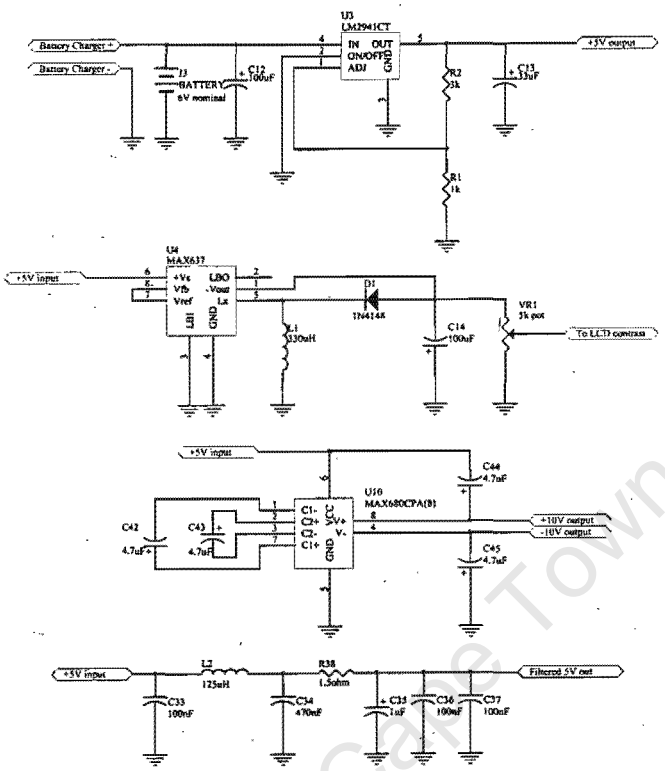



*Figure 9: Power supply circuitry*

No decoupling is shown in this diagram although extensive decoupling was used.

University of Cape Town

# Chapter 5

# The Complete Hardware System: How the Subsystems are Integrated

## 5.1 How the System Processor Connects to the CompactFlash and Display Module

The system processor has a dedicated address space for peripherals [7, chap 10]. This space is accessed by means of a 16 bit data bus and an 11 bit address bus. There is a strobe line to differentiate between peripheral access and memory access as well as read and write strobe lines. The bus speed is the same as the processor speed, 33MHz.

A programmable logic device (PLD) is used to create simple ports from the complex high speed bus. The device used is Altera's EPM7064SLC84. This device is from Altera's MAX range. It has 84 pins, 64 of which are usable as input or output pins in this system's configuration [2]. The pins are used as follows:

University of Cape Town

| *Function* | *Number of Pins Used* | *Signal Type* |
|---|---|---|
| Processor data bus | 16 | Input/Output |
| Processor address bus | 3 | Input |
| Processor control lines | 3 | Input |
| CompactFlash data lines | 16 | Input/Output |
| CompactFlash Control lines | 8 | Output |
| Display data lines | 8 | Output |
| Display control lines | 4 | Output |
| Integrator control lines | 2 | Output |
| Spare | 4 | |

*Figure 10: Programmable Logic Device pin usage*

The two lines for integrator control are used to control the integrators in the analog preprocessor stage.

The address decoding was configured to give the following address table:

| *Address* | *Address usage* | *Data alignment* |
|---|---|---|
| LCD Data | 0x001 | Bit 0 to bit 7 |
| LCD Control | 0x001 | Bit 8 to bit 11 |
| Integrator control | 0x001 | Bit 13 and bit 14 |
| IDE read/write strobe | 0x003 | Bit 5 and bit 6 |
| IDE read write control | 0x004 | Bit 12 |
| IDE data bus (write) | 0x005 | All 16 bits |
| IDE control bus | 0x006 | Bit 0 to 4 and bit 8 |
| IDE data bus (read) | 0x007 | All 16 bits |

*Figure 11: Address space usage*

The IDE lines are used to control the CompactFlash in true IDE mode.

In the early stages of experimentation the port circuitry was assembled from discrete logic elements and working circuit diagrams had been drawn up before the programming of the device commenced. For this reason it was decided that schematic entry would be

University of Cape Town

the best method of specifying the logic systems.
The development system used to configure the programmable logic is described later in this document.

### 5.1.1 Programmable Logic Internal Design

The ports are programmed into the PLD. The PLD is programmed from a schematic which is defined in terms of standard 74 series logic. The address decoding is carried out by a 74138 3 to 8 line decoder. The peripheral address strobe line is used to enable the decoder. The outputs from this decoder are combined with the read and write strobe lines by using NOR gates.
The unidirectional output ports consist of latches whose inputs are connected to the data bus. The output from the address decoder strobes the clock lines of the latches, thereby latching the data bus data onto their outputs.
The bidirectional port that connects to the IDE data bus is more complex. The output system comprises a set of latches whose inputs are connected to the processor's data lines. The outputs of these latches are buffered by tri-state latches. This enables the output section to be disabled when reading data in through the port. The input system consists of tri-state buffers whose inputs are fed from the IDE data bus. The outputs of these latches connect on to the processor data bus, thus providing the possibility of forcing data from the IDE data bus onto the processor's data bus.
The bidirectional port that is used for the CompactFlash operates as follows: If data is to be written out to the CompactFlash, first the data 0x1000 is written to address 0x004. This switches the port into output mode and data may be written out to the port by asserting that data on the data bus and accessing address 0x005. If data is to be read in from the CompactFlash, first the data 0x0000 is written to address 0x004. This places the CompactFlash data bus output drivers into a high impedance state. A subsequent read from address 0x007 will return the data that is present on the CompactFlash's data bus.
The full schematic programmed into the device is given in Appendix D.4

University of Cape Town

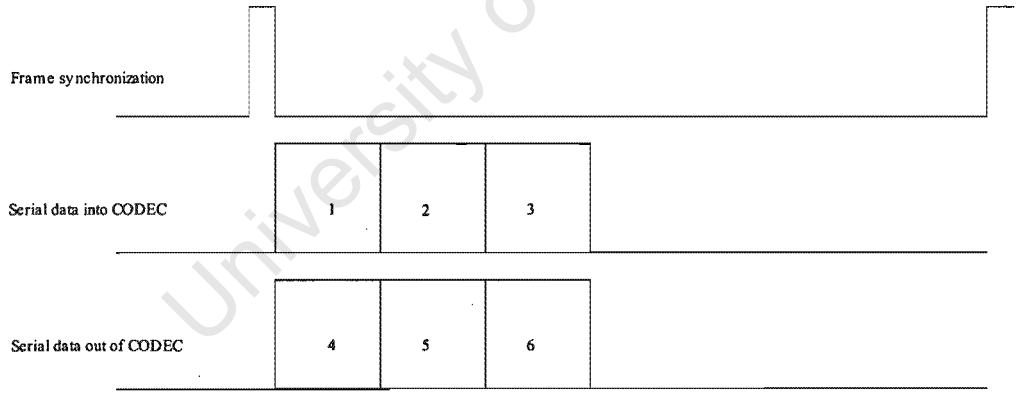
## 5.2 How the Analog to Digital Converter Interfaces to the System Processor

The communication between the ADC and the processor occurs over a synchronous serial port. The port is bidirectional. The speed of the link depends on the sample rate. For each sample that is taken by the ADC, three sixteen bit words are sent to the ADC and three sixteen bit words are sent to the ADC.

It must be borne in mind that the ADC is only part of a CODEC device designed for use in computer sound cards. This means that the CODEC is a stereo device and has both ADC and DAC within it.

The serial interface uses two data lines, one to send and the other to receive. There are two synchronization lines, one of which is the serial clock and the other is a frame synchronization line. The data is clocked into and out of the data lines on clock edges. The frame synchronization line is pulsed once every time a sample is taken. This line allows the processor to determine when a set of data is starting.

The transfer protocol is as follows:




1. Control word input
2. Left Playback data input
3. Right Playback data input
4. Status word output
5. Left capture output
6. Right capture output

*Figure 12: CODEC data format*

University of Cape Town

In this instrument data is only captured from one transducer, which is connected on the left input of the CODEC. No playback (output) is required from the CODEC. The control and status words are used, as well as the left capture output. All other time slots are received by the processor but ignored.

## 5.3 Overall System Diagram

The overall system is shown for clarity.

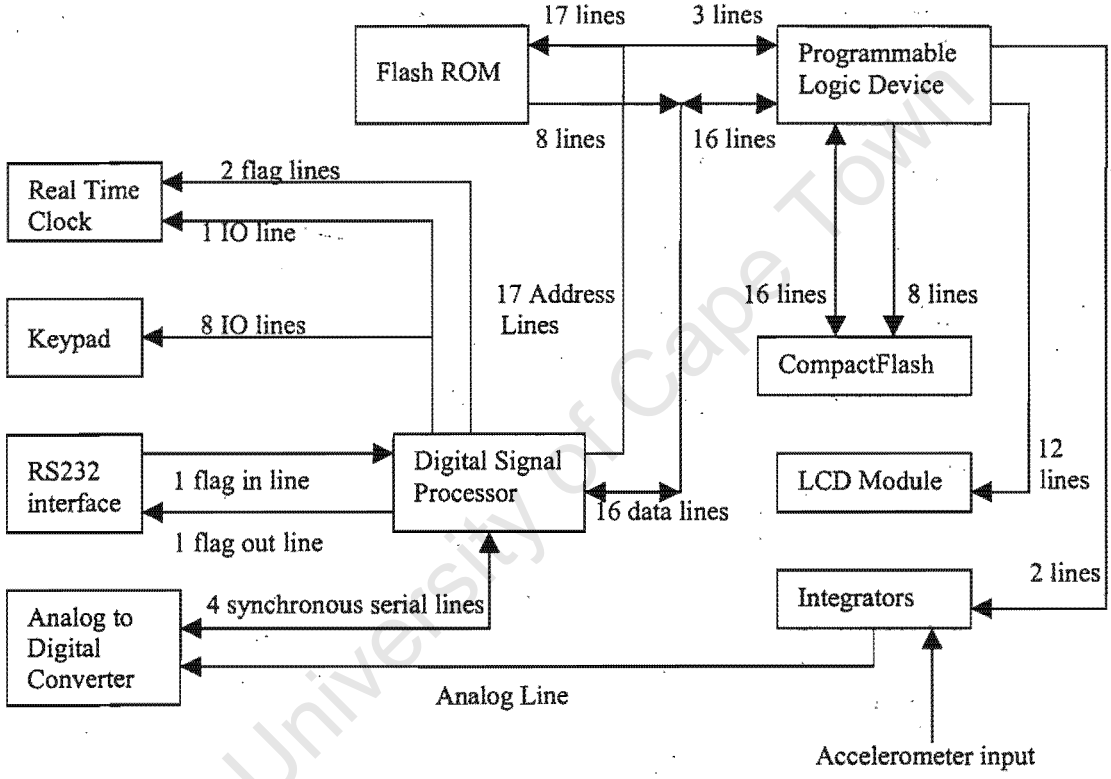



*Figure 13: Overall Hardware System Diagram*

University of Cape Town

## 5.4 Design of the printed circuit boards

Printed circuit boards are an essential part of any real electronic system and so their design, population and testing was undertaken as part of the project.

The instrument consists of three circuit boards, viz. the main board, analog board and CompactFlash board.

The main board contains the system processor, its program ROM, the programmable logic device, the real time clock, parts of the power supply, the keypad circuitry and the serial port interface. The analog board contains the input signal conditioning, part of the power supply circuitry and the CODEC system. The CompactFlash board contains the CompactFlash socket as well as status indicators.

Before the board layout commenced an enclosure was selected for the instrument. The mechanical diagrams for the enclosure were obtained and the board footprint was determined. The system schematic was captured in the Protel 98 CAD system. The schematic was converted to a netlist and imported into Protel 98 PCB layout CAD, which was used for all the PCB layout.

Some surface mount components were used on the PCB. The system processor is a 128 pin device and is only available in a surface mount package. This implied that the board would have to use surface mount components, and since the board was already using surface mount components it was decided that all of the resistors would also be surface mounted in order to save space.

The main board was designed first. The first issue that was considered on the main board was the interconnection of the program ROM, the programmable logic device and the system processor. These devices require a large bus to interconnect them. The bus runs at a speed of 33MHz, and so the signals must be carefully routed in order to maintain signal integrity. Great care was taken to ensure that the bus is as straight as possible and has as few vias as can be achieved. The large number of bus lines take considerable space on the circuit board. The width of the board is constrained by the housing, and this limited the track widths and copper to copper clearances used on the board.

At this stage in the design the programmable device showed one of its major advantages.

University of Cape Town

The design of the board was greatly simplified by assigning functions in the PLD to the most convenient pins. This allowed simple bus design. The only exception to this occurred on the interface to the CompactFlash. The large number of connections to the IDE data bus was not manageable on the most convenient pins because of device restrictions. The solution was to route one of the CompactFlash connections to a pin on the opposite side of the device. This was more attractive than using a more sophisticated logic device, and so there is one trace that has an inelegant and unusual routing.

The programmable logic device is in circuit programmable via a 4 wire JTAG interface. Provision was made for in circuit programming of the PLD.

Once the major bus had been laid out on the main board, the rest of the components were grouped into functional blocks and placed on the board in these groups. This led to simple and efficient design with good use of space, resulting in acceptably high component densities.

The analog board layout was complicated by the need for a split ground system and separate 5V supplies for the analog and digital circuitry. A ground plane was used underneath the CODEC device. The ground was split into an analog and a digital ground. The grounds are connected at only one point, as recommended by the Analog Devices in the CODEC's datasheet [9].

The CompactFlash board is essentially a passive adapter. The purpose behind the board is to provide a robust means of mounting the CompactFlash receptacle, and a means of connecting to it via a standard IDE bus. One of the major advantages of this system is that the board could be used as a standalone adapter to read CompactFlash devices from a standard computer. Should the CompactFlash devices become obsolete this module could be replaced with a standard IDE flash drive.

In capturing the schematics various components had to be captured into schematic libraries. These components include the system processor, ADSP2181KS, and the CODEC, AD1847JP.

When the boards were designed not all of the required footprints were available. Footprints that had to be captured include the 84 pin through hole PLCC socket for the programmable logic device, the 44 pin through hole PLCC socket for the CODEC and

University of Cape Town

the CompactFlash connector.

The boards were designed using a design rule of 0.01inch minimum track width and 0.01inch minimum copper to copper clearance. The boards are double sided and through hole plated. There are two solder masks, top and bottom, and a top silkscreen overlay. The boards were manufactured locally and populated by hand.

The full PCB layouts are presented in Appendix E. and photographs of the completed boards are shown in Appendix F.

University of Cape Town

# Chapter 6

# Software Requirements

## 6.1 Hardware Support

Interfacing to the system hardware is a very major portion of the system's software. There are many different hardware modules that need to be interfaced to. In order to make the system modular, with all the inherent advantages of modularity, the hardware is all accessed through a system of drivers.

The hardware drivers are all programmed in assembler in order to obtain maximum control over the hardware. The fact that the drivers are programmed in assembly and the main program is written in C means that there are issues relating to assembler/C interfacing that were also addressed.

## 6.2 Display of Vibration Waveforms

The time domain representation of vibration waveforms is important in diagnosing some types of machine faults, as discussed above. The instrument therefore needs to present the information to the user in a meaningful manner.

The data that is captured from the accelerometer is displayed on an oscilloscope type display, with time on the horizontal axis and magnitude on the vertical axis. The instrument provides a zoom function to allow the user to see details clearly. The zoom factor is settable from 1 to 32. When the zoom factor is greater than unity, not all of the

University of Cape Town

information can be visible at one time and so a pan function is also provided.

When the waveform is captured 4096 samples are taken per capture. The display unit only has 128 pixels in the horizontal direction, and as a result some system of compression is needed. The compression must give the user as accurate a representation of all of the compressed data as is possible within the limitations of the display.

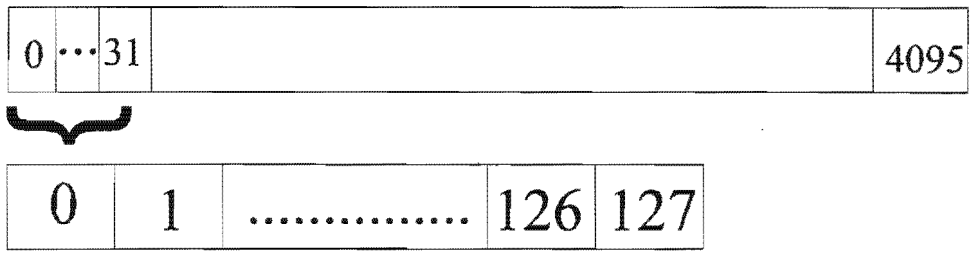



*Figure 14: The effect of the compression algorithm.*

In some cases there may be a strong periodic component in the vibration waveform [31, p58]. A system of triggering can be useful in accentuating such phenomena. The trigger level is adjustable to make certain unusual events trigger the instrument.

The time domain display features a cursor that may be moved over the waveform. This cursor can be used to measure the magnitude of vibrations at any point on the waveform. In addition the cursor gives a time readout (measured in seconds). The time that is read out is the time at which the sample was taken relative to the trigger event. This feature is useful for determining the time between events.

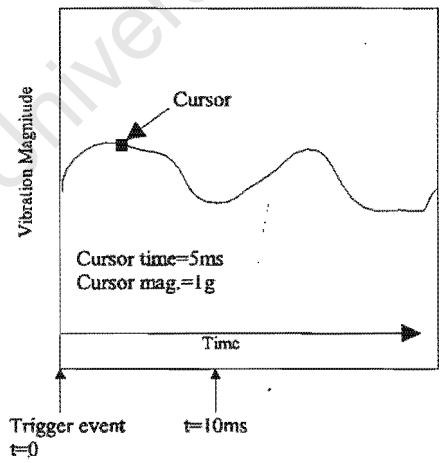



*Figure 15: The time domain with cursor and cursor readout.*

University of Cape Town

The RMS amplitude of the vibration is given. The RMS amplitude is calculated over all of the samples in the capture.

The crest factor of the waveform is given. The crest factor is defined as the ratio of the peak value in the capture divided by the RMS amplitude of the waveform. This metric is used in determining if any large amplitude events are occurring on an otherwise smoothly running machine.

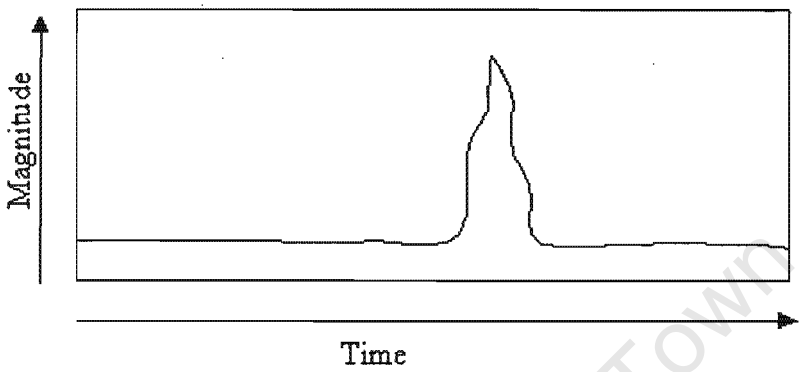



*Figure 16: Time domain waveform with a very large crest factor. Note that the peak does not contribute significantly to the average vibration amplitude, but it is of importance.*

The limited resolution of the display could result in inaccurate information being conveyed. This problem is partly overcome with an autoranging system. The autoranging system is designed to choose a range that fits the biggest peak on the screen. This means that if the machine only exhibits small amplitude vibrations they too will be accurately portrayed.

## 6.3 Display of Vibration Spectra

Most of the information in vibration readings is visible in the frequency domain. The basic requirements for the transformation into the frequency domain are outlined in previous chapters.

The spectrum display consists of a horizontal frequency axis and a vertical axis which shows the magnitude of the spectral components.

In order to get optimal display of the information that is available, zoom and pan functions are provided.

As for the time domain display, a system of compression is required to portray the data accurately on the screen. The compression technique is different to the time domain

University of Cape Town

compression as different features are of importance.

The display also shows RMS amplitude and Crest factor. These have the same relevance as they do in a time domain representation.

The instrument has autoranging capabilities for the spectrum as well. This is essential in showing the user as much data as possible.

There is a cursor in the frequency display. The cursor can be moved along the frequency display. The cursor position is displayed at the bottom of the display. As the cursor moves along a frequency axis, the cursor readout is given in Hertz. The ability to determine the exact frequency of a spectral peak is crucial to diagnosing a machinery fault. The cursor readout also gives magnitude, so that the magnitude of a spectral peak may be determined.

When the data is compressed on the screen several spectral components are represented by one pixel on the display. If the resolution of the screen is allowed to determine the frequency resolution of the cursor readout, the overall resolution will be suboptimal unless the instrument is zoomed in fully for every frequency component. This would vastly degrade the usability of the instrument and so a special algorithm is used. The algorithm examines all of the frequency components represented by the cursor position and determines which of those components has the maximum amplitude. The frequency of that component is calculated and displayed.

There is also a peak finder function. This function finds all of the harmonics of the current cursor position and highlights them with small circles if they are on the display. This function may be turned off if not required.

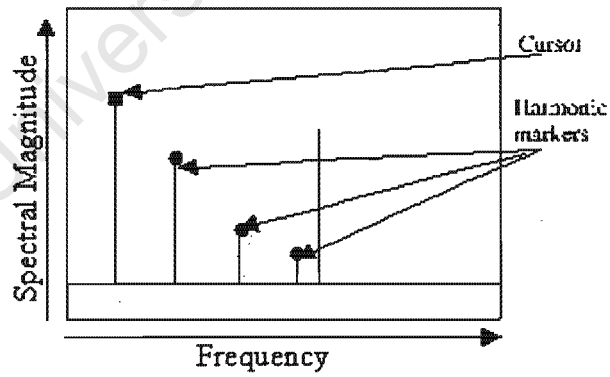



*Figure 17: Illustration showing the effect of the peak finder. Note that there is no harmonic marker on the highest frequency component as this is not a harmonic.*

University of Cape Town

## 6.4 Vibration Data Capture

In order for vibration analysis to be effective, a record must be kept of a machine's vibration characteristics. This implies that the readings that are taken with the instrument must be recorded and kept for extended periods.

The instrument thus needs a system for data storage. The storage system needs a large enough capacity to store very many readings, as described in the hardware requirements chapter.

Because the instrument can store many readings, a system of organization is needed in order to save and retrieve waveforms in a meaningful manner. This implies the need for a file system. The file system must allow each record to be named and stamped with the time and date. The user must be able to get a listing of records and select a record to capture data into or read data out of.

The file system must satisfy these needs without consuming excessive system resources. This means that the system used must be both simple and powerful. Owing to the consistent nature of the data that is to be recorded the system does not need extreme flexibility and so this can be sacrificed for compactness and efficiency.

The instrument must have all the facilities for storage handling. This includes formatting the storage medium, creating records and deleting records.

## 6.5 Route Walking Facility

A full preventative maintenance plan must include regular checks of a set of machines in a plant. If the process of taking these readings can be automated to some extent then a less qualified operator can take the recordings, thus increasing the number of potential operators. This is beneficial to the maintenance plan as readings can be taken more frequently.

Automation is achieved by arranging a group of readings into a route. The route is then '"walked", i.e. an operator walks through the predefined route and makes recordings as

University of Cape Town

per instructions that are displayed by the instrument.

The route is set up on a computer and sent to the instrument through the serial link. The instrument then has the ability to read the route information and prompt the user to move to the next machine. The user presses a key to confirm that (s)he is at the specified machine. The instrument displays the vibration waveform and when the operator is satisfied with the data that is captured (s)he presses the save button. The instrument saves the data and prompts the operator to move to the next machine.

If the user does not wish to capture new data for a machine there is a facility to skip machines on a route. This is required for cases in which a machine may not be running or is inaccessible.

In the course of walking a route the operator may wish to review a record taken during the route, or they may wish to take supplemental readings if a problem is detected. The instrument must allow the user to suspend the route, make other measurements and then to seamlessly resume the route from the point at which it was suspended.

## 6.6 Upload Facilities

The instrument must have the ability to upload data to a computer. The user must be able to select a data record to upload and send it to the computer.

When data is arranged in a group, e.g. in a route the user must be able to send the entire route to the computer for ease of use.

## 6.7 Setting Up Instrument Parameters

A complex instrument needs to have its operational parameters set up by the user to optimize performance.

The sampling rate needs to be set up. The sample rate is a trade off between frequency range and resolution in the frequency domain. The user must be able to set the sample rate in order that the spectral features can be identified with optimal frequency resolution.

University of Cape Town

The window type is settable. Rectangular, Triangular, Hamming and Hanning windows are industry standards [13] and as such they are given as user settable options on the instrument.

The vertical range of the instrument is settable. The default setting is for the instrument to automatically use the optimal range, but in some circumstances the user may wish to force the use of a higher range. If the user selected range would result in a screen overflow the screen output is suppressed and the user is notified.

The instrument can measure acceleration, velocity or displacement of a vibration. The user can select whichever option is most suited to the application.

All readings that are taken are stamped with the time and date. This is essential for proper record keeping and machine performance tracking. The instrument has a real time clock/calendar on board. The user must be able to set this up easily.

## 6.8 Hotkey System

There are very many options available on the instrument. In order to make the system intuitive keys should be labeled. There are too many functions to assign one function per key. Some keys perform as many as four different functions in different contexts. The most elegant solution is to have a set of keys under the display. These 'display keys' change function as required in every different context. The function which the key will perform is displayed on the bottom of the screen, just above the button itself.

The hotkey legends are all textual, but the available font on the display is not suitable for the purpose, as it is too big. The key legends are all stored in a bitmap format and displayed as graphic images for this reason.

University of Cape Town

# Chapter 7

# Software subsystems

## 7.1 Display Support

This module provides all of the low level display interfacing, as well as primitives for text and graphic display and screen management.

The dispay functions are summarized in the following table:

| *Function* | *Effect* |
|---|---|
| cdlo, celo, rdlo, wrlo | Switch control lines to low state |
| cdhi, cehi, rdhi, wrhi | Switch control lines to high state |
| pput | Place data onto data lines |
| cput | Send a command to the display |
| dput | Send a data byte to the display |
| lcd_setup | Ensures correct power up status |
| lcd_init | Initializes the display's parameters |
| setpixel | Turns a pixel on or off |
| lcd_put_char | Places a character on the screen |
| lcd_xy | Move the text cursor to position (x,y) |
| lcd_clear_text | Clear all text from the display |
| lcd_clear_graph | Clear all graphics from the display |
| lcd_print | Put a text character on the display |
| lcd_putarray | Print a graph on the display |

*Figure 18: Summary of display functions*

University of Cape Town

The module is written in C with inline assembler code. The code is all contained in the file ana_lcd.c. The drivers for the display were based largely on material sourced on the internet [19,20,41].

The lowest level (closest to the hardware) consists of functions that manipulate the control and port lines that connect to the display.

There are four lines that connect to LCD control lines. There are two functions per line, one to raise it and the other to lower it. These functions are called *cdhi, cdlo, cehi, celo, rdhi, rdlo, wrhi* and *wrlo.*

There is a function called outp that sets up the data for the LCD onto the data lines of the display. This function uses a function called *pput* to place the data onto the lines. It should be noted that the data and control lines are accessed using the same address and so care must be taken that manipulation of data lines does not corrupt control signals and vice versa.

The next level of abstraction is in the *cput* and *dput* functions. The cput function puts a command byte out to the display while dput puts a data byte into the display. These functions make use of the lower level functions. The put functions automatically manipulate the control and data lines to provide the correct level of abstraction.

The *lcd_setup* function is used to ensure that when the system is powered up the control lines are deasserted for reliable operation.

The *lcd_init* function is responsible for setting up the display for operation. This function sets up the display's memory by assigning memory areas for graphics and other areas for text. The memory usage (lines per row of text and graphics) is also set up. The final stage in the initialization is to define the behavior of overlapping text and graphics (an OR function is used) and to turn both text and graphics display on.

The fundamental graphics function out of which all other graphics functions are built is the *setpixel* function. This function takes in horizontal and vertical coordinates and a variable called optype. The coordinates determine which pixel on the screen is to be affected. The optype variable determines if the pixel will be turned on or off. The function translates the coordinates into a display memory address and sends the address as well as the command to turn a pixel on or off to the LCD.

University of Cape Town

All text is displayed as individual characters. *The lcd_put_char* function puts a character on the screen. The character is determined by a byte that is sent to the function. The character is placed at the current position of the text cursor. After the character has been sent to the display, the cursor position is advanced. The display does have a feature to automatically advance the text cursor, but this was found to give unpredictable results when used simultaneously with graphic display.

Often there is a need to place text at arbitrary positions on the display. The *lcd_xy* function can set the text cursor to point to any text position on the screen. This is accomplished by calculating the LCD memory address from a set of coordinates and setting the LCD cursor to point to that memory address.

The display needs to be cleared when changing screen displays. The function *lcd_clear_text* clears all text off the screen, while leaving all graphics unaffected. This function works by writing null data (shown on screen as a blank) to the entire text memory.

The graphics display can be cleared independent of text by invoking the *lcd_clear_graph* function. This is achieved by writing null data to the entire graphics memory.

Text strings are displayed with the *lcd_print* function. This function takes in a null terminated string and displays it on the screen, beginning at the position of the text cursor.

Graph display is a major requirement of the instrument and so a special routine was developed to do this. The function is called *lcd_putarray*. The data for the graph is placed in a 128 element global array called *grapharray* and *lcd_putarray* is invoked. The data from the array is plotted, one element per pixel. The input array consists of signed integers. When an element is zero, that element is placed exactly halfway down the screen. The incoming elements are scaled down by a factor of 512 to allow the function to take input directly from the ADC capture or from the transform used to produce the frequency spectrum.

If the display only shows 128 pixels in response to an input array the result is very difficult to read. In order to improve readability, an interpolation system is needed. This interpolation joins the pixels with straight lines. The interpolation works by calculating

University of Cape Town

the height difference between consecutive pixels. This gives the number of pixels that need to be added. The direction of the interpolation (up or down) is also calculated. Half of the interpolation points are added in on the column of the first datum, and the rest are plotted in the column of the second datum. This gives the truest possible interpolation within the restrictions of the display.

It must be noted that the interpolation does not create or extrapolate to obtain extra data. The interpolation is purely to enhance the readability of the graph. The instrument will always show at least 128 measured data points on the screen.

## 7.2 Keypad Support

The keypad must be read in and the information from it must be presented to the main C program in order to influence program flow. In addition the software must allow the keypad to interrupt the main program so that critical keys are handled immediately.

The keypad is arranged in a matrix with four rows and four columns. The software must scan the rows, examine the columns and decode which key, if any, was pressed.

When the function is entered the keypad interrupt is disabled to prevent conflicts. The rows are then pulled high in turn. For each row that is pulled high all of the columns are scanned. If any column is high then a key has been pressed, and that key is uniquely defined by which row was raised and which column was detected.

When the function has determined which key was pressed the keypad interrupt is re-enabled. The result of the function is placed in the system processor's AR register. The AR register is used by the C compiler to contain the return data for functions. This means that the buttons function is called as a normal C function would be and it returns data as any C function would.

University of Cape Town

## 7.3 Analog to Digital Converter Support

The variables used by the analog to digital converter are summarized for easy reference:

| *Variable name* | *Function of variable* |
|---|---|
| xreal | 4096 element data array |
| ximag | 4096 element data array |
| full | Flag indicating if the data arrays are full |
| trigger | Flag indicating if the capture system is triggered |
| triglev | The level required to triggrer the capture process |

*Figure 19: Summary of variables used in the data capture process*

The Analog to Digital Converter (ADC) is a sophisticated system. The connection between the ADC and the processor is through a synchronous serial port. The hardware level interfacing is built in to the system processor and thus needs no software. The system has an autobuffering system, whereby the data coming into the serial port is buffered directly into memory. The buffering system interrupts the processor only when the designated area in memory is full. The memory area is designated by the registers i2, l2, i3 and l3. These registers are therefore not available for use by the rest of the software. The compiler must be directed not to use these registers or else a conflict will result.

To initialize the ADC several stages are required. The first stage is to set up the serial port to which the device is connected. This is achieved by setting up the port to run at full speed, to match the setting on the ADC, and then setting up the port to use autobuffering. The autobuffering registers must be loaded with the appropriate memory addresses and the serial port must be set up to use the correct registers.

The next stage is to configure the ADC itself. This is done by sending configuration words to the converter. The first word that is sent contains data that is placed into the miscellaneous information register. This configures the ADC's serial interface. The input setup registers are transmitted. These specify which inputs the ADC is to use and what gain should be used on each of those inputs. The Data Format Register is then set up.

University of Cape Town

This register specifies the sample rate among other things. The Interface Configuration Register is set up to allow the ADC to autocalibrate. Autocalibration occurs after the ADC is set up.

The ADC samples and sends data automatically. This data is buffered into memory and an interrupt is called when the buffer is full. The interrupt handler is responsible for triggering, i.e. deciding when an event has occurred, and packing the data into the large arrays needed for the transformation to the frequency domain. In addition the interrupt handler must interact with the main program so that while the main program is busy with data that data is not overwritten.

The main program sets the global variable called *full* to zero when it is waiting for the ADC to capture data. The ADC capture function sets *full* to a non-zero value when the data arrays are full. No ADC data capture occurs when *full* is non-zero.

The interrupt handler is a function called *sp0r_ctrl*. When the function starts it checks the status of the variable *full*. If the main program does not need more data, the interrupt handler exits immediately in order to free up the processor for other computations. If the main program is waiting for data, the interrupt handler checks to see if the system is in a triggered state. If the system is not triggered the incoming data is examined to determine if a trigger event has occurred. If the system is not triggered and no trigger event has occurred the interrupt handler exits. If the system was triggered, or if the trigger event occurred in the latest sample then the interrupt handler starts packing the data into the arrays for use by the rest of the software. If the latest sample is the last sample needed to fill the data array, then the variable *full* is made non zero to indicate to the main program that the data array is full.

The trigger system is comparatively rudimentary. The system triggers when there is a positive slope on the incoming data, and the data exceeds the trigger level set by the user. The interrupt handler therefore stores all samples for one sample period after they are captured, in order to determine if the slope of the incoming data is positive. The user can set up the trigger level in the waveform display screen. The user setting is communicated to the interrupt handler by means of the global variable called *triglev*. The interrupt handler keeps record of if it is triggered by means of the *trigger* variable, which is

University of Cape Town

cleared to zero if the system is not triggered.

The data must be packed into the data array in a specific way to facilitate efficient transformation. Further details of this are given in a later section. The data arrays are called *xreal* and *ximag*. Each of these arrays is a 4096 element array of integers. The data is arranged with the first sample in the first element of *xreal*, the second sample in the first element of *ximag*, the third sample in the second element of *xreal* etc. In short the even numbered samples are packed into the *xreal* array in the order that they are captured and the odd numbered samples are packed into *ximag* in the order that they are captured. In order to pack the data correctly, the interrupt handler keeps record of the current position in the xreal and ximag arrays, as well as a variable which tracks if the sample is even numbered or odd numbered. The variables which track the array positions are used as pointers and they point to the memory address of the next array element to be packed. This requires separate pointers for the *xreal* and *ximag* arrays. The addressing mode used to pack the data into the array automatically increments the pointers to the next array position.

When the *xreal* and *ximag* arrays are full the *full* variable is made non zero. The pointers used to access the *xreal* and *ximag* are reset to point to the start of the arrays. The *trigger* variable is made zero to indicate that a trigger event must occur before packing is restarted.

## 7.4 Real Time Clock Support

The low level interface to the real time clock is written in assembly code and is contained in the file ana_clk.dsp. The higher level code is written in C and is contained in the main C module, ana.c.

The functions are summarized here for reference:

University of Cape Town

| *Function* | *Effect* |
|---|---|
| put_clk | Puts data into the real time clock |
| get_clk | Gets data from the real time clock |
| clk_init | Initializes the real time clock |
| clk_string | Converts clock data into packed string format |
| showclk | Shows the current time and/or date on the system display |
| set_clk | Sets the current time and/or date |

*Figure 20: The functions that form the real time clock system*

The variables used by the real time clock system are as follows:

| *Variable* | *Function of variable* |
|---|---|
| clk_com | Holds a command to send to the real time clock |
| clk_dat | Holds data to be written to/ read from the real time clock |
| detail | Determines if the time or the date are to be read |
| clockstring | Output string containing packed clock data |

*Figure 21: The variables used by the real time clock system*

The low level interface to the clock consists of two functions, one to write data to the clock and one to read data from it. The function to write data is called *put_clk* and the function to read data is called *get_clk*. Both of these functions were written in such a way as to be compatible with the C calling and parameter passing conventions.

When the clock is written to there are two parts to the process. First the address and control byte is sent to the clock in a serial fashion, and then the data to be written to that address is sent, also serially.

When clock data is to be read, the control and address byte is sent to the clock serially and then the data from that address is read back serially.

The time is set and obtained by writing to or reading from certain addresses. This means that the real time clock appears to the processor as a memory device in which some addresses contain the current time and date.

The keypad shares a data line with the real time clock, and so before the clock is accessed the keypad interrupt is disabled in order to prevent spurious interrupts.

University of Cape Town

When writing to the real time clock the command (which comprises control and address information) is passed into the function by means of the global variable *clk_com* and the data is passed into the function through the global variable *clk_dat*. To write to the clock the command is loaded into the shifter unit in the processor. The command is shifted out of the shifter into the clock. When the command has been sent the data is loaded into the shifter and sent in the same manner.

When the clock is to be read, the command is passed to the function by means of the global variable *clk_com* and the function passes data back to the main program through the variable *clk_dat*. To read from the clock, first the command is loaded into the shifter unit and shifted into the clock. The data line is switched to input mode and the data is shifted into the processor's shifter unit.

The higher level clock functions are written in C. The function to initialize the clock when the system is powered up is called *clk_init*. This function checks that the clock is running, checks that the clock is in 24 hour mode and that the trickle charger is switched off to avoid damage to the alkaline backup batteries.

The function *clk_string* reads the time and/or date and converts it to a string for use by other clock access functions. There is an input parameter for this function, called *detail*. If *detail* is equal to zero only the time is packed into the string, if *detail* is equal to one then only the date is packed and if *detail* is equal to two both the time and date are packed into the string. The function's output is always into a string called *clockstring*.

When the current time and/or date are to be shown on the screen, the function *showclk* is invoked. This function calls the *clockstring* function and prints the resulting string on the screen at the position of the text cursor. *Showclk* takes in an input called *detail* which has the same effect as the input parameter to the function *clk_string*.

The clock is set by means of the function *setclk*. This function displays the time or date and allows the user to set the time and date. The function continuously displays the time or date and updates the real time clock as soon as a user presses an appropriate key. Great care is taken to prevent the user from entering an invalid time or date. This complicates the function considerably.

When setting the time, each digit is set independently. This imposes some rules on what

University of Cape Town

data may be set. The rules are:

- Tens of hours<3 provided that hours<4. If hours>3 tens of hours<2
- If tens of hours>2 hours<3
- tens of minutes<6
- tens of seconds<6

The user selects which time field is to be set by using the hotkeys which are marked with left and right arrows. As soon as a field is accepted and shown on screen it is set into the real time clock.

The real time clock packs the data into its memory with typically two fields per byte. This means that the data entered by the user must be packed into the bytes.

The date is also set digit by digit. The rules for date setting are as follows:

- tens of months<2 provided that months<3. If months>2 then tens of months<1
- If tens of months==1 months<3
- tens of days<4 provided that days<2. If days >1 then tens of days<3

The system does not have a means of checking that the date is appropriate for the month. This means that the user may set the clock to display the 31st of February. The real time clock does feature graceful recovery from impossible settings. Thus if a user sets an impossible date the clock will recover to a possible but inaccurate state.

Selection of fields is made with the hotkeys and field entry is via the numbered keys on the keypad.

## 7.5 Serial Port Support

The serial port is used to communicate with a personal computer. Ordinarily a Universal Asynchronous Receiver Transmitter (UART) would be used, but this would complicate the system hardware unnecessarily and so a UART is emulated in software.

The UART emulation software was taken from an Analog Devices application note [8]. The software was modified to suit the application.

The UART software is all written in assembler code and is contained in the file called ana_uart.dsp.

University of Cape Town

The functions used in the emulation are as follows:

| *Function* | *Effect* |
|---|---|
| init_uart | Sets up the UART |
| out_char_ar | Sends a character out of the UART |
| turn_rx_on | Enables the UART receiver |
| turn_rx_off | Disables the UART receiver |
| get_char_ar | Reads a character in through the UART |

*Figure 22: Functions used in the UART emulation system*

The function called *init_uart* initializes the UART system by setting up the input and output flag lines used by the serial interface. The function also sets up the system timer. The function of the timer will be explained below. The control variables used for the UART emulation are also set to default values.

To transmit, the system timer is started up. The timer interrupts at three times the bit rate. This is to simplify the system for reasons that will be explained below. The data to be transmitted is padded with start and stop bits and loaded into the shifter unit. No parity bits are used. The system then waits for a timer interrupt. When the timer interrupts, the first bit (start bit) is shifted out of the shifter onto the UART transmit pin. Thereafter every third timer interrupt causes the next bit to be shifted out to the transmit pin, until the shifter data is exhausted. During the whole process the UART status flags are set to indicate that transmission is in progress.

The transmission function is invoked by calling the function *out_char_ar* with the data as an input parameter.

Before a character can be received into the serial port the function *turn_rx_on* must be invoked. This function sets a status flag that informs the UART that reception can occur. Bytes are captured by invoking the *get_char_ar* function.

The receiver also uses the timer as a basis for baud rate generation. The UART emulation is capable of full duplex operation and for this reason the timer must run at identical rates for the UART receive operation and the transmit operation.

University of Cape Town

In order to provide the widest baud rate error tolerance all incoming bits should be sampled in the middle of the bit period. This allows each bit to be displaced in time by up to half a bit time in either direction (too fast or too slow) before a reception error occurs.

All received data bytes begin with a start bit. This is a low pulse that has exactly the same duration as each of the data bits. The UART thus waits for a negative edge before commencing reception. Upon receiving this negative edge, the UART waits for 1.5 bit periods before sampling the first data bit. This ensures that the sample is taken at the optimal moment for the first bit. Subsequent samples are taken at exactly one bit period intervals until the required number of bits has been received.

The period that the UART waits after the detection of the start bit is 4/(3*baud rate). In order to generate that period the timer interrupts occur at three times the baud rate for convenience. This is the reason that both the receiver and transmitter use timer interrupts that occur at three times the baud rate.

The relative timing for the UART is shown in the following diagram.

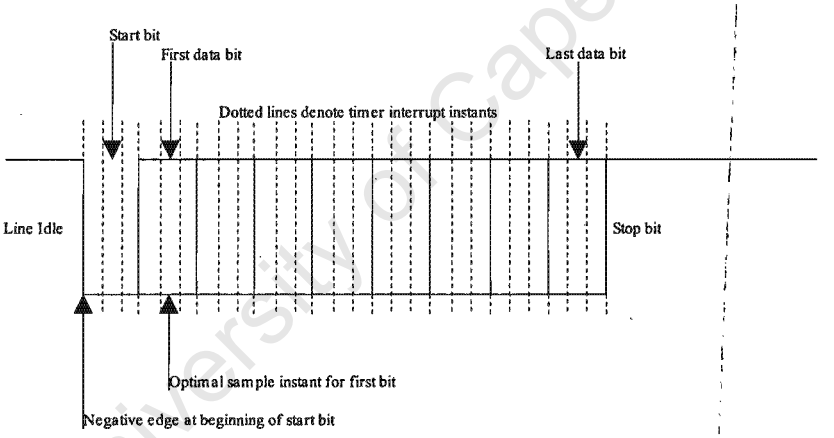



*Figure 23: Timing for the UART receiver system*

The incoming bits are shifted into the shifter unit. The shifter is used to convert the serial bit stream into a complete byte.

When reception is complete the receiver can be turned off by invoking the *turn_rx_off* function.

University of Cape Town

## 7.6 Fast Fourier Transformation

The Fast Fourier Transform (FFT) is responsible for converting time domain information into the frequency domain. The system requires that the transform produces 4096 individual spectral lines. This means that the FFT needs 8192 time domain data points to work on. The data comes into the system as 16 bit signed integers. It was found by experimentation that the FFT needs more than 8 bits of precision for its computation, or else excessive errors occur. These errors are exacerbated by the sequential nature of the FFT algorithm. The data is used in computations and the results are used in further calculations and those results are input to another set of working. This process is repeated many times. This means that any computational inaccuracies accumulate to produce incorrect output. The amount of memory available limited the precision of the calculation results to 16 bits. This is adequate precision to produce accurate outputs. The smallest floating point type available occupies 32 bits per number. For this reason the FFT was adapted to work with integer storage for its data vectors. Floating point numbers are only used for intermediate calculations, such as multiplication with trigonometric functions. The reduction in the use of floating point numbers increases the execution speed of the algorithm, resulting in a fast, accurate system that is feasible within the limited memory available to the system.

The basic algorithm was taken from E Oran Brigham's book [12]. The major modification that was made was the use of integer storage types for the data vectors. The use of integer types results in the possibility that the data will overflow. Overflow of any variable at an intermediate stage in the calculation results in total distortion of the output, rendering the entire transform useless. If the data is scaled down before the calculation commences smaller features of the waveform will be lost in the rounding errors inherently associated with the integer storage used for the data vectors. The method used in the algorithm avoids these problems by progressively scaling the data vectors down. At each stage in the algorithm the data element being written is scaled down by a factor of two. The nature of the FFT calculations is such that at each stage the new data to be stored is no larger than twice the data that it is derived from. Thus dividing by two ensures that the

University of Cape Town

data is always as large as it may be with no possibility of overflow. In addition to this the division by two is efficiently implemented as a right shift by the compiler. This helps to produce an algorithm that runs fast.

Because the FFT must perform in a minimum of memory an in-place algorithm is used. The in-place algorithm progressively overwrites its own results with the new results. This means that many available FFT systems could not be used. The C compiler from Analog Devices has a built in FFT function. This was considered, but because it is not an in-place algorithm [4] it is unsuitable in this application.

The data captured by the ADC is real data. The data results from a real process and as a consequence it has no imaginary part. This allows an optimization to be made. The 8192 data points are packed into the input vectors in a special order and a 4096 point FFT is run. The results of the FFT are then unpacked by a separate algorithm to yield 4096 spectral components. The overall execution time of the 'real FFT' [26] is almost identical to the time used by a 8192 point FFT because of the inefficiency of the unpacking algorithm compared to the efficiency of the FFT algorithm, but the memory requirement for the transform is halved.

Each spectral component consists of a real and an imaginary part. The magnitude spectrum is calculated and it is this that is displayed for the user.

The overall FFT takes approximately six seconds to run. This is considered acceptable for the size of the task, but the instrument response time to a command should be considerably shorter to enhance usability. The FFT is thus modified to allow the instrument to respond to user commands. This modification in no way changes the mathematical processes used in the transformation, they are merely a repeated interlude in the algorithm. The user can thus quit from the spectrum display, toggle the peak finder on and off, zoom in on the display and pan across the display while the FFT is running. If the user does not require these functions the routine to check the keypad executes fast, so that the overall impact on the execution time of the FFT is minimal.

All of the FFT routines are written in C and are contained in the file ana_math.c. The function *fft* is the actual FFT and the function called *unpack* contains the unpacking system. Also within the file ana_math.c are the supporting mathematical functions used

University of Cape Town

by the FFT.

## 7.7 Waveform and Spectrum Display Systems

There are some algorithms used in the display routines that need further discussion.

### 7.7.1 Zooming and Panning

The zoom factor is settable from 1 to 32. When the zoom factor is set to 1 there is exactly one sample shown per pixel on the graph. When the zoom factor is 32 there are 32 measured data points represented by each pixel on the display. The pan function allows the screen display to be moved over the data, showing the data section by section.
The variable that determines the zoom factor is called *number_per_line*. The pan position is controlled by the variable *start_display*. When the user zooms in on the graph the zoom variable is halved and when the user zooms out the zoom variable is doubled. When the user pans left the pan variable is decreased and if the user pans right the pan variable is increased. The formula used to determine the size of the change in the pan variable is:
start_display±=32*number_per_line
The effect of this is to shift in one quarter of a screen of information per pan operation.
When the graph is displayed the information that must be displayed is compressed into an array of 128 8 bit elements, one per horizontal pixel on the screen. The data is packed into the 128 element array starting at position *start_display*. *Number_per_line* elements are compressed into each position of the 128 element array.

### 7.7.2 Screen Data Compression

Compression is needed to produce the 128 element screen array. Different compression

University of Cape Town

systems are used for the frequency domain and the time domain displays.

In the frequency domain it is very important that the user can see any spectral peaks, as these could indicate a problem. If the spectral peaks are in any way reduced in size the user may not see important features. For this reason the biggest element in the set of compressed data is the output.

For waveform displays there are conflicting requirements. The first requirement is that the waveform of the vibrations is represented accurately. The second requirement is that any particularly large peaks are shown. Large peaks of this nature can indicate the presence of problems such as cracked bearing races [15, p84]. In order to preserve the wave shape an average system is used. All the data to be compressed into a pixel is averaged and that data is shown on the pixel. Any large peaks in the display are attenuated slightly, depending on their duration, but the variable trigger level can be used to overcome this. When the zoom factor is low the attenuation of large spikes is reduced.

### 7.7.3 Peak Finder Function

The peak finder is a feature of the spectrum display that is used to find harmonics. When the cursor is moved along the display all the harmonics of the current frequency are examined, and if there is a spectral peak at any of these frequencies a marker is placed on the graph.

The algorithm is as follows:

Calculate all harmonics of the current frequency from $2^{nd}$ harmonic up to the largest harmonic that can be represented. Examine each of those harmonics to determine if a peak is present at that frequency. Place markers at all harmonic frequencies.

## 7.8 CompactFlash Support

The CompactFlash (CF) system is one of the largest and most complex systems in the analyzer. The support system for the CF comprises low level drivers which operate on the

University of Cape Town

port lines connected to the card as well as high level functions to manage data storage functions and data organization. This section will deal with the interface to the CF and the next section will concentrate on the organizational system used to achieve consistent data storage and retrieval.

The CompactFlash is used in an IDE emulation mode. This means that the CF is accessed in the same manner as a standard ATA 3 compliant hard disk drive. Four major sources of information were used to derive the IDE interface system. There are articles on the internet [18,33,34,36] that outline a microcontroller to IDE interface. These articles give details of working systems that were implemented by the articles' authors. There is a book available that gives general information about the IDE interface [35]. The ATA 3 specification was obtained and consulted [39]. The CompactFlash specifications were obtained [14] and these also yielded useful information.

Before discussing further details of the drivers certain basic concepts must be introduced. The CF has a 16 bit wide parallel data bus. The data is arranged in sectors. Each sector is composed of 256 16 bit works. The sector is the smallest addressable data unit.

The IDE interface standard evolved from other magnetic disk drive standards [35], and is itself designed primarily for magnetic drives. For this reason there are some peculiarities in the interface. The addressing system refers to heads, cylinders and sectors. This convention is applied to the CF device even though they do not use heads or multiple disk platters.

The IDE interface is intended to address to devices that are connected on the same cable. The devices are known as the master and the slave, although it is important to note that the master device in no way controls the slave. The IDE interface thus makes provision for accessing these devices independently. The interface implemented here is for a master only system.

University of Cape Town

### 7.8.1 Development of CompactFlash Drivers

The complexity of the interface was realized soon after research into the subject commenced. For this reason the development was carried out on several platforms for

ease of debugging. The ultimate result was that the CompactFlash was interfaced to the analyzer, but the steps taken in order to achieve this were many and time consuming.

The first stage in the development was to connect a hard disk drive to a computer through two parallel ports. The object of this exercise was to test the basic operation of the IDE interface. The advantage of this approach was that almost all of the interface hardware was ready made. This eliminated hardware issues and meant that development could proceed from there. This system was never fully developed. The basic system was developed until it was clear that the basic mode of operation was correct. The interface was not reliable before work on the next system commenced.

The next platform that was interfaced to the CompactFlash was the Motorola MC68HC908GP32 microcontroller. This system was chosen because it involved moderate hardware development and it offers an extremely good debugging environment. By this stage CompactFlash devices and adapter boards were available and so these were used in conjunction with the microcontroller. This interface was developed to the extent that the microcontroller could store and retrieve information from the CF. The interface was also supplemented with diagnostic tools, such as the ability to read error information as well as extended error codes from the CF. The basic timing parameters of the CompactFlash were verified and it was decided that the overall system should be suitable for implementation on the final system.

The final stage was to drive the CompactFlash from the DSP. This was chosen as the last step in the process because the development environment for the system processor does not offer the advanced features needed to develop the interface efficiently.

University of Cape Town

### 7.8.2 The IDE Interface in PIO Mode

The IDE interface supports multiple data exchange metaformats. These all serve different needs and are used to achieve the desired goals within different environments. The most appropriate system in this case is the programmed input output (PIO) mode.

The control lines for the interface were discussed above, but they will be mentioned here because they are essential to understanding the interface.

The CF appears to the host as a set of registers. All data and commands are passed through these registers. This reduces the number of control/address lines to a feasible number and helps ensure compatibility with the ever expanding device capacities.
The internal registers are addressed with a set of three address lines (A2,A1 and A0) and two chip select lines (!CS1 and !CS0). In addition there is a read strobe line (!RD) and a write strobe line (!WR). These two lines are used to specify whether a register is being written to or read from. The !reset line is only asserted at power up. It must be noted that the chip select lines as well as the read and write strobe lines are active low.

### 7.8.2.1 The IDE Registers

The registers that are used by the interface are as follows:

| *Register name* | *A2* | *A1* | *A0* | *!CS1* | *!CS0* | *!RD* | *!WR* |
|---|---|---|---|---|---|---|---|
| Data IO Register | 0 | 0 | 0 | 1 | 0 | X | X |
| Error Information Register | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| Sector Counter Register | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| Start Sector Register | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| Low Byte of Cylinder Number | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| High Byte of Cylinder Number | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| Head and Device select | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| Command Register | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| Status Register | 1 | 1 | 1 | 1 | 0 | 0 | 1 |

*Figure 24: The IDE registers used in the CompactFlash interface*

The address of the register is specified completely by the combination of the three address lines and the two chip select lines as shown above.
It must be noted that other registers do exist in the IDE interface system, but only these basic registers are needed for this PIO interface system. Some of the cylinders are shown as being written to only, while they are in fact read or write accessible. This table only

University of Cape Town

shows the registers in the context in which they are used in the analyzer. Any further information should be obtained from the information referenced above.

The data IO register is the register through which data is written into the CF and read out of the device. For this reason the register is bidirectional.

The command and status registers share the same address. When that address is written to the command register is accessed and when it is read from the status register is returned.

The error information register contains information about any error that may have occurred. This register is read only. The error information register is only valid when the status register indicates that an error occurred. The meanings of the individual bits in the error register are different depending on the physical device being accessed. The meanings of bits are not the same for CompactFlash modules and Hard Disk Drives. The interface described here is not sophisticated enough to use the error register. The error register was used in diagnostic work during system development but the final system merely checks whether there was an error without considering the exact details of the error.

All of the registers are byte wide except the data register which is 16 bits wide in order to accommodate the 16 bit wide data.

The sector counter register is used to specify to the CF how many sectors are to be transferred in the next data exchange. When the sector counter register holds zero, then one sector will be transferred in the next exchange. When the sector counter register is initialized to a number x then x+1 sectors are to be transferred in the next exchange. A maximum of 256 sectors can be exchanged in a single data exchange. In this system only one sector is transferred at a time.

The start sector register is one of several registers that are used to specify where in the CompactFlash the data is to be written to or read from. The reason for the terminology 'start sector register' is that in multiple sector transfers this register is used (among others) to specify where the read or write must start from. Successive reads or writes would be made to/from consecutively numbered sectors.

The low byte of the cylinder number is also used to address the sector that is to be read from or written to.

University of Cape Town

The high byte of the cylinder number is appended to the left of the low byte of the cylinder number to form a 16 bit number which specifies the cylinder number. Earlier versions of the IDE specification only used two bits in this field. The analyzer does not support the older standard as the CompactFlash was introduced after those versions were made obsolete and thus there should be no need to support the older standard.

The head and device select register is also used to address the sector to be accessed. There is a bit which specifies which device (master or slave) is to be accessed. There is also a bit which specifies which addressing mode will be used to access the device. In addition there are two bits which had to be set under the older standard and have subsequently been made obsolete. This interface uses those bits in their old context as they were only discontinued after the introduction of CompactFlash devices. The interface only accesses the master device and does not use the four head addressing bits as they would always be zero for devices smaller than 8GB, which is considerably bigger than the analyzer would possibly need.

The format of the head and device select register is as follows:

| ***MSB*** | | | | | | | ***LSB*** |
|---|---|---|---|---|---|---|---|
| Set to 1 | Address mode selection bit | Set to 1 | Make this bit 0 to select master device | Head number bit 3 | Head number bit 2 | Head number bit 1 | Head number bit 0 |

*Figure 25: The head and device select register*

The command register is the register into which all instructions are written. The instructions control the CF and are the only means of specifying CF operation.

The status register is used by the host to determine the status of the CF device. This register is used to determine if the device is ready, if the device is busy executing a command, waiting for data or has data ready for the host to read. The status register is used by the host to determine if an error has occurred. The meanings of the bits that are used in this interface are shown here:

University of Cape Town

| *MSB* | | | | | | | *LSB* |
|---|---|---|---|---|---|---|---|
| Busy bit. Set when the device is busy | Ready bit. Set when the device is ready | | | DRQ bit Data request bit | | | Error bit. Set when an error has occurred |

*Figure 26: The IDE status register*

### 7.8.2.2 The IDE Commands

There are very many commands in the IDE command set. The CompactFlash command set is similar, but not quite identical. In this interface system only three commands are used. These three commands suffice to read data, write data and automatically configure the host for different CompactFlash devices.

The write data command is represented by the code 30 hexadecimal. When this code is issued the device accesses the required data.

The read data command is represented by the command 20 hexadecimal. When this command is issued the device prepares to receive data to be written.

The Identify drive command instructs the device to prepare a set of data which describes the device parameters. The command is represented by the number EC hexadecimal. These parameters can be used to determine the size of the device. The size of the device is the only data needed by the analyzer to set up the device for correct operation. The identify device data contains many data fields. Only three of these are needed to determine the size. These three parameters are the number of logical cylinders, the number of logical heads and the number of logical sectors per logical track.

### 7.8.2.3 The IDE Bus Cycle for Reading From a Register and Writing to a Register

The timing diagram and an explanation for register writes is given here. All the exact timing parameters are given in the device specifications. The specification should be



University of Cape Town

consulted if fast device access is required. In the interface that was implemented the data volumes were low enough that a slow interface was used and the timing parameters are of little importance. It must be noted that during debugging an extremely slow interface was attempted in order to allow easy examination of line levels. This interface failed. No official specification offered any reason for this, nor were any of the CompactFlash manufacturers' technical support teams able to offer any explanation.

When reading from an IDE register the procedure consists of several steps. Firstly the data bus is set into read mode. The register address is then asserted on lines A2, A1, A0, !CS1 and !CS0. The !RD line is asserted (pulled low). The register contents become available on the data bus. When the host has read this data it deasserts the !RD line.

The procedure for writing to an IDE register is extremely similar to the process of reading from a register. The data bus is set into write mode. The register address is asserted on the control lines. The data to write to the register is asserted on the data bus and the write strobe, !WR, is asserted and then deasserted. The data may be removed from the data lines.

It is recommended that the data bus be kept in a read state by the host as this will avoid the possibility of the IDE device attempting to assert data on a bus which is configured as output from the host.

University of Cape Town

#### 7.8.2.4 Identifying the Device, Reading Sectors and Writing Sectors

IDE devices have a data buffer in them that can store at least one sector. When data is written to a sector or read from the sector it always goes through this buffer. The individual words within the buffer are not addressable, rather the buffer is read in a sequential manner. Consecutive reads from the buffer access consecutive words within the buffer.

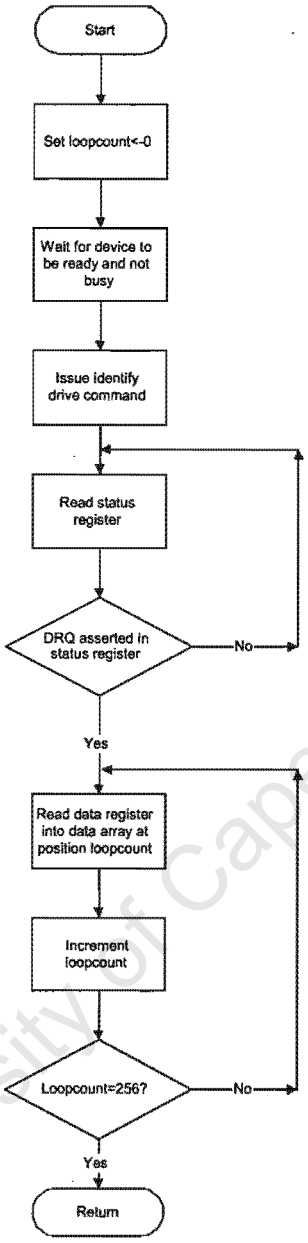
Before any device operation is attempted the host must read the IDE status register. The busy and ready bits must be examined to determine if the device is ready to perform the required operation. Attempting an operation on a device that is not ready or is busy will produce unpredictable results and has been known to damage data that is stored on the

device.

The CompactFlash features a self identification feature. This feature is accessed by writing the identify device command ($EC_{16}$) to the command register. The device fills the buffer with identification data. The status register must be read to determine if the device is ready to be read from. When the device is ready to be read from the DRQ (data request) bit in the status register will be asserted. When the device has asserted the DRQ bit the host must read the buffer. This is not optional. When the buffer is full it must be read or spurious operation may result. The buffer is read by executing 256 read commands on the data register, thereby retrieving the entire buffer full of data. The entire buffer must always be read, even if the data is not all required.

The operation may be represented diagrammatically as follows:

University of Cape Town




University of Cape Town

*Figure 27: Operation of the self identification feature*

Before a sector can be read the device must be ready to accept the command. This is done by reading the status register and ensuring that the busy bit is not asserted and that the ready bit is asserted. In this system the status register is polled continuously until these bits indicate the correct condition.

When a sector is to be read the address of the sector must be set up in the IDE registers.

This is done by writing the appropriate data into the cylinder high and low registers, the start sector registers, the head and device register and the sector counter register. The addressing scheme will be discussed below.

When the address has been set up the read sector command is written into the command register. Processing of the command commences immediately after the device receives the command. When the data is available the device asserts the DRQ bit in the status register. In this system the status register is polled continually until the DRQ bit is asserted. The busy and ready bits should also be polled as the specification does not guarantee that the device will be ready until these bits are at the correct values. When the data is ready it may be read from the data register. The data register must be read exactly 256 times to retrieve all of the data. It is mandatory to read all of the data from the buffer. The process of reading a sector is represented diagrammatically for clarity:

University of Cape Town

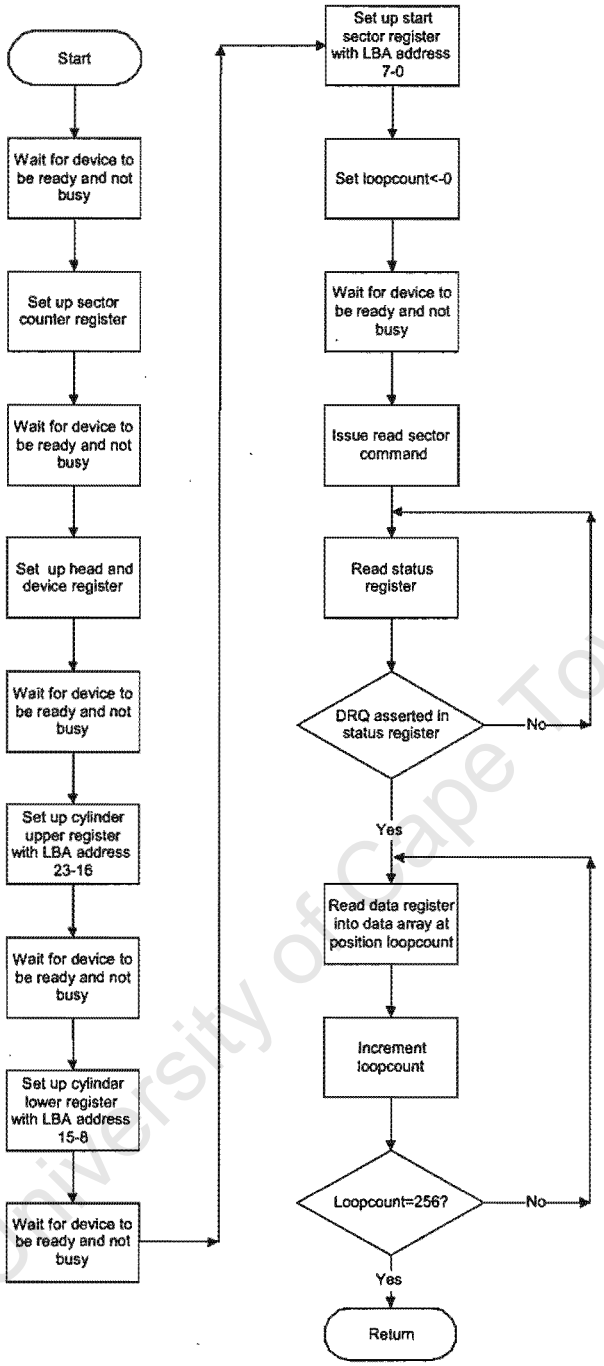



*Figure 28: Operation of the read sector feature*

Before writing to any sector the device must be ready to accept the command. This is checked by examining the ready and busy bits in the status register until they reflect the

University of Cape Town

appropriate status.

The address of the sector to be written to must first be set up in the cylinder high and low registers, the start sector register, the head and device register and the sector count register. The addressing system will be discussed below.

The write sector command is written into the command register. The status register is polled until the DRQ bit indicates that the device is ready to receive data to be stored. The ready and busy bits must also be checked to ensure that the device is fully ready.

The data to be written into the sector is written to the data register in 256 consecutive writes. The data is loaded into the buffer in the same order that it will be retrieved.

The system for writing a sector is shown diagrammatically:

University of Cape Town

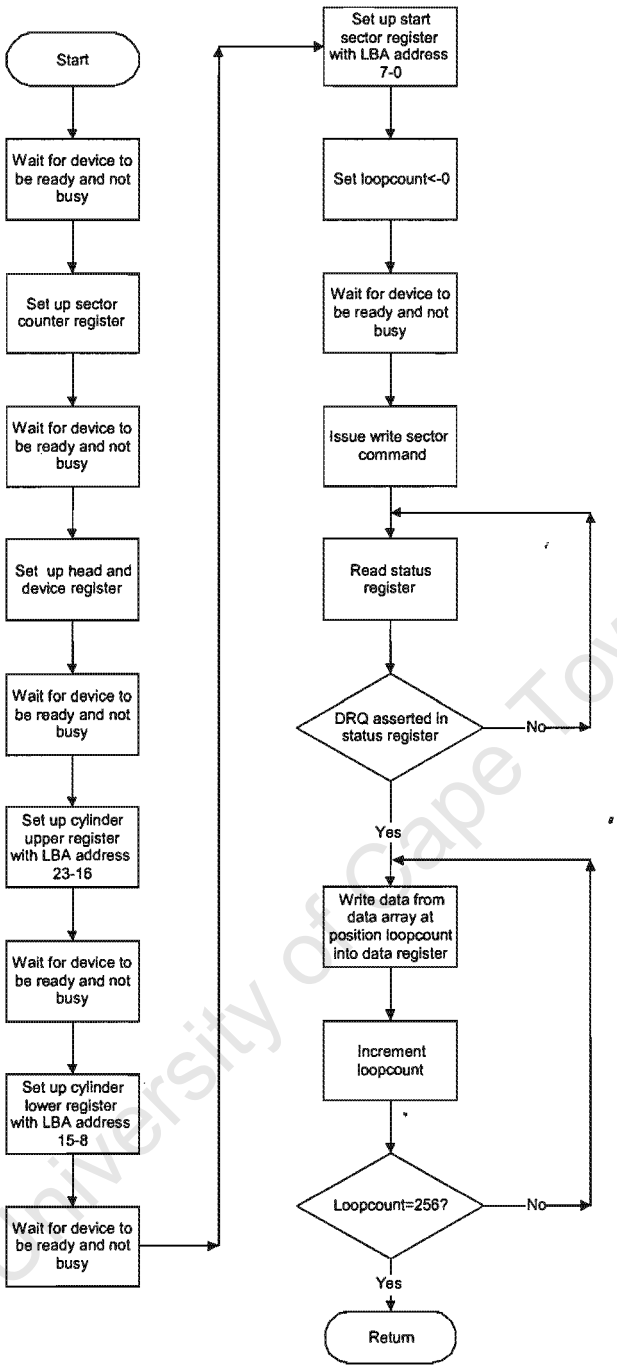



*Figure 29: Operation of the write sector operation*

University of Cape Town

### 7.8.2.5 Sector Addressing Systems

The data on the device must be addressed correctly and consistently in order to always retrieve and store data in the intended location.

The device is only sector addressable. Each address references a sector of data which contains 512 bytes of data, arranged as 256 16 bit words.

For historical reasons there are two major addressing systems in IDE systems. The older system is known as the CHS system. CHS stands for Cylinder Head and Sector. The newer system is known as LBA, which stands for logical block addressing [18]. All new CompactFlash devices support LBA mode [32] and so only this mode is used as it is more suited to the application.

In the LBA mode (also known as the LBA translation mode) the sectors are organized linearly and are numbered from zero to the number of sectors available-1. The address is set up in the cylinder registers, sector and head registers. These registers take on modified meaning in the LBA mode. The new meanings are:

Start Sector register: Address bits 0-7

Cylinder Low Register: Address bits 8-15

Cylinder High Register: Address bits 16-23

Head and Device Register: Address bits 24-27

The address is therefore composed of the concatenation of the address registers. In this system the head and device register is not used to address the device, and so its value remains constant at $E0_{16}$. The remaining registers give 24 bits of address space, which means that a total of $2^{24}$sectors*512 bytes per sector = 8GB of memory may be addressed. This is vastly more than will ever be needed in this application.

Details of the CHS addressing system may be found in the references [18] and [35].

## 7.9 Flash File System

The system is capable of storing large numbers of sets of vibration data. These may be

University of Cape Town

individual records or they may arranged into routes. This calls for an efficient filing system.

Before the file system was designed some existing file systems were examined. These file systems include the FAT16 and FAT32 file systems used by popular personal computer operating systems. The examination revealed that these systems were not suitable for a number of reasons. The other file systems allow for features such as full hierarchical filing (folders within folders within folders). The general purpose file systems also allow for non uniform size files. The problem with these systems is that they are complex. This implies a large processing overhead and wastage of storage space. In addition these systems can suffer from problems such as fragmentation.

The nature of the instrument imposes natural restrictions on the data to be stored. By using these restrictions, and by imposing some artificial but not unreasonable restrictions the file system may be reduced to a simple and efficient system.

The instrument needs to store vibration records. These may or may not be associated with a particular route. The instrument needs to store long names for each record, as these serve as user prompts during route walking. The data records are all of identical size. The user will require a listing of records and will wish to create new records and delete old records. There is no need to support functions such as record copying or CompactFlash duplication. The user must be able to name routes. The filenames for routes need not be as descriptive as for records. It must be possible to create and destroy routes and all of their associated records. It is essential that provision be made for file system recognition. The analyzer must be able to recognize that a CompactFlash has been formatted with the correct file system.

In designing the system it was decided that 250 characters would be enough to name each record. 10 characters was decided as the name length for routes. Each route may contain up to 250 records.

### 7.9.1 File System Structure

The basic format for the CompactFlash is as follows:

University of Cape Town

| Card Information Block | Route Entry Table | File Allocation Table | Data Area |
|---|---|---|---|

Increasing Sector Addresses

*Figure 30: Basic Card format*

The card information block contains information about the rest of the card format. This block is exactly one sector big and is located at address zero on the CompactFlash. In FAT16 and FAT32 file systems the zeroth sector is known as the Master Boot Record (MBR) and has specific information written to it. The fact that the card information also uses sector zero means that the instrument can determine quickly and precisely if the card is formatted using the correct system.

The layout of the card information block is as follows:

| ***Word number*** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Contents*** | 'V' | 'I' | 'B' | 'R' | 'A' | 'T' | 'I' | 'O' | 'N' | NULL | NULL | FAT start | FAT end |

*Figure 31: Details of the Card Information Block*

The first 9 words spell out the word "VIBRATION". This allows the instrument to determine if a vibration analyzer card is present. The next two works are empty and are reserved for future expansion. Word 11 holds the address of the start of the FAT area. This is useful for two reasons: It demarcates the end of the Route Entry Table and it is used by file system functions to determine where to start looking for the file allocation table. The FAT end word demarcates the end of the FAT area. This position is also the start of the data area.

The Route Entry Table (RET) holds information about the composition of each route. The entire RET is divided up into route blocks. Each route block is composed of two sectors. These two sectors hold a route name of 10 characters, a word describing the length of the route and a spare word for future expansion. The rest of the space in the route block contains pointers to the routes that make up the block.

University of Cape Town

The structure of the Route Entry Table can be summarized in the following two diagrams:

| ***Sector Number*** | 1 | 3 | 5 | 7 | 9 |
|---|---|---|---|---|---|
| ***Contents*** | Route 1 | Route 2 | Route 3 | Route 4 | Route 5 |

*Figure 32: The Overall structure of the Route Entry Table*

| ***Word Number*** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| ***Contents*** | Name 1 | Name 2 | Name 3 | Name 4 | Name 5 | Name 6 | Name 7 | Name 8 |

| ***Word Number*** | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|
| ***Contents*** | Name 9 | Name 10 | Length | NULL | Pointer 1a | Pointer 1b | Pointer 2a |

| ***Word Number*** | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
|---|---|---|---|---|---|---|---|
| ***Contents*** | Pointer 2b | Pointer 3a | Pointer 3b | Pointer 4a | Pointer 4b | Pointer 5a | Pointer 5b |

*Figure 33: Detail structure of each route entry*

The pointers are each made up of two words, labeled pointer Xa and pointer Xb in the above diagram. Only the lower eight bits in each word are used. Pointer Xa holds the lowest byte of the LBA address of the FAT entry for a record, while pointer Xb holds the next byte of the LBA address of the FAT entry for that record. This allows access to a FAT of 32MB which allows access to a card of over 2GB.

The route entries each consist of 512 words, 12 of which are taken up by the name, route length and a spare word. This leaves 500 words per route, which allows a maximum of 250 records per route.

The File Allocation Table holds the addresses of the actual data records as well as their names. There is exactly one FAT entry for each data record. Each FAT entry occupies

University of Cape Town

one sector. The structure of each FAT entry is as follows:

| *Size* | 1 | 1 | 1 | 1 | 252 |
|---|---|---|---|---|---|
| ***Contents*** | Bit 0-7 of data address | Bit 8-15 of data address | Bit 16-24 of data address | Spare | Name of Record |

*Figure 34: Structure of a File Allocation Table entry*

The FAT entry thus allows the instrument to find the data and to display its name. The name of the record is also used to determine if the record contains valid data. Note that the FAT allows full 24 bit sector addressing and 252 characters of naming. In the extremely unlikely event of the address range proving inadequate, further address range can be easily implemented in the spare word. This could then offer backward compatibility.

The data area is composed of data records arranged contiguously on the CompactFlash. Each data record occupies exactly 33 sectors. The first sector holds information about the record, while the remaining 32 sectors (32*256=8192 words) hold the actual vibration data. The words in the information sector have the following meanings:

| *Word* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| ***Contents*** | H | H | : | M | M | : | S | S | NULL | Y |

| *Word* | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|
| ***Contents*** | Y | / | M | M | / | D | D | NULL | Sample Rate | Vertical Range |

*Figure 35: Structure of the Information Sector for each record*

The day and date information are packaged in a very inefficient format. The reason for this is that there is a large quantity of spare space in the sector and therefore efficiency of storage is of no particular concern. The benefit of this storage format is that the characters require no further processing for screen display. The sample rate is stored in

University of Cape Town

exactly the same format as is used to store the sample rate in the RAM area of the real time clock, as described above. The range is stored as an integer multiplier, exactly corresponding to the range variable used by the autoranging function, also discussed above.

### 7.9.2 File System Operations

There is a large range of possible operations that can be carried out through the file system. These are discussed here.

#### 7.9.2.1 Formatting a CompactFlash

All new CompactFlash cards must be formatted by the instrument before use. Formatting a card prepares the card for use by the file system and destroys all of the information on the card.

The formatting process consists of three stages. These set up the Card Information Block, the Route entry table and the File Allocation Table. In addition the card parameters are displayed for the user's information.

Before the Card Information Block can be set up, the start and end positions of the FAT must be determined so that they can be written into the block. The first parameter that is determined is the size of the card in sectors. This is determined by performing an identify drive command. The command returns a buffer with 256 words of information. Word 1 contains the number of cylinders, word 3 contains the number of heads and word 6 contains the number of sectors per virtual track. The product cylinders*heads*sectors gives the total number of sectors on the media. The number of sectors is divided by 2 and that gives the size of the media in kB for the user display.

The start address of the FAT is calculated with the formula fat_start=floor (number of sectors/1500). The floor function rounds the result of the division down to the nearest

University of Cape Town

integer. The division factor of 1500 was determined by estimating the number of routes that a user would want to store on various sizes of CompactFlash. The smallest CompactFlash card currently available is 16MB and this would give 9 routes. The exact factor is not critical due to the small space requirements for each route.

The number of routes that may be stored is calculated with the formula: number_of_routes=(fat_start-1)/2. The number of routes is displayed for the user.

The end of the FAT area is determined with the formula fat_end=sectors/34+fat_start This formula is designed to give near optimal memory usage. Each record occupies a total of 34 sectors, including information and FAT entry, and so using 1/34 of the sectors for the FAT is nearly optimal. The number of records that may be stored on the card is calculated with the formula total_records=fat_end-fat_start. This information is displayed to the user.

Once fat_start and fat_end have been calculated, the card information block is written.

The next stage in formatting the card is to clear out the route entry table. This is done by writing zeros to every word in all the sectors in the RET.

The final stage in formatting a card involves setting up the FAT area. Each FAT entry is set up with the address of the information sector for that record. All of the records occupy the same space and so this system works correctly and ensures that no fragmentation can occur. The process of setting up the addresses is simple. The first FAT entry has the address of the first record in the data area, and all addresses thereafter are simply 33 sectors after the previous sector.

The fat entries are marked as blank. An entry is defined as blank when the first character in its name is a question mark '?'.

University of Cape Town

### 7.9.2.2 Creating a Record

The first stage in creating a record is to obtain a name for that record. This can either be downloaded from a computer or assigned by the instrument.

The next stage in record creation is to find a FAT entry that is marked as blank. This is done by looking through the FAT sequentially and finding the first entry that is marked

as blank. The address words in the FAT entry are left unchanged and the record name is written into the FAT entry. The address of the FAT entry is returned to the calling function for use in route creation. The sequential search through the FAT is fast because of the relatively small size of the FAT. The speed of the search is further enhanced by the simplicity of the system for marking blank records.

### 7.9.2.3 Deleting a Record

The file system uses a 'lazy deletion' method to delete records. The appropriate fat entry is simply marked as blank. This is done by making the first character in the name field into a question mark. All other information in the FAT entry is left unchanged.
The lazy deletion system is used because of its efficiency. Lazy deletion also allows the possibility of undeleting data that was accidentally deleted.

### 7.9.2.4 Saving Data to a Record

The data to be saved is made up of vibration samples and an information block. The information block is saved first into the first sector allocated to the record. The samples are then saved into the following 32 sectors in chronological order, i.e. The first sample taken occupies the first word of the first sector and the second sample occupies the next word etcetera.

University of Cape Town

### 7.9.2.5 Recalling Data from a Record

The information sector is read first and the information is used to set the instrument up and for time and date stamping of data. The remaining 32 sectors of data are copied into the data arrays for use by the instrument.
When a record is recalled from memory the global full flag is set to indicate to the analog to digital converter that it must not overwrite the recalled data.

### 7.9.2.6 Creating a Route

The first step in creating a route is to search through the route entry table and find a blank route. This is done by a sequential search, which is fast for the small table size that is searched. If no free routes are found an error message is displayed for the user and the operation is aborted. Free routes are marked by having a name composed entirely of nulls.

Once a free route has been found the instrument gets 12 bytes of data in through its serial port. This data contains a 10 letter name for the route, the number of records in the route and a spare byte that may be used for future expansion. This data is stored on the CompactFlash. The instrument then sends out a character from its serial port to inform the host that the last set of data has been processed.

This acknowledgment is vital as the amount of time needed to write the data to the CompactFlash may vary and if it is too long it may cause the instrument to miss data coming in from the serial port.

The record names are then sent to the instrument, one after the other. The instrument sends acknowledgment after each record name.

For every record name that is received by the instrument the instrument creates a record. The address of the FAT entry for that record is returned to the route creation function. This address is written into a pointer in the route entry table.

The procedure is complicated slightly by the fact that the route entry for each route is spread across two sectors. The route creation algorithm must take this into account, as well as the fact that the second sector is filled entirely with pointers, while the first sector contains other information too.

### 7.9.2.7 Deleting a Route

When deleting a route there are two basic phases. The first phase is deletion of all the

University of Cape Town

individual records in the route. Only once all of the records in the route have been removed may the route entry be deleted.

The record deletion is achieved by working through the route entry, obtaining the FAT addresses of the records and marking these records as blank, as for normal route deletion. When all the routes have been deleted, the entire route entry is overwritten with nulls. This effectively marks the route entry as unused.

University of Cape Town

# Chapter 8

# The Complete Software System

## 8.1 How the Functions Interact

The previous chapter discussed the instrument's functions in detail. There was no indication of how these may be combined to form one integrated software system. The purpose of this chapter is to explain the overall instrument operation modes and to show how these are implemented.

The integrating functions play a key role in the overall system. An instrument is almost useless if it is cumbersome to use. Users have expectations about how the system will work. These expectations are based on industry standards. The instrument's operation is based on these standards, making it similar to other instruments on the market.

### 8.1.1 Instrument Operation

The instrument is menu operated. Each menu item is numbered. The keys on the keypad are also numbered. Menu items are accessed by pressing the appropriate number. Where applicable the current menu selection is marked with an asterisk. The menu system is multi layered. The multi level menu allows the user's options to be organized into meaningful groupings.

An example of a menu screen is shown here:

University of Cape Town

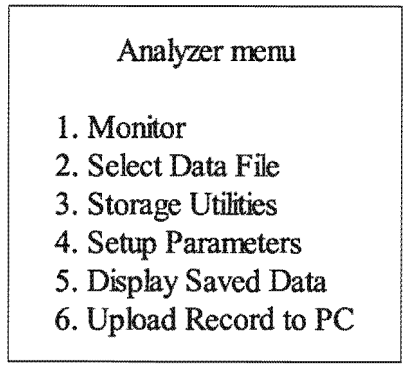



*Figure 36: Example of a user menu screen*

When the user boots up the instrument they are shown the main menu. This menu allows the user to monitor vibrations, manage the CompactFlash card, set the instrument up, view previously saved data or upload data to a computer there is also an option to select a data file. This option was included in the main menu for ease of access and it is duplicated elsewhere in the menu structure.

If the user chooses to monitor a vibration they are shown another menu and they have the choice of monitoring the vibration in the time or frequency domain.

If the user wishes to work with the CompactFlash card they are shown another menu with the CompactFlash utilities. These allow the user to format the flash, create and destroy records, select an active record or to load a record from a computer. In addition the user may choose to use the route facilities of the instrument. The route facilities allow the user to create and destroy routes, select a route and walk a route.

The instrument parameters are configured by choosing the 'setup parameters' option from the main menu. This causes another menu to be displayed. The setup parameters menu allows the user to set up the window type used for the Fast Fourier Transform, allows the time and date to be set, allows the user to set up the vertical and horizontal ranges used by the instrument and configures the instrument for acceleration, velocity or displacement display.

When data is recalled from memory the user is given the option to view that data in either the time or frequency domain.

Within the vibration monitoring displays the user is also offered various options. These

University of Cape Town

are available as hotkey functions and are discussed above, in the section on hotkey support.

### 8.1.2 Menu System Details

The entire menu system is contained in the file ana.c. All of the menu functions are written in C.

The main menu is contained in the function called menu1. The menu functions provide keypad management functions as well. The menu is responsible for keypad debouncing, as well as anti shoot through logic. The anti shoot through logic is important because a user may hold a key down for an extended period. The anti shoot through logic ensures that only one command will be executed for every separate keypress.

The main menu calls additional menu functions. The function which displays the monitor menu is called menu2. The setup parameters function is in menu3. The menu to display saved data is in menu4. Menu 5 facilitates the upload of data to a computer. Menu6 leads the user to the CompactFlash utilities and menu7 is used to select a data file for access.

The overall menu structure is shown in the following menu tree diagram.

University of Cape Town

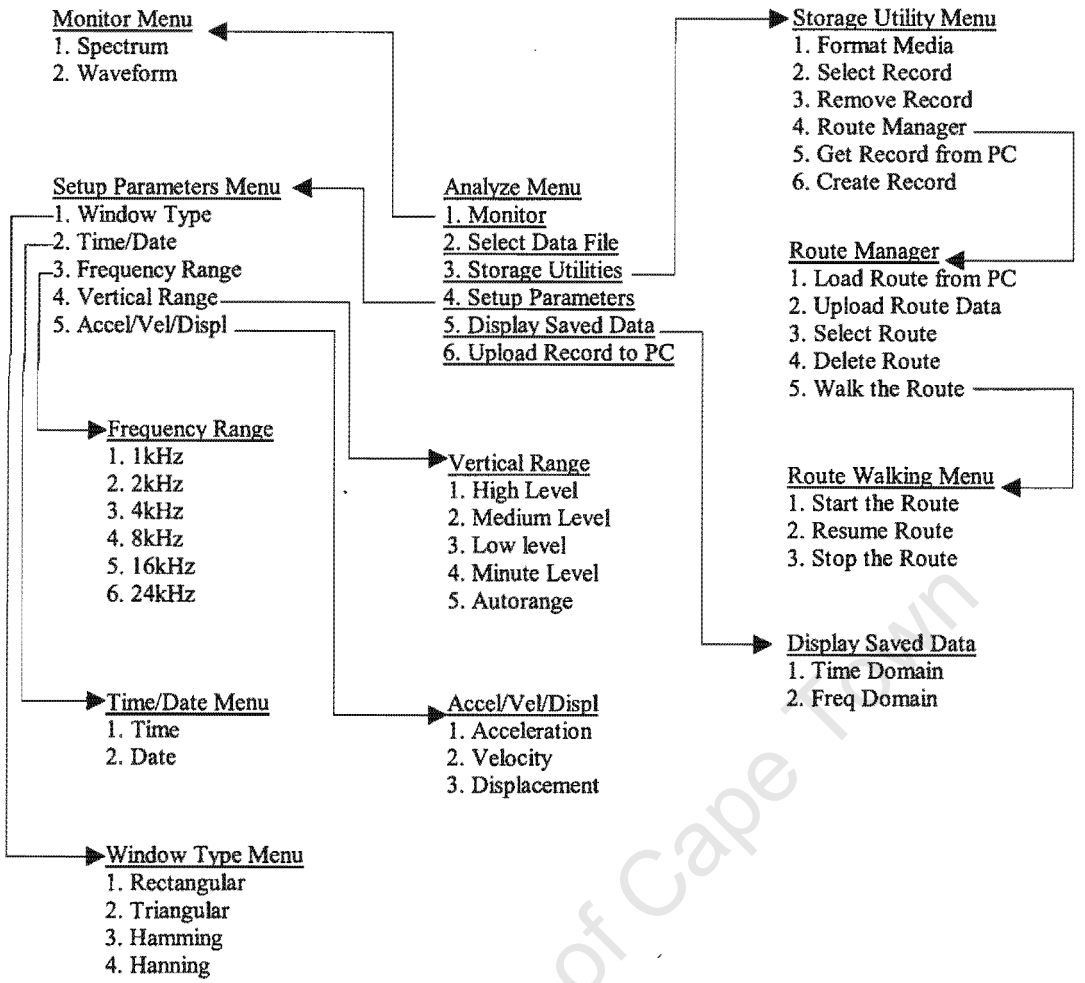



*Figure 37: The instrument's menu structure*

University of Cape Town

### 8.1.3 Main Loop Operation and System States

The overall system operates in an infinite loop. This structure is most appropriate as the system is continually performing the same set of operations as long as it is powered up. This loop is divided into two sections. There is a section that is performed during every iteration and a section that is only performed when new vibration data is available. The loop determines if new data is available by examining the full flag that is set by the ADC interrupt handler.

The instrument has five states. There is a state in which the user is using the menu

system, a state in which the user is monitoring vibrations in the time domain, a state in which the user is monitoring a vibration in the frequency domain, a state in which a stored waveform is being recalled and displayed in the time domain and a state in which stored waveforms are displayed in the frequency domain. The current state of the instrument is reflected in a variable called state. Each state is represented as a different number. The previous state of the instrument is stored in a variable called old_state. The use of a previous state variable makes it possible to return to a previous state.

The infinite loop examines the keypad and the state variable and decides what action to take. This means that the entire main loop may be modeled as a state machine whose transitions depend on keypad input.

The main loop can be simplified into the following diagram:

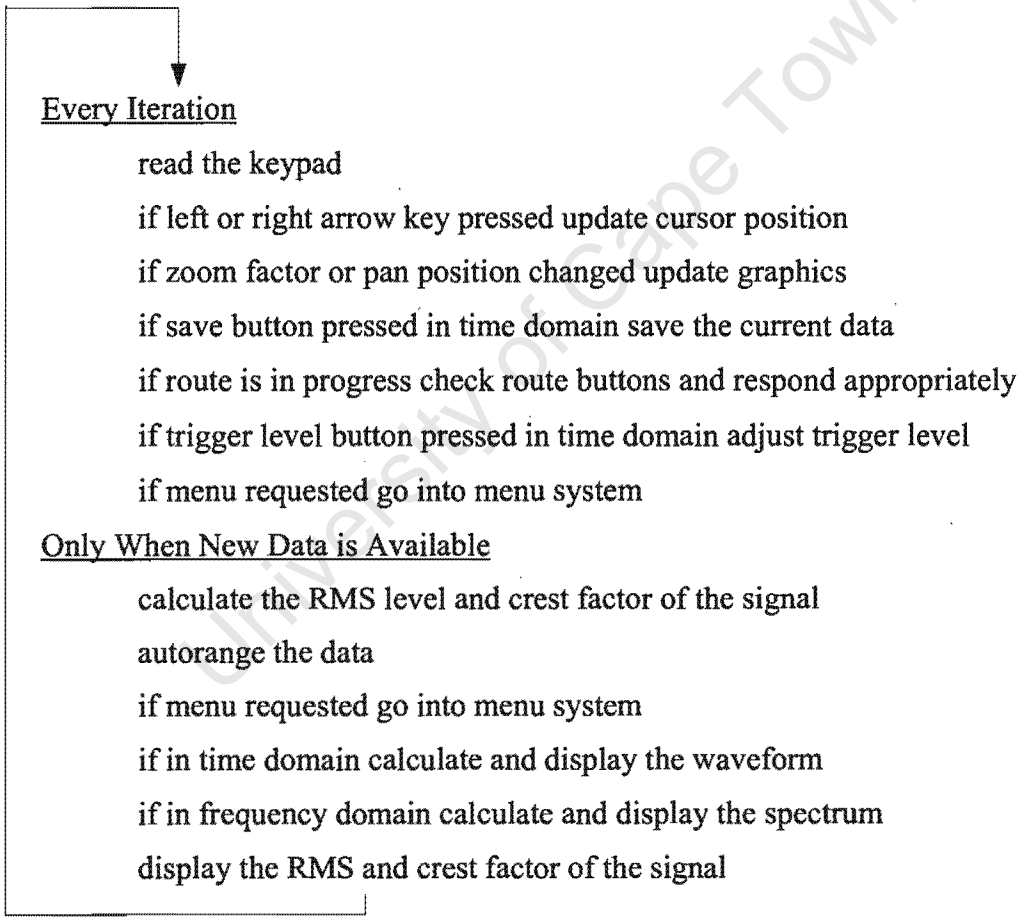



*Figure 38: Basic operation of the main loop*

University of Cape Town

# Chapter 9

# Hardware Development Systems

## 9.1 Digital Signal Processor Development Kit

The development environment used for the Digital Signal Processor (DSP) is the ADSP21xx EZ-KIT from Analog Devices.

The kit consists of a circuit board and software. The circuit board has a DSP and a CODEC on it. There are connectors which give access to all of the DSP's expansion connections. On the board is a monitor ROM as well.

The software consists of an assembler, compiler, linker, ROM splitter utility and an upload program.

For smaller programs the program is written, compiled, linked and sent to the upload program. The hardware is specified by means of the system builder utility. The upload program then sends the compiled code over a serial link to the board where the monitor system loads the code into memory and triggers execution.

The monitor occupies 2kwords of program memory, out of a total of 16kwords. This is a real limitation when the code requires more than 14kwords.

The software was initially developed using the monitor and download program system. When this became unfeasible new systems were investigated. These ideas are given here.

Modifications were made to the monitor software. The monitor was reduced in complexity and size in the hope that this would free up program memory. This approach

University of Cape Town

failed because the download software limits the code size even when the monitor is reduced in size. The difficulties of creating of software to interact with the DSP system was not justified for the small benefit that it would offer.

The next idea that was experimented with was a ROM emulator. Several ROM emulators were available, but all of these have too small a capacity to be of any use. If a higher capacity emulator were available this would be an excellent method of development.

Ultimately flash memory was used to develop the final code. The code was written in assembler and C and the hardware defined with the system builder. The assembler files were assembled and the C files were compiled. The object code was then linked and sent to the ROM splitter. The ROM splitter adds extra code to the beginning of the object code. This extra code implements a boot loader. The boot loader is responsible for the transfer of code from the ROM into the DSP's Flash memory when the system is booted.

The Flash memory was then programmed with the object code using an EMP10 device programmer. The device used is an Atmel AT29C010A [10].

### 9.1.1 System Builder Usage

The system builder utility is called bld21. This program reads in a file which specifies the memory structure of the hardware system and converts it to an architecture file. This architecture file is used by the linker to locate code correctly in the DSP's memory.

The essential elements in the system builder file are as follows:

```
.seg/pm/ram/abs=0/code/data int_pm[16384];
.seg/dm/ram/abs=0/data int_dm[16351];
```

The first line specifies that the program memory is 16kwords in size and that it starts at address 0. The second line specifies that there are 16351 words of data memory, starting at address 0.

The system builder is invoked with the command bld21 <filename>.

University of Cape Town

### 9.1.2 Assembler Usage

The assembler used is called asm21. This is produced by Analog Devices for the ADSP218x family of Digital Signal Processors.
The assembler is invoked with the command:
asm21 <filename> -c -2181
The '-c' argument instructs the assembler to be case sensitive. This is required for compatibility with the C compiler. The -2181 directive informs the assembler that the ADSP2181 DSP is to be used. This is important as the ADSP2181 has extra assembly instructions that are not used by other ADSP21xx family members.
There is an argument to set the output file name. This is used for assembling the header file and is necessary for correct linking. This is explained below.

### 9.1.3 C Compiler Usage

The C compiler used is called g21. This is an ANSI C compliant C compiler produced by Analog Devices for the ADSP218x family of digital Signal Processors.
The compiler is invoked with the command
g21 <filename> -c -mreserved=i2,i3
The '-c' option instructs the compiler to generate assembly object files only. This is needed because the linking of the code occurs at a later stage.
The '-mreserved=i2,i3' argument instructs the compiler not to use the i2,l2,i3 and l3 registers in the processor because they are reserved for use by the ADC system in this application. The reasons for this are explained in the section on ADC support.

### 9.1.4 Linker Usage

The linker is called ld21. This program's operation is specified in a group file. The command to invoke the linker and make it use a group file called ana.grp is given here:



University of Cape Town

ld21 -group ana.grp

The group file is given in appendix C.2. This file lists all of the object code files that must be linked together. The group file also specifies the architecture file name and the C library names.

The header file used by the linker must be named run_hdr. The assembler in thus instructed to assemble the header file into an object file called run_hdr.

### 9.1.5 ROM Splitter Usage

The ROM splitter is responsible for converting linked object files to a format suitable for programming onto a ROM device. The splitter is called spl21 and it is invoked with the following command

spl21 ana anarom -loader -2181 -i

This command tells the splitter to convert the file called ana to a file called anarom. The -i switch specifies that the output file must be in the popular Intel HEX format. The argument -loader -2181 specify that the splitter must add in a boot loader to the file and that the processor being used is the ADSP2181.

## 9.2 Programmable Logic Development Kit

The programmable logic device (PLD) needs a development system. This system consists of a design entry tool, a compiler and a programming utility. A hardware system is needed to form the physical link between the programming software and the device itself. The PLD is the EPM7064SLC84 from Altera. Altera's MAX PLUS II version 9.5 Baseline was used as the development environment. See reference [3] for further information. The internal logic for the device was set up using the schematic entry mode. This was then compiled under MAX PLUS II.

The programming interface to the PLD is via a JTAG IEEE 1149.1 interface. The programmer was constructed according to the details given by Altera in their ByteBlaster

University of Cape Town

data sheet [1].

The target board has a JTAG connector on it and this allows in circuit programming of the programmable logic.

University of Cape Town

# Chapter 10

# Conclusions

## 10.0 Test Results

### 10.0.1 Simulation Testing

Throughout the instrument's development it was necessary to verify that signals that are applied to the instrument's input are represented correctly. In order to facilitate this signals were generated with a function generator and fed into the system's input.

The advantages of this approach are numerous. The input is repeatable and results are easily verified. The application of simple input waveforms is conducive to ease of verification, particularly when developing the frequency domain subsystems, such as the Fast Fourier Transform. In addition the simplicity of the input waveform allows for easy confirmation of the numerical data produced by the instrument.

University of Cape Town

### 10.0.2 Machinery Based Testing

Ultimately the instrument must be capable of producing useful output based on input recorded from real machinery. This was seen as a final step, as the basic system operation had already been verified by simulation.

The first stage in this testing involved connecting an accelerometer to the system input. The accelerometer used was an integrated circuit containing a micromachined accelerometer as well as its initial stages of signal conditioning. The specific device used was the ADXL105 manufactured by Analog Devices. This accelerometer was encased in a housing which was rigidly attached to a permanent magnet. The accelerometer device was powered by the instrument's internal power supply.

The accelerometer was magnetically attached to a four machines in turn. The machines included a single phase induction machine, a three phase induction machine and an induction machine driven

from a variable speed drive. The results were examined and found to meet expectations, although these tests do not constitute an exhaustive proof of the instrument's correctness of operation. The reader is referred to page 195 for an illustration of a typical instrument output.

## 10.1 Fulfillment of Requirements

The instrument that was developed is a functional portable vibration analyzer.

The instrument is small enough to be held in one hand. There are pictures of the completed unit in the appendix F. The entire unit runs off batteries which fit into the instrument housing. A facility for recharging the batteries without removing the battery pack has been provided. The instrument has been designed into a casing which is rugged enough to withstand the industrial environment in which the analyzer will be used.

The instrument has a sampling rate of up to 48kSa/s. This is deemed adequate for capturing vibration waveforms faithfully.

The instrument has an automatic range selection feature. This makes it easier for the user to take measurements and boosts productivity.

The instrument has enough processing power to perform the required mathematical operations in an acceptably short time.

The instrument has facilities for monitoring vibrations in the time and frequency domains. The system can store and recall large numbers of waveforms, and when recalling data it can present the data in either the time or the frequency domain.

The instrument can upload recorded data to a computer via a standard serial link. The serial link can also be used to set up the instrument from a computer.

The system has route walking facilities. The route walking system allows automated viewing and recording of waveforms as well as the ability to pause and resume the automated procedure.

The instrument is configurable. The major measurement parameters may be configured independently through the menu system.

Great care has been paid to the usability of the instrument. The system is menu driven and is similar to other instruments in its operational procedures. The menu system is supplemented by the hotkey system which provides an additional intuitive interface.

## 10.2 System Deficiencies

The system is not perfect. There are several deficiencies that should be corrected if the instrument is to be viable in the future.

University of Cape Town

### 10.2.1 Single Sensor Only

The instrument only allows a single accelerometer to be connected. The hardware exists for two accelerometers to be connected, but there is no software to support this.

The advantage of having two accelerometers is that it would allow the user to view lissajous phase plots of the vibration waveform. This could aid in fault diagnosis [22, p125]. The use of two accelerometers would open up the potential for an automatic balancing system for rotating machines. This could make the instrument significantly more useful.

It should be noted that most of the advantages in having two sensors lie in the time domain representation of data from those sensors. This means that the instrument could use fewer samples per sensor when reading two sensors. This would allow the system to use the same amount of data memory as it does at present.

Another sensor that would be extremely useful is a tachometer. The hardware to interface to a tachometer exists on the main board, but there is no software to support this. The use of a tachometer would allow the instrument to present its measurements in relation to the machine's speed instead of simply presenting results in absolute frequency terms. The tachometer would also allow the instrument to determine the phase relations present in the vibration. This would make the instrument more useful as a diagnostic tool.

### 10.2.2 Poor Computer Based Support

The instrument features upload facilities. The instrument also uses a personal computer for route setup and record naming. At present there is no software suite to run on a personal computer which satisfies these needs. The software would have to be developed before these features are truly useful. This was not undertaken as it was outside the scope of this project.

### 10.2.3 Limited Program Memory

The program memory is large enough for the system in its present state, but the future expansion of the instrument will be hampered by lack of program memory. The code was optimized for size during development and despite this the program occupies approximately 95% of the total available memory.

University of Cape Town

### 10.2.4 Difficulty in Obtaining CODEC Devices

The CODEC device that is used in the analyzer has recently been declared obsolete. The manufacturers are still able to supply these devices, but they insist on large volume orders only. Several companies are able to source the integrated circuit in low volumes but the lead times are long and the pricing is more than double the price of similar devices from other product ranges [40].

### 10.2.5 High Power Consumption

The total current consumption of the unit is 350mA. Of this total approximately 140mA is used by the CODEC system. Other similar CODECs offer current consumption below 40mA [40]. This would decrease the current consumption of the instrument by 28%, resulting in a longer battery life, or alternately a lighter, more ergonomic instrument.

The power consumption could be reduced, and the CODEC replaced by simply changing the analog board. These changes need not affect the main system board. There are software implications to changing the analog board, unless another CODEC with a very similar serial interface can be sourced.

University of Cape Town

### 10.2.6 Lack of Power Management

The instrument lacks a full power management system. Without this system the instrument cannot perform functions such as automatic shutdown or battery charge indication. The battery charging process is limited to a standard slow charge as there is no support in the instrument for a fast charging strategy.

The power management board would be an extra board, and need not affect the main system board.

### 10.2.7 Too Few Sample Rates Available

The analyzer offers a very high resolution Fast Fourier Transform. The analyzer is capable of giving very high frequency resolution if the sample rate could be lowered. This would be of great use in discriminating between line frequency related vibrations and shaft frequency related vibrations on slow turning machines. The lowest sample rate available is 2kSa/S which gives a frequency resolution of 244 mHz.

# Chapter 11

# Recommendations

The instrument could be improved and moved closer to an industrially acceptable system by making the following improvements.

## 11.1 Addition of Sensors

The CODEC allows for two accelerometers to be attached and read simultaneously. This should be done and the software updated to gain advantage from the extra sensor. The software update should at least allow the instrument to produce lissajous plots from the sensors.

The programmable logic device has spare pins which should be used to read in a tachometer. The software should be upgraded to make use of the extra information which may be gathered via this sensor. The upgrade should allow the instrument to display all frequency components relative to the shaft speed and should also enable the machine to display phase information.

These changes will make the instrument's sensory abilities compete with the finest systems available today.

University of Cape Town

## 11.2 Creation of a Software Suite

The instrument is not fully enabled without a computer based software suite. This software should be developed and bundled together with the instrument so that users may make use of all of the instrument's functionality. The suite should include a facility for creating record names and downloading them to the instrument. The software should have a route editor which allows users to create complete routes with all of their records and download this information to the instrument.

The software must also upload vibration information from the instrument. At the very minimum this information should be stored on the computer in an appropriate format. The software could also plot the data and it may be integrated into a larger vibration monitoring system.

## 11.3 Addition of Program Memory

The program memory must be expanded in order to meet future memory demands. There are several feasible ways of accomplishing this.

External memory could be added on to the processor system and interfaced through the expansion bus provided for that purpose. This involves adding 24 bit wide memory on the main board. The other disadvantage is that the processor would then access the memory in a banked fashion, and this would complicate the software.

The processor device used in this family is part of an expanding family of processors. The newer members of the family have more program memory, which they can access in a non-paged manner. Examples of such devices are the ADSP 219x family from Analog Devices. These devices are allegedly code compatible with the current system processor, the ADSP 2181. There would probably be some software implications in the migration and it is extremely likely that the main board would have to be redesigned.

University of Cape Town

## 11.4 Replace the CODEC

The CODEC device is officially obsolete. It consumes more power than any other device in the system. Replacement of the CODEC is recommended. This involves redesigning the analog board around a newer device.

## 11.5 Addition of Power Management

A power management board should be added. This board must include a battery charge level indicator, an automatic shutdown circuit and an intelligent battery charger circuit. The charge level could be indicated on a set of LEDs. The battery voltage should be an adequate indication of the state of charge of the batteries.

The shutdown circuit should monitor the keypad. If no key is pressed for 10 minutes the unit should automatically shut down.

The intelligent charger circuit should monitor the battery temperature and regulate the charging current accordingly. The use of a dedicated battery charger integrated circuit is recommended as these will simplify the system and ensure rapid and safe battery charging.

## 11.6 Addition of Lower Sample Rates

Lower sampling rates should be implemented. This is a software modification only. The lower sampling rate will provide enhanced resolution and will aid in discriminating between electrical and mechanical problems.

University of Cape Town

# References


[1] Altera Corporation. *ByteBlaster Parallel Port Download Cable Data Sheet* Altera Corporation 101 Innovation Drive San Jose CA 95134 also available from www.altera.com/literature/lit-config.html

[2] Altera Corporation. *MAX7000 Programmable Logic Device Family Data Sheet* Altera Corporation 101 Innovation Drive San Jose CA 95134 also available from www.altera.com/literature/lit-m7k.html

[3] Altera Corporation. *Max+Plus II Getting Started* September 1997 Altera Corporation 101 Innovation Drive San Jose CA 95134 also available from www.altera.com/literature/lit-index.html

[4] Analog Devices. *ADSP2100 Family C Runtime Library Manual* 1995 Analog Devices Incorporated Computer Products Division 1 Technology Way P.O. Box 9106 Norwood MA 02062 URL: www.analog.com

[5] Analog Devices. *ADSP2100 Family C Tools Manual* 1995 Analog Devices Incorporated Computer Products Division 1 Technology Way P.O. Box 9106 Norwood MA 02062 URL: www.analog.com

[6] Analog Devices. *ADSP2100 Family EZ-KIT-Lite Reference Manual* 1st Ed 1995 Analog Devices Incorporated Computer Products Division 1 Technology Way P.O. Box 9106 Norwood MA 02062 URL: www.analog.com

[7] Analog Devices. *ADSP2100 Family User's Manual* 3rd Ed September 1995 Analog Devices Incorporated Computer Products Division 1 Technology Way P.O. Box 9106 Norwood MA 02062 URL: www.analog.com

[8] Analog Devices. *Implementing a Software UART on the ADSP2181 EZ-KIT-Lite* undated. Engineer to Engineer Note EE89 www.analog.com/dsp

[9] Analog Devices. *Serial Port 16 Bit SoundPort Stereo CODEC AD1847* Rev 0 Undated Document Number 85-001065-01 available from www.analog.com

[10] Atmel Corporation. *1 Megabit 5 Volt Only Flash Memory AT29C010A* see www.atmel.com

[11] Balfour, Alexander *Programming in Standard FORTRAN 77* 1979 001.6424FORT London: Heinemann Educational

[12] Brigham, E Oran. *The Fast Fourier Transform and its Applications* 1988 515.723BRIG Prentice Hall Englewood Cliffs NJ

[13] Commtest Instruments. *V Series Vibration Analysis Brochure* 23202 Mariposa Avenue Torrance CA 90502-2609 see www.commtest.com/fset1.html


University of Cape Town

[14] CompactFlash Association. *CompactFlash Specification* Version 1.4 1999 Available from www.compactflash.org

[15] Crawford, Arthur R. *The Simplified Handbook of Vibration Analysis Volume 1* 1992 621.811CRAW Computational Systems Incorporated 835 Innovation Drive Knoxville TN 3792

[16] Crawford, Arthur R. *The Simplified Handbook of Vibration Analysis Volume 2* 1992 621.811CRAW Computational Systems Incorporated 835 Innovation Drive Knoxville TN 3792

[17] Dallas Semiconductors. *DS1302 Trickle Charge Timekeeping Chip* Maxim Integrated Products 120 San Gabriel Drive Sunnyvale CA 94086 USA Available from http://pdfserv.maxim-ic.com/arpdf/MAX220-MAX249.pdf

[18] Faase, Peter. *IDE Controller* Undated. Available from http://minyos.its.rmit.edu.au/~s9906768/pic/IDE_to_8255.html

[19] Lawther, Steve. *T6963C Controller Based Displays* Undated Available from http://ourworld.compuserve.com/homepages/steve_lawther/t6963c.htm

[20] Lawther, Steve. *Writing Software for T6963C Based Graphic LCDs* Undated Available from http://ourworld.compuserve.com/homepages/steve_lawther/t6963c.pdf

[21] Lindsey, C.H. *Informal Introduction to Algol 68* 2nd Ed 1977 001.6424ALGO London: North-Holland

[22] Lipovszky, G, Sólyomvári, K, Varga, G. *Vibration Testing of Machines and Their Maintenance* 1990 621.811LIPO Translated by S. Bars and E. Darabant Elsevier Science Publishing Company Incorporated 655 Avenue of the Americas New York 10010 USA

[23] Maxim Semiconductors. *+5V Powered Multichannel RS232 Driver Receivers* Maxim Integrated Products 120 San Gabriel Drive Sunnyvale CA 94086 USA Available from http://pdfserv.maxim-ic.com/arpdf/MAX220-MAX249.pdf

[24] Maxim Semiconductors. *+5V to ±10V Voltage Converters MAX680/MAX681* Maxim Integrated Products 120 San Gabriel Drive Sunnyvale CA 94086 USA Available from http://pdfserv.maxim-ic.com/arpdf/MAX680-MAX681.pdf

[25] Maxim Semiconductors. *Preset/Adjustable Output CMOS Inverting Switching Regulators MAX635/MAX636/MAX637* Maxim Integrated Products 120 San Gabriel Drive Sunnyvale CA 94086 USA Available from http://pdfserv.maxim-ic.com/arpdf/MAX220-MAX249.pdf

[26] Morrison, Norman I. *Introduction to Fourier Analysis* 1994 515.2433MORR Wiley Interscience New York

[27] National Semiconductor. *CD4051BM/CD4051BC, CD4052BM/CD4052BC, CD4053BM/CD4053BC Analog Multiplexer Demultiplexer* National Semiconductor Corporation 2900 Semiconductor Drive P.O. Box 58090 Santa Clara CA 95052 also available from www.national.com/ds/CD/CD4051BC.pdf

University of Cape Town

[28] National Semiconductor. *LM2941/LM2941C 1A Low Dropout Adjustable Regulator* National Semiconductor Corporation 2900 Semiconductor Drive P.O. Box 58090 Santa Clara CA 95052 also available from www.national.com/pf/LM/LM2941C.html

[29] OKI Semiconductor. *MSM82C55A CMOS Programmable Peripheral Interface* Obsolete data sheet. Available from www.okisemi.com/communicator/public/nf/docs/intro-6110.html

[30] OKW Gehause Systeme. *Datec Control Series Catalog* Available from www2.okw.com/engl/prod.htm

[31] Rao, B.K.N. *Handbook of Condition Monitoring* 1st Ed 1996 621.816RAO Elsevier Science Ltd. The Boulevard, Langford Lane Kidlington Oxford, OX5 1GB UK

[32] SanDisk Corporation. *CompactFlash Memory Card Product Manual* SanDisk Corporation 140 Caspian Court Sunnyvale CA 94089 also available from www.sandisk.com

[33] SanDisk Corporation. *Interfacing SanDisk CompactFlash Cards to an 80C51 Microcontroller* SanDisk Corporation 140 Caspian Court Sunnyvale CA 94089 also available from www.sandisk.com

[34] SanDisk Corporation. *SanDisk CompactFlash & Motorola 8 Bit Microcontroller Interface Design Reference Example* SanDisk Corporation 140 Caspian Court Sunnyvale CA 94089 also available from www.sandisk.com

[35] Schmidt, Friedhelm. *The SCSI Bus and IDE Interface: Protocols, Applications and Programming* 1998 2nd Ed 004.62SCHM Addison Wesley

[36] Stoffregen, Paul. *Using an IDE Hard Drive with an 8051 board and 82C55* 2000 available from www.pjrc.com/tech/8051/ide/

[37] Stremler F.G. *Introduction to Communication Systems* 3rd Ed 1990 621.382STRE Addison Wesley page542-543

[38] Taylor, James I. *The Vibration Analysis Handbook* 1994 621.811TAYL Vibration Consultants Incorporated 5733 South Dale Mabry Highway Tampa FL

[39] Technical committee T13 AT Attachment. *2008D AT Attatchment-3 Interface (ATA3)* Available from ANSI 11 West 42nd Street New York NY 10036 or www.ansi.org

[40] Texas Instruments. *Audio Band DSP CODEC offers Variable Sampling Rate to 48kHz* Texas Instruments tech innovations Volume 5 Available from www.ti.com/sc/techinnovations5

[41] Toshiba Semiconductors. *T6963C Dot Matrix LCD Control LSI* available from http://doc.semicon.toshiba.co.jp/noseek/us/td/03frame.htm

[42] Williams, J.H, Davies, A, Drake, P.R. *Condition Based Maintenance and Machine Diagnostics* 1994 621.816WILL Chapman & Hall

University of Cape Town

# Bibliography


[1] Altera Corporation. *ByteBlaster Parallel Port Download Cable Data Sheet* Altera Corporation 101 Innovation Drive San Jose CA 95134 also available from www.altera.com/literature/lit-config.html

[2] Altera Corporation. *MAX7000 Programmable Logic Device Family Data Sheet* Altera Corporation 101 Innovation Drive San Jose CA 95134 also available from www.altera.com/literature/lit-m7k.html

[3] Altera Corporation. *Max+Plus II Getting Started* September 1997 Altera Corporation 101 Innovation Drive San Jose CA 95134 also available from www.altera.com/literature/lit-index.html

[4] Analog Devices. *ADSP2100 Family C Runtime Library Manual* 1995 Analog Devices Incorporated Computer Products Division 1 Technology Way P.O. Box 9106 Norwood MA 02062 URL: www.analog.com

[5] Analog Devices. *ADSP2100 Family C Tools Manual* 1995 Analog Devices Incorporated Computer Products Division 1 Technology Way P.O. Box 9106 Norwood MA 02062 URL: www.analog.com

[6] Analog Devices. *ADSP2100 Family EZ-KIT-Lite Reference Manual* 1st Ed 1995 Analog Devices Incorporated Computer Products Division 1 Technology Way P.O. Box 9106 Norwood MA 02062 URL: www.analog.com

[7] Analog Devices. *ADSP2100 Family User's Manual* 3rd Ed September 1995 Analog Devices Incorporated Computer Products Division 1 Technology Way P.O. Box 9106 Norwood MA 02062 URL: www.analog.com

[8] Analog Devices. *Implementing a Software UART on the ADSP2181 EZ-KIT-Lite* undated. Engineer to Engineer Note EE89 www.analog.com/dsp

[9] Analog Devices. *Serial Port 16 Bit SoundPort Stereo CODEC AD1847* Rev 0 Undated Document Number 85-001065-01 available from www.analog.com

[10] Atmel Corporation. *1 Megabit 5 Volt Only Flash Memory AT29C010A* see www.atmel.com

[11] Balfour, Alexander *Programming in Standard FORTRAN 77* 1979 001.6424FORT London: Heinemann Educational

[12] Brigham, E Oran. *The Fast Fourier Transform and its Applications* 1988 515.723BRIG Prentice Hall Englewood Cliffs NJ

[13] Commtest Instruments. *V Series Vibration Analysis Brochure* 23202 Mariposa Avenue Torrance CA 90502-2609 see www.commtest.com/fset1.html


University of Cape Town

[14] CompactFlash Association. *CompactFlash Specification* Version 1.4 1999 Available from www.compactflash.org

[15] Crawford, Arthur R. *The Simplified Handbook of Vibration Analysis Volume 1*1992 621.811CRAW Computational Systems Incorporated 835 Innovation Drive Knoxville TN 3792

[16] Crawford, Arthur R. *The Simplified Handbook of Vibration Analysis Volume 2* 1992 621.811CRAW Computational Systems Incorporated 835 Innovation Drive Knoxville TN 3792

[17] Dallas Semiconductors. *DS1302 Trickle Charge Timekeeping Chip* Maxim Integrated Products 120 San Gabriel Drive Sunnyvale CA 94086 USA Available from http://pdfserv.maxim-ic.com/arpdf/MAX220-MAX249.pdf

[18] Damper, RI. *Introduction to Discrete Time Signals and Systems* 1995 Chapman and Hall 2-6 Boundry Row London SE1 8HN, UK

[19] Faase, Peter. *IDE Controller* Undated. Available from http://minyos.its.rmit.edu.au/~s9906768/pic/IDE_to_8255.html

[20] Halliday, David. Resnick, Robert. Walker, Jearl. *The Fundamentals of Physics* 4th Ed 1993 John Wiley and Sons Inc.

[21] Horowitz, Paul. Hill, Winfield. *The Art of Electronics* 2nd Ed 1993 Cambridge University Press 40 West 20th Street, New York NY 10011-4211, USA 621.381HORO

[22] Lawther, Steve. *T6963C Controller Based Displays* Undated Available from http://ourworld.compuserve.com/homepages/steve_lawther/t6963c.htm

[23] Lawther, Steve. *Writing Software for T6963C Based Graphic LCDs* Undated Available from http://ourworld.compuserve.com/homepages/steve_lawther/t6963c.pdf

[24] Lindsey, C.H. *Informal Introduction to Algol 68* 2nd Ed 1977 001.6424ALGO London: North-Holland

[25] Lipovszky, G, Sólyomvári, K, Varga, G. *Vibration Testing of Machines and Their Maintenance* 1990 621.811LIPO Translated by S. Bars and E. Darabant Elsevier Science Publishing Company Incorporated 655 Avenue of the Americas New York 10010 USA

[26] Maxim Semiconductors. *+5V Powered Multichannel RS232 Driver Receivers* Maxim Integrated Products 120 San Gabriel Drive Sunnyvale CA 94086 USA Available from http://pdfserv.maxim-ic.com/arpdf/MAX220-MAX249.pdf

[27] Maxim Semiconductors. *+5V to ±10V Voltage Converters MAX680/MAX681* Maxim Integrated Products 120 San Gabriel Drive Sunnyvale CA 94086 USA Available from http://pdfserv.maxim-ic.com/arpdf/MAX680-MAX681.pdf

University of Cape Town

[28] Maxim Semiconductors. *Preset/Adjustable Output CMOS Inverting Switching Regulators MAX635/MAX636/MAX637* Maxim Integrated Products 120 San Gabriel Drive Sunnyvale CA 94086 USA Available from http://pdfserv.maxim-ic.com/arpdf/MAX220-MAX249.pdf

[29] Morrison, Norman I. *Introduction to Fourier Analysis* 1994 515.2433MORR Wiley Interscience New York

[30] National Semiconductor. *CD4051BM/CD4051BC, CD4052BM/CD4052BC, CD4053BM/CD4053BC Analog Multiplexer Demultiplexer* National Semiconductor Corporation 2900 Semiconductor Drive P.O. Box 58090 Santa Clara CA 95052 also available from www.national.com/ds/CD/CD4051BC.pdf

[31] National Semiconductor. *LM2941/LM2941C 1A Low Dropout Adjustable Regulator* National Semiconductor Corporation 2900 Semiconductor Drive P.O. Box 58090 Santa Clara CA 95052 also available from www.national.com/pf/LM/LM2941C.html

[32] OKI Semiconductor. *MSM82C55A CMOS Programmable Peripheral Interface* Obsolete data sheet. Available from www.okisemi.com/communicator/public/nf/docs/intro-6110.html

[33] OKW Gehause Systeme. *Datec Control Series Catalog* Available from www2.okw.com/engl/prod.htm

[34] Rao, B.K.N. *Handbook of Condition Monitoring* 1st Ed 1996 621.816RAO Elsevier Science Ltd. The Boulevard, Langford Lane Kidlington Oxford, OX5 1GB UK

[35] SanDisk Corporation. *CompactFlash Memory Card Product Manual* SanDisk Corporation 140 Caspian Court Sunnyvale CA 94089 also available from www.sandisk.com

[36] SanDisk Corporation. *Interfacing SanDisk CompactFlash Cards to an 80C51 Microcontroller* SanDisk Corporation 140 Caspian Court Sunnyvale CA 94089 also available from www.sandisk.com

[37] SanDisk Corporation. *SanDisk CompactFlash & Motorola 8 Bit Microcontroller Interface Design Reference Example* SanDisk Corporation 140 Caspian Court Sunnyvale CA 94089 also available from www.sandisk.com

[38] Schmidt, Friedhelm. *The SCSI Bus and IDE Interface: Protocols, Applications and Programming* 1998 2nd Ed 004.62SCHM Addison Wesley

[39] Schrire, Stephen. *Notes to EEE359W, Components, Circuits and Modules* 1998 University of Cape Town, South Africa

[40] Stoffregen, Paul. *Using an IDE Hard Drive with an 8051 board and 82C55* 2000 available from www.pjrc.com/tech/8051/ide/

[41] Stremler F.G. *Introduction to Communication Systems* 3rd Ed 1990 621.382STRE Addison Wesley page542-543

[42] Taylor, James I. *The Vibration Analysis Handbook* 1994 621.811TAYL Vibration Consultants Incorporated 5733 South Dale Mabry Highway Tampa FL

University of Cape Town

[43] Technical committee T13 AT Attachment. *2008D AT Attatchment-3 Interface (ATA3)* Available from ANSI 11 West 42nd Street New York NY 10036 or www.ansi.org

[44] Texas Instruments. *Audio Band DSP CODEC offers Variable Sampling Rate to 48kHz* Texas Instruments tech innovations Volume 5 Available from www.ti.com/sc/techinnovations5

[45] Toshiba Semiconductors. *T6963C Dot Matrix LCD Control LSI* available from http://doc.semicon.toshiba.co.jp/noseek/us/td/03frame.htm

[46] Williams, J.H, Davies, A, Drake, P.R. *Condition Based Maintenance and Machine Diagnostics* 1994 621.816WILL Chapman & Hall

University of Cape Town

# Appendix A

# Assembly Program Files

## A.1 Ana_hdr.dsp

```
.MODULE/ABS=0           ADSP2181_Runtime_Header;

{Partially written by Samuel Ginsberg

 This file is the runtime header for the DSP chip. It contains the interrupt
 vector table, the declarations for a large number of variables, including
 many which are global to all the analyzer modules. In addition this file
 contains all the interrupt service routines except one which is used in the
 uart module.

 The code in this file is largely based on Analog Devices' examples.
 The code to process the input samples is mine, as is the code to handle
 keypad interrupts and the function to change the CODEC sampling rate.
 All of the functions have been modified to fit the application.

 The following functions are present:
 start: This is the reset vector of the DSP. It begins by setting up the i2
        and i3 registers for CODEC autobuffering. It then does some
        initialization on the Real Time Clock chip. The SPORT modules
        (synchronous serial ports) are then set up. The timer is set up and
        the DSP memory and busses are set up. The function init_codec is
        invoked to set up the CODEC. Finally this function hands control over
        to void main (), the main C module.
 sp0r_ctrl: This function is the interrupt handler for the serial port that
        connects to the CODEC. The function is called whenever data is
        received from the CODEC. This function provides all the triggering on
        the data and packs the data into arrays ready for the higher level
        functions to use.
 sp0x_ctrl: This function is used to send data to the CODEC in order to set
        the CODEC up. It is the interrupt handler for the serial port 0
        transmit complete interrupt. The function sends out all of the
        commands contained in the init_commands array. When all of the
        commands have been sent a flag is set to indicate completion.
 init_codec: This function is responsible for setting up the CODEC. It
        enables the relevant interrupts, sends all the commands to the CODEC
        and waits to ensure that all the commands are accepted and executed.
 int2_ctrl: This interrupt handler is called by interrupt 2, the interrupt
        that occurs when a top row key (softkey) on the keypad is pressed.
 change_fs: This changes the sample frequency of the CODEC. It sets up the
        configuration word to send to the CODEC and ensures that it gets
        sent. The function then flushes the data arrays to ensure correct
        results.
 pause_: A simple delay function.}

{=============================================================================}

#include <asm_sprt.h>;       {holds macros to set up the DSP system to run C}
                                                    {compiled code.}
```

University of Cape Town

```
{=========================================================================}

{The .external declarations tell the assembler that the code for these
 functions is contained in other modules and is declared to be global in
 those modules.}

.external    ___lib_setup_everything; {initializes the C runtime environment}
.external    main_;                                  {the C function void main ()}
.external    process_a_bit_;   {an interrupt handler used in the UART module}

{=========================================================================}

{The .global declarations tell the assembler to make these functions
 accessible to other modules}

.global change_fs_;
.global pause_;

{=========================================================================}

{Variable and Buffer Declarations}

.var/dm/ram/circ rx_buf_ [3];                          {AD1847 receive buffer}
.var/dm/ram/circ tx_buf_[3];                          {AD1847 transmit buffer}
.var/dm/ram/abs=0 xreal_ [4096];  {xreal and ximag are the arrays into which}
.var/dm/ram/abs=0x1000 ximag_ [4096];{the CODEC A/D places vibration samples}
.var/dm/ram/circ init_cmds[13];    {init commands holds a string of commands}
                                        {to send to the CODEC to initialize it}
.var/dm soft0_[11];   {the softX_ arrays contain bitmap graphics data. These}
.var/dm soft1_[8];   {bitmaps are displayed at the bottom of the screen, and}
.var/dm soft2_[8];       {tell the user what function a softkey will perform}
.var/dm soft3_[5];
.var/dm soft4_[3];
.var/dm soft5_[3];
.var/dm soft6_[7];
.var/dm soft7_[3];
.var/dm soft8_[3];
.var/dm soft9_[11];
.var/dm stat_flag;   {a status flag that tells the main CODEC initialization}
            {function when the SPORT interrupt handler has finished its job}
.var/dm temp_i1;                          {temp storage for i1 scratch register}
.var/dm temp_i1_2;                {temp storage for i1 scratch register. Used}
                                                        {when temp_i1 is in use}
.var/dm temp_ax0;                                 {temp storage for ax0 register}
.var/dm temp_ay0;                                 {temp storage for ay0 register}
.var/dm change_ax0;   {temp storage for the ax0 register. Used in change_fs_}
.var/dm init_ax1;     {temp storage for ax1. Used by the init_codec function}
.var/dm init_ax0;     {temp storage for ax0. Used by the init_codec function}
.var/dm init_ay0;     {temp storage for ay0. Used by the init_codec function}
.var/dm temp_l1;     {temporary store for l1, in this case the length of the}
                                                              {init_cmds array}
.var/dm temp_ar;    {temporary sorage for the ar register. Normally ar would}
                 {not be backed up, as it is a scratch register for C called}
                   {assembly functions, but it should be backed up when used}
                                                 {by assembly interrupt handlers}
.var/dm int2_ax0;  {temp store for ax0. Used by the keypad interrupt handler}
.var/dm int2_af;    {temp store for af. Used by the keypad interrupt handler}
.var/dm int2_ar;    {temp store for ar. Used by the keypad interrupt handler}
.var/dm int2_ay0;  {temp store for ay0. Used by the keypad interrupt handler}
.var/dm full_;  {a flag that is set when the xreal and ximag arrays are full}
.var/dm trigger_;      {a flag that is set if the trigger event has occurred}
.var/dm triglev_;                       {the voltage level at which to trigger}
.var/dm last_sample;     {records the last sample to determine if a positive}
                                             {edge has ocurred for triggering}
```

University of Cape Town

```
.var/dm datapos_r;        {the absolute memory address of the position in the}
                                  {xreal array to place the next CODEC sample}
.var/dm datapos_i;        {the absolute memory address of the position in the}
                                  {ximag array to place the next CODEC sample}
.var/dm sample_no;      {counts the data samples coming in to determine their}
                                                              {array position}
.var/dm state_;
.var/dm change_peaks_;    {a flag indicating that the peak finder was toggled}
.var/dm show_peaks_;    {a flag that is set if the peak finder function is on}
.var/dm old_state_;   {stores the old state of the analyzer, e.g. was there a}
                      {menu on screen, or was it in spectrum or waveform mode}

{The following variables are accessible to other modules.}

.global rx_buf_;
.global tx_buf_;
.global xreal_;
.global ximag_;
.global full_;
.global trigger_;
.global triglev_;
.global state_;
.global old_state_;
.global change_peaks_;
.global show_peaks_;
.global soft0_;
.global soft1_;
.global soft2_;
.global soft3_;
.global soft4_;
.global soft5_;
.global soft6_;
.global soft7_;
.global soft8_;
.global soft9_;

{=========================================================================}

{Constant Declarations. These are the memory mapped ADSP2181 control
 registers that are used here.}

.const PFCS= 0x3FE6;        {Setup for the port lines connected to the keypad}
.const PFD= 0x3FE5;                    {controls data lines for the keypad }
.const SPORT1_Autobuf= 0x3fef;     {Synchronous serial port 1 setup registers}
.const SPORT1_RFSDIV= 0x3ff0;
.const SPORT1_SCLKDIV= 0x3ff1;
.const SPORT1_Control_Reg= 0x3ff2;
.const SPORT0_Autobuf= 0x3ff3;     {Synchronous serial port 0 setup registers}
.const SPORT0_RFSDIV= 0x3ff4;
.const SPORT0_SCLKDIV= 0x3ff5;
.const SPORT0_Control_Reg= 0x3ff6;
.const SPORT0_TX_Channels0= 0x3ff7;
.const SPORT0_TX_Channels1= 0x3ff8;
.const SPORT0_RX_Channels0= 0x3ff9;
.const SPORT0_RX_Channels1= 0x3ffa;
.const TSCALE= 0x3ffb;                          {Timer module setup registers}
.const TCOUNT= 0x3ffc;
.const TPERIOD= 0x3ffd;
.const DM_Wait_Reg= 0x3ffe;           {Data memory wait state setup register}
.const System_Control_Reg= 0x3fff;    {Boot memory wait state setup register}

{=========================================================================}

{Here is the interrupt vector table. Each interrupt is spaced by four words.
 All the interrupt handlers are written in assembly code.}
```

University of Cape Town

```
__Reset_vector:          call ___lib_setup_everything;        {power on/reset}
                         call start;
                         nop;
                         nop;
__Interrupt2:            jump int2_ctrl;                    {keypad interrupt}
                         nop;
                         nop;
                         nop;
__InterruptL1:           rti;
                         nop;
                         nop;
                         nop;
__InterruptL0:           rti;
                         nop;
                         nop;
                         nop;
__Sport0_trans:          jump sp0x_ctrl;   {CODEC transmit complete interrupt}
                         nop;
                         nop;
                         nop;
__Sport0_recv:           jump sp0r_ctrl;             {CODEC receive interrupt}
                         nop;
                         nop;
                         nop;
__InterruptE:            rti;
                         nop;
                         nop;
                         nop;
__BDMA_interrupt:        rti;
                         nop;
                         nop;
                         nop;
__Interrupt1:            rti;
                         nop;
                         nop;
                         nop;
__Interrupt0:            rti;
                         nop;
                         nop;
                         nop;
__Timer_interrupt:       jump process_a_bit_;        {used by the UART module}
                         nop;
                         nop;
                         nop;
__Powerdown_interrupt:   rti;
                         nop;
                         nop;
                         nop;

{============================================================================}

{Variable and buffer initialization}

{Initialize the transmit buffer to send to the CODEC}

.init tx_buf_: 0xcc40, 0x0000,0x0000;


{Initialize the array of commands to send to the CODEC}

.init init_cmds:
0xc000,                                    {AD1847 Left input control reg}
0xc100,                                   {AD1847 Right input control reg}
0xc288,                                    {AD1847 left aux 1 control reg}
```

University of Cape Town

```
0xc388,                                          {AD1847 right aux 1 control reg}
0xc488,                                           {AD1847 left aux 2 control reg}
0xc588,                                          {AD1847 right aux 2 control reg}
0xc69F,                                             {AD1847 left DAC control reg}
0xc79F,                                           {AD1847 right DAC control reg}
0xc843,                                            {AD1847 data format register}
0xc909,                                  {AD1847 interface configuration reg}
0xca00,                                               {AD1847 pin control reg}
0xcc40,                                {AD1847 miscellaneous information reg}
0xcd00;                                     {AD1847 digital mix control reg}

{Initialize the bitmap arrays that display the legends above the softkeys}

.init soft0_: 0x000A,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000
              ,0x0000,0x0000,0x0000;
                              {soft0_ will clear a softkey display above a key}

.init soft1_: 0x0007,0x7D04,0x23E0,0x7EB1,0x03E8,0x105F,0x03C1,0x07C0;
                                                    {soft1_ will display "menu"}
.init soft2_: 0x0007,0x7E88,0x03F5,0x440F,0x51E0,0x7C8A,0x4419,0x5660;
                                                   {soft2_ will display "peaks"}
.init soft3_: 0x0004,0x66B3,0x03F5,0x4410,0x7E00;
                                                     {soft3_ will display "set"}
.init soft4_: 0x0002,0x1155,0x1080;
                                                      {soft4_ will display "<-"}
.init soft5_: 0x0002,0x0084,0x5544;
                                                      {soft5_ will display "->"}
.init soft6_: 0x0006,0x3A35,0x3820,0x7821,0x7811,0x7E20,0x43F0;
                                                    {soft6_ will display "Quit"}
.init soft7_: 0x0002,0x1117,0x2080;          {soft7_ will display an up arrow}
.init soft8_: 0x0002,0x105D,0x0880;        {soft8_ will display a down arrow}
.init soft9_: 0x000A,0x43F0,0x03F4,0x2C11,0x7E20,0x3A35,0x180E,
              0x46A6,0x03F5,0x441F,0x5160;  {soft9_ will display "Trigger"}

{============================================================================}

{******** ADSP 2181 intialization ********}
start:
i2 = ^rx_buf_;      {initialize an address pointer to start of receive buffer}
l2 = %rx_buf_;       {initialize a length register to size of receive buffer}
i3 = ^tx_buf_;    {initialize an address pointer to start of transmit buffer}
l3 = %tx_buf_;       {initialize a length register to size of transmit buffer}
i1 = ^init_cmds;     {initialize an address pointer to start of init commands}
l1 = %init_cmds;     {initialize a length register to number of init commands}
dm (temp_i1)=i1; {store the scratch register's values before C clobbers them}
dm (temp_l1)=l1;
i1 = ^xreal_;
dm (datapos_r)=i1;  {set up the datapos pointers to the start of data arrays}
i1= ^ximag_;
dm (datapos_i) =i1;

l1 = 0;                  {restore l1 to the value that the C compiler expects}

reset fl2;                                        {Real time clock set up part 1}
reset fl1;

{Serial Port 0 (SPORT0) Set Up}

save_reg;                                          {save registers on C stack}

{Serial port 0 connects to the CODEC. The port uses autobuffering. This means
 that when data comes into the port it is automatically placed in the DSP's
 memory. The data is placed in memory at a position pointed to by the i2
 register (RIREG). The length of the incoming data array is held by the DSP
 in the l2 register. When the data array is full, the SPORT generates an
```

University of Cape Town

```
 interrupt. A similar feature applies to the transmission system. Note that
 because of this the C compiler cannot use the i2 and i3 registers. This
 means that the -mreserved=i2,i3 compiler option is required for this to work
 correctly.}
ax0 = b#0000011010100111;
dm (SPORT0_Autobuf) = ax0;
ax0 = 0;
dm (SPORT0_RFSDIV) = ax0;  {These two registers define the rate at which the}
dm (SPORT0_SCLKDIV) = ax0;        {port runs. This option selects full speed}
{Set up the port for 16 bit words, enable muliple channel operation with 32
channels.}
ax0 = b#1000011000001111;
dm (SPORT0_Control_Reg) = ax0;
{Multi channel operation is in use, but we only wish to receive/transmit
 three words at a time to the CODEC, so we disable all except three of the
 channels on transmit and receive. See CODEC datasheet for details of time
 slot usage.}
ax0 = b#0000000000000111;
dm (SPORT0_TX_Channels0) = ax0;
ax0 = b#0000000000000111;
dm (SPORT0_TX_Channels1) = ax0;
ax0 = b#0000000000000111;
dm (SPORT0_RX_Channels0) = ax0;
ax0 = b#0000000000000111;
dm (SPORT0_RX_Channels1) = ax0;

{Serial Port 1 (SPORT1) setup. The SPORT1 is only used by the UART, and then
 only as a set of flag pins, so we disable it here.}
ax0=0;
dm(SPORT1_Autobuf)=ax0;                                  {autobuffering disabled}
dm(SPORT1_RFSDIV)=ax0;                                         {RFSDIV not used}
dm(SPORT1_SCLKDIV)=ax0;                                      {SCLKDIV not used}
dm(SPORT1_Control_Reg)=ax0;                            {ctrl functions disabled}

{Timer setup. The timer is only used by the UART and it is explicitly set up
 in that module. Thus here we simply disable it.}
ax0=0;
dm(TSCALE)=ax0;
dm(TCOUNT)=ax0;
dm(TPERIOD)=ax0;

{System and Memory setup}
ax0 = b#0000000000000000;              {Set up for no wait states on IO memory}
dm (DM_Wait_Reg) = ax0;
ax0 = b#0001000000000000;                                      {enable SPORT0}
dm (System_Control_Reg) = ax0;
ifc = b#00000011111111;                          {clear any pending interrupts}
nop;
icntl = b#00100;                 {edge sensitive on IRQ2, the keypad interrupt}

call init_codec;                            {call a routine to set up the CODEC}

restore_reg;                                    {restore C runtime registers}

CALL main_;                    {Hand control over to void main (), the C module}

{=========================================================================}

sp0r_ctrl:             {Called from Location 0014: SPORT0 rx interrupt handler}
{Stores the data from rx_buf_[1] into successive positions in xreal_ and
 ximag and when xreal_ and ximag are full (total 8192 samples) set the full_
 flag=1.
 The triggering routine is also here. We look for a zero crossing
 and positive slope before starting to pack the xreal_ and ximag arrays. When
 the trigger point is found, triggered_ is set to 1. When the array is
```

University of Cape Town

```
 full trigger_ is reset.}

        dm (temp_ar) = ar;     {store some registers so we don't clobber them}
      dm (temp_ax0) = ax0;
      dm (temp_ay0) = ay0;
      dm (temp_i1_2) = i1;

        ar=dm (full_);           {check to see if the main loop has finished}
        ar = pass ar;            {with the data. Main clears full_ when it's}
        if gt jump done_rec;              {finished with the last set of data}

        ar= dm (trigger_);                                 {triggering function}
      ar = pass ar;
        if gt jump check_full;          {if triggered carry on filling array}
        ay0=dm (triglev_);
        ax0 = dm (rx_buf_+1);     {else see if the trigger event has occurred}
        ar= ax0-ay0;           {triggering is done by seeing if the sample is}
        if le jump save_last;  {greater then the tigger level. If the sample}
        ax0= dm (last_sample);   {is big enough then it checks to see if the}
        ar = ax0-ay0;  {slope is positive by seeing is this sample is bigger}
        if gt jump save_last;                             {than the last sample}
      ax0=1;
      dm (trigger_) = ax0;

check_full:
        ax0=dm (datapos_r);        {check to see if the xreal_ array is full}
        ay0=4096;                         {if it is then set the full flag.}
      ar=ax0-ay0;
      ay0=ar;
      ax0=^xreal_;
      ar=ax0-ay0;
      if eq jump buffer_full;
        ax0=dm (datapos_i);        {check to see if the ximag_ array is full}
        ay0=4096;                         {if it is then set the full flag.}
      ar=ax0-ay0;
      ay0=ar;
      ax0=^ximag_;
      ar=ax0-ay0;
      if eq jump buffer_full;

        ax0 = dm (sample_no);                          {if sample_no is zero}
        ar = pass ax0;                  {put data into xreal_ or else put}
        if ne jump pack_ximag;                            {data into ximag}

        ax0=0xFFFF;                  {if sample_no was zero make it non-zero}
      dm (sample_no)=ax0;

        ax0 = dm (rx_buf_+1);           {put the next value in the xreal_ at}
        i1 = dm (datapos_r);                          {position datapos_r}
        dm (i1,m1) = ax0;                {automatically increment datapos}
      dm (datapos_r) = i1;
      jump save_last;
pack_ximag:
        ax0=0;                       {if sample_no was non-zero make it zero}
      dm(sample_no)=ax0;

        ax0=dm (rx_buf_+1);                {put the next value in ximag_ at}
        i1= dm (datapos_i);                           {position datapos_i}
      dm (i1,m1)= ax0;
      dm (datapos_i) = i1;

save_last:
      dm (last_sample)=ax0; {store the last sample for use in next trigger}
      jump done_rec;
buffer_full:
```

University of Cape Town

```
        i1= ^xreal_;        {reset the datapos_r pointer to the start of xreal}
      dm (datapos_r) = i1;
        i1=^ximag_;         {reset the datapos_i pointer to the start of ximag}
      dm (datapos_i) =i1;
      ax0=1;
        dm (full_) = ax0;       {set full_ to 1 to indicate a full data array}
      ax0=0;
        dm (trigger_) =ax0;                             {wait to be retriggered}
        dm (sample_no)=ax0;      {start packing into first element of xreal_}
done_rec:
        i1 = dm (temp_i1_2);                {restore the clobbered registers}
      ax0 = dm (temp_ax0);
      ay0 = dm (temp_ay0);
      ar = dm (temp_ar);
      l1=0;
  rti;

{=============================================================================}

{Transmit interrupt used for CODEC initialization.
 Called from location 0010 SPORT0 tx
 This interrupt is called when the previous contents of register tx0 have
 been sent out of SPORT0.}
sp0x_ctrl:
        dm (temp_ar) = ar;          {Back up registers which will be clobbered}
        dm (temp_ax0) = ax0;
        dm (temp_ay0) = ay0;
        dm (temp_i1_2) = i1;

        ar = dm(stat_flag);    {check the status flag to see if there is data}
        ar = pass ar;             {to be transmitted. If not skip to the end}
        if eq jump done;

        i1=dm (temp_i1);       {pull the values of i1 and l1 from memory, the}
        l1=dm (temp_l1); {values of i1 and l1 were placed there at boot time}
        ax0 = dm (i1, m1);                          {fetch next control word and}
        dm (tx_buf_) = ax0;                         {place it in transmit slot 0}
        dm (temp_i1) = i1;                           {retain the new value of i1}
        ax0 = i1;
        ay0 = ^init_cmds; {check if we've transmitted all the data from the}
        ar = ax0 - ay0;                                {circular init_cmds array}
        if gt jump done;  {if not simply return from the interrupt and wait}
        ax0 = 0x8000;     {else set done flag and remove the MCE bit from the}
        dm (tx_buf_) = ax0;                                    {CODEC commands}
        ax0 = 0;
        dm (stat_flag) = ax0;      {reset status flag to indicate completion}

done:
        l1=0;                                       {restore clobbered registers}
        i1 = dm (temp_i1_2);
        ax0 = dm (temp_ax0);
        ay0 = dm (temp_ay0);
        ar = dm (temp_ar);
        rti;

{=============================================================================}

int2_ctrl:
{interrupt 2 is the keyboard interrupt. This interrupt is triggered by some
 high priority keypad buttons, those which are used as softkeys.}

        dm (int2_ax0) =ax0;      {save registers which get clobbered. This is}
        dm (int2_ar) =ar;           {an interrupt handler so back up scratch}
        dm (int2_ay0) =ay0;                                   {registers too}
      ar=pass af;
```

University of Cape Town

```
        dm(int2_af)=ar;

          ax0=0xFFFF;     {This delay is for debouncing on some types of switch}
        af=pass ax0;
        do delay_loop until eq;
        af=af-1;
delay_loop:
          ax0=0x000F;       {set the switch rows as outputs, columns as inputs}
        dm (PFCS)=ax0;
          ax0=0x0001;      {pull up the row which drives the interrupt buttons}
        dm (PFD)=ax0;
          ax0=dm(PFD);                                              {read pins in}

        ar = pass ax0;
          ar=tstbit 4 of ar;                {test the 'menu' or 'quit' button}
        if eq jump end_int2_menu;
          ax0=dm (state_);     {if state is set to 2 then goto end so we don't}
          ay0=2;                             {clobber the old_state_ variable}
        ar=ax0-ay0;
        if eq jump end_int2;

          ax0 =dm (state_);          {if the menu button was pressed, save the}
          dm (old_state_) =ax0;                {old state value to old_state_}
          ax0 = 2;             {and set state_ to 2 to signal to main_ that the}
          dm (state_)=ax0;                                  {menu should be shown}
end_int2_menu:

          ax0=dm(PFD);                                              {read pins in}

          ar = pass ax0;         {determine if the find peaks mode was toggled}
        ar=tstbit 5 of ar;
        if eq jump end_int2;
        ax0=dm(show_peaks_);
        ar=pass ax0;
        ar=tstbit 0 of ar;
        if eq jump setpeaks;
          ax0=0;                          {if the peak finder was on turn it off}
        jump delay;
setpeaks:
          ax0=1;                          {if the peak finder was off turn it on}
delay:  dm(show_peaks_)=ax0;
        ax0=1;
          dm (change_peaks_)=ax0;         {raise a flag to indicate the toggle}
        ax0=0xFFFF;
        af=pass ax0;
        do end_delay until eq;
        nop;
        nop;
        nop;
        nop;
        af=af-1;
end_delay:
end_int2:
          ax0=dm(int2_af);                          {restore clobbered registers}
        af=pass ax0;
        ay0=dm (int2_ay0);
        ar =dm(int2_ar);
        ax0=dm(int2_ax0);

          rti;

{=============================================================================}

init_codec:
```

University of Cape Town

```
{AD1847 Codec intialization}

        dm(init_ax1)=ax1;                          {backup clobbered registers}
        dm(init_ax0)=ax0;
        dm(init_ay0)=ay0;

        ax0 = 1;
        dm(stat_flag) = ax0;  {clear status flag to indicate that data is to}
                                                                      {be sent}
        imask = b#1001000000;   {enable transmit interrupt. Also modified by}
                                                                    {init_uart}
        ax0 = dm (i3, m1); {start interrupt by loading the transmit register}
        tx0 = ax0;
check_init:
        ax0 = dm (stat_flag);                          {wait for entire init}
        af = pass ax0;                                 {buffer to be sent to}
        if ne jump check_init;                                  {the CODEC}
        ay0 = 2;
check_acih:
        ax0 = dm (rx_buf_);                           {once initialized wait}
        ar = ax0 and ay0;                                  {for codec to go}
        if eq jump check_acih;                         {into autocalibration}
check_acil:
        ax0 = dm (rx_buf_);          {once autocalibration has started, wait}
        ar = ax0 and ay0;                             {for codec to come out}
        if ne jump check_acil;                           {of autocalibration}
        idle;
        ax0 = 0xc901;                          {clear autocalibration request}
        dm (tx_buf_) = ax0;
        idle;
        ax1 = 0x8000;                {control word to clear over-range flags}
        dm (tx_buf_) = ax1;
        ifc = b#00000011111111;                 {clear any pending interrupt}
        nop;

        imask = b#1000110001;        {enable rx0 interrupt and IRQ2 and IRQE}
        ay0=dm(init_ay0);
        ax0=dm(init_ax0);
        ax1=dm(init_ax1);

        rts;

{=========================================================================}
change_fs_:
{This routine changes the sampling rate of the CODEC. The command to send to
 the CODEC comes in from a C function via the ar register.}

        dm(change_ax0)=ax0;                                        {save ax0}
        dm(tx_buf_)=ar;

check_init_bit:   {Wait for the CODEC init bit to go low indicating that the}
ay0=1;
        ax0=dm (rx_buf_);              {CODEC is ready to change sample rate}
        ar=ax0 and ay0;
        if eq jump check_init_bit;

        ax0=0x8000;     {clear the MCE (mode change enable) bit in the CODEC}
        dm(tx_buf_)=ax0;

        ax0 = ^xreal_;
        dm (datapos_r)=ax0;                          {restart the data capture}
        ax0= ^ximag_;                {because previous samples were taken at}
        dm (datapos_i) =ax0;                          {a different sample rate}
```

University of Cape Town

```
        ax0=dm(change_ax0);                                    {restore ax0}
        rts;

{=============================================================================}
pause_:
{A simple pause function that works by letting the counter module count down
 to zero. The calling function places the initial value in the counter.}
        do end_pause until eq;
        ar=ar-1;
  end_pause:
        rts;

{=============================================================================}

.ENDMOD;
```

University of Cape Town

# A.2 Ana_io.dsp

```
.MODULE           ana_io;

{Written by: Samuel Ginsberg

 This file contains the keypad drivers for the vibration analyzer.
 There is one function here. This is called buttons_ and it scans
 the keypad and returns a number that corresponds to the button pressed.
 The original keypad used for the early versions of the analyzer had a 16
 line connection to the DSP bus and it returned 2^X where X was the button
 pressed. For this reason the function returns the following codes:

 Upper row: (left to right)   1    2    4    8
 Second row:                  16   32   64   128

 and so on.

 The return value is passed out in the ar register for C compatibility.}

.global buttons_;

.const PFCS= 0x3FE6;
.const PFD=  0x3FE5;

{Variable  Declarations}
.var/dm buttons_ax0;
.var/dm buttons_af;
.var/dm buttons_temp1;

{============================================================}
buttons_:
        imask = b#0000110001;{enable rx0 interrupt and IRQE and disable IRQ2}
        dm (buttons_ax0)=ax0;                                          {save ax0}
        ax0=0x000F;      {set up row lines as output, column lines as inputs}
        dm (PFCS)=ax0;

        ax0=0x0001;                                              {pull row 1 high}
        dm (PFD)=ax0;

        af=pass 0x0011;                      {delay while keypad cct settles.}
        do end_delay1 until eq;
        af=af-1;
end_delay1:

        ax0=1;                                 {set output to 1 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                             {read pins in}
        ar=tstbit 4 of ax0;                                      {test column1}
        if ne jump done_scan;

        ax0=2;                                 {set output to 2 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                             {read pins in}
        ar=tstbit 5 of ax0;                                      {test column2}
        if ne jump done_scan;

        ax0=4;                                 {set output to 4 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                             {read pins in}
        ar=tstbit 6 of ax0;                                      {test column3}
        if ne jump done_scan;
```

University of Cape Town

```
        ax0=8;                                  {set output to 8 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                          {read pins in}
        ar=tstbit 7 of ax0;                                   {test column4}
        if ne jump done_scan;

        ax0=0x0002;                                        {pull row 2 high}
        dm (PFD)=ax0;

        af= pass 0x0011;                {delay while keypad cct settles.}
        do end_delay2 until eq;
        af=af-1;
end_delay2:

        ax0=16;                                 {set output to 16 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                          {read pins in}
        ar=tstbit 4 of ax0;                                   {test column1}
        if ne jump done_scan;

        ax0=32;                                 {set output to 32 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                          {read pins in}
        ar=tstbit 5 of ax0;                                   {test column2}
        if ne jump done_scan;

        ax0=64;                                 {set output to 64 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                          {read pins in}
        ar=tstbit 6 of ax0;                                   {test column3}
        if ne jump done_scan;

        ax0=128;                               {set output to 128 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                          {read pins in}
        ar=tstbit 7 of ax0;                                   {test column4}
        if ne jump done_scan;

        ax0=0x0004;                                        {pull row 3 high}
        dm (PFD)=ax0;

        af= pass 0x0011;                {delay while keypad cct settles.}
        do end_delay3 until eq;
        af=af-1;
end_delay3:

        ax0=256;                                {set output to 256 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                          {read pins in}
        ar=tstbit 4 of ax0;                                   {test column1}
        if ne jump done_scan;

        ax0=512;                                {set output to 512 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                          {read pins in}
        ar=tstbit 5 of ax0;                                   {test column2}
        if ne jump done_scan;

        ax0=1024;                              {set output to 1024 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                          {read pins in}
        ar=tstbit 6 of ax0;                                   {test column3}
        if ne jump done_scan;

        ax0=2048;                              {set output to 2048 provisonally}
```

University of Cape Town

```
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                          {read pins in}
        ar=tstbit 7 of ax0;                                   {test column4}
        if ne jump done_scan;

        ax0=0x0008;                                        {pull row 4 high}
        dm (PFD)=ax0;

        af= pass 0x0011;                    {delay while keypad cct settles.}
        do end_delay4 until eq;
        af=af-1;
        end_delay4:

        ax0=4096;                           {set output to 4096 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                          {read pins in}
        ar=tstbit 4 of ax0;                                   {test column1}
        if ne jump done_scan;

        ax0=8192;                           {set output to 8192 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                          {read pins in}
        ar=tstbit 5 of ax0;                                   {test column2}
        if ne jump done_scan;

        ax0=16384;                         {set output to 16384 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                          {read pins in}
        ar=tstbit 6 of ax0;                                   {test column3}
        if ne jump done_scan;

        ax0=32768;                         {set output to 32768 provisonally}
        dm (buttons_temp1)= ax0;
        ax0=dm(PFD);                                          {read pins in}
        ar=tstbit 7 of ax0;                                   {test column4}
        if ne jump done_scan;

        ax0=0;                            {if no button pressed return zero}
        dm (buttons_temp1)=ax0;

done_scan:
        ax0=0x0001;                {pull row 1 high so interrupts can occur}
        dm (PFD)=ax0;
        ax0=dm(buttons_temp1);
        ar=pass ax0;
        ax0= dm(buttons_ax0);                                    {restore ax0}
        imask = b#1000110001;       {enable rx0 interrupt and IRQ2 and IRQE}
rts;

{============================================================}

.ENDMOD;
```

University of Cape Town

# A.3 Ana_clk.dsp

```
.MODULE          ana_clk;

{Written by: Samuel Ginsberg

 This module contains the functions to access the real time clock chip, the
 Dallas DS1302.
 There are three functions in this module. They are get_clk_, put_clk_ and
 delay_1_micro_sec.
 get_clk_ retrieves data from the clock. The command (which includes the
 address information) to send to the clock chip is put into dm(clk_com_),
 the get_clk_ function is invoked and the result is returned in dm(clk_dat_).

 put_clk_ sends data to the clock. The command (which includes the address
 information) to send to the clock is put into dm(clk_com_), the data to send
 to the clock is put into dm(clk_dat_) and the function is called.

 delay_1_micro_sec is simply a 1 microsecond delay used by get_clk_ and
 put_clk_.

 The clock commands can be found in the datasheet for the Dallas DS1302 RTC
 chip.}

.const PFCS= 0x3FE6;
.const PFD= 0x3FE5;

.global get_clk_;
.global put_clk_;

.var/dm clk_com_;
.var/dm clk_dat_;
.global clk_com_;
.global clk_dat_;

.var/dm clk_ax0;
.var/dm clk_tmp;

{=============================================================================}

get_clk_:
{This function gets a byte from the real time clock chip, DS1302.
 The command to send to the RTC chip comes in from the variable dm(clk_com_)
 and the data from the chip is put into the variable dm(clk_dat_)}

     imask=b#0001110001;                          {disable keypad interrupts}
     dm (clk_ax0)=ax0;                           {save non scratch registers}
     si=dm(clk_com_);                    {load command into shifter input}
     sr= lshift si by 0 (lo);                    {copy command byte into sr0}
     reset fl0;                                   {make clock line start low}
     call delay_1_micro_sec;
     ax0=0x0001;
     dm(PFCS)=ax0;                                        {make IO line output}
     call delay_1_micro_sec;
     set fl2;                                               {pull !reset high}
     call delay_1_micro_sec;
     ax0=0x0008;                                    {8 bits in command byte}
     af=pass ax0;
     do sendloop1 until eq;
        {rotate the command byte right and set/clear the IO line accordingly}
     ar=tstbit 0 of sr0;
     if gt jump set1;
     ax0=0x0000;
```

University of Cape Town

```
     jump clr1;
set1: ax0=0x0001;
clr1: dm (PFD)=ax0;
     sr=lshift si by -1(lo);
     si=sr0;
     set fl0;       {set clock pulse by setting then clearing the clock line}
     call delay_1_micro_sec;
     reset fl0;
     call delay_1_micro_sec;
     af=af-1;
sendloop1:
     ax0=0x0000;
     dm(PFCS)=ax0;                                    {set up IO line as input}
     dm (clk_dat_)=ax0;
     call delay_1_micro_sec;
     si=0x0000;
     sr0=si;
     ax0=0x0008;                                        {8 bits in data byte}
     af=pass ax0;
     do getloop1 until eq;
     sr=lshift sr0 by -1 (lo);                        {rotate clk_dat_ right}
     ax0=dm(PFD);                {read IO pin in and set up LSB of clk_dat_}
     ar=tstbit 0 of ax0;
     if eq jump clr2;
     ar=setbit 7 of sr0;
     sr0=ar;
clr2:
     set fl0;              {send clock pulse by setting and clearing clk pin}
     call delay_1_micro_sec;
     reset fl0;
     af=af-1;
getloop1:
        dm (clk_dat_)=sr0;
        reset fl2;                                      {pull reset pin low}
        ax0=0x0001;
        dm(PFD)=ax0;     {set up pfcs and pfd so keypad interrupts can occur}
        ax0=0x000F;
        dm(PFCS)=ax0;
        cntr=0x3FFF;         {a long delay loop to allow clock settling time}
        imask=b#1001110001;                      {restore interrupt control}
        ax0=dm(clk_ax0);
      rts;

{=========================================================================}

put_clk_:

{This function puts a byte into the real time clock chip, DS1302.
 The command to send to the RTC chip comes in from the variable dm(clk_com_)
 and the data to go the chip is put into the variable dm(clk_dat_)}
     imask=b#0001110001;                          {disable keypad interrupts}
     dm (clk_ax0)=ax0;                           {save non scratch registers}
     ax0=1;
     dm(clk_tmp)=ax0;
load:
     si=dm(clk_com_);                      {load command into shifter input}
     sr=lshift si by 0 (lo);                    {copy command byte into sr0}
     reset fl0;                                  {make clock line start low}
     call delay_1_micro_sec;
     ax0=0x0001;
     dm(PFCS)=ax0;                                    {make IO line output}
     call delay_1_micro_sec;
     set fl2;                                          {pull !reset high}
     call delay_1_micro_sec;
     ax0=0x0008;                                {8 bits in command byte}
```

University of Cape Town

```
     af=pass ax0;
     do sendloop2 until eq;
        {rotate the command byte right and set/clear the IO line accordingly}
     ar=tstbit 0 of sr0;
     if gt jump set2;
     ax0=0x0000;
     jump clr3;
set2: ax0=0x0001;
clr3: dm (PFD)=ax0;
     sr=lshift si by -1(lo);
     si=sr0;
     set fl0;       {set clock pulse by setting then clearing the clock line}
     call delay_1_micro_sec;
     reset fl0;
     call delay_1_micro_sec;
     af=af-1;
sendloop2:
      ax0=dm(clk_tmp);
      af=pass ax0;
      if eq jump done_put;
      ax0=dm(clk_dat_);
      dm(clk_com_)=ax0;
      ax0=0;
      dm (clk_tmp)=ax0;
      jump load;
done_put:
        call delay_1_micro_sec;
        reset fl2;                                    {pull reset pin low}
        ax0=0x0001;
        dm(PFD)=ax0;     {set up pfd and pfcs so keypad interrupts can occur}
        ax0=0x000F;
        dm(PFCS)=ax0;
        imask=b#1001110001;                    {restore interrupt control}
      ax0=dm(clk_ax0);
      rts;

{=============================================================================}

delay_1_micro_sec:                                    {no comment needed}
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
nop;
```

University of Cape Town

```
nop;
nop;
nop;
nop;
nop;
nop;
rts;

{=============================================================================}

.ENDMOD;
```

University of Cape Town

# A.4 Ana_uart.dsp

```
{*************************************************************************

This file was taken from an Analog devices application note. The software
contained herein is not the work of Samuel Ginsberg.
Adaptations have been made to remove the automatic baud rate detection.

   ADSP-2101 Family Software UART                                UART.DSP

    This uses FLAG_IN, FLAG_OUT and the TIMER of ADSP-2101 to interface to
    an RS-232 asynchronous serial device such as a VT100 terminal.  ex:

      ADSP-2101 FLAG_OUT ----------> AD233 ----------> RS-232 RX

      ADSP-2101  FLAG_IN <---------- AD233 <---------- RS-232 TX
                   (TIMER maintains baudrate)

    Parameters bits/word, baudrate, stopbits & parity are user-programmable.

    An RS-232 line driver chip (such as the AD233) can be used to
    electrically interface +5 VDC to the RS-232 line voltage levels.

    The operation of the transmitter setup routine is completely
    independent on the the receiver setup routine operation.  Although
    both tx and rx use the same timer as a master clock source, the
    xmitted bits need not be in sync with the received bits. The default
    state of the reciever is OFF, so the "turn_rx_on" subroutine must be
    used to enable RX.

Calling Argument:
          for autobaud load the baud constant
        dm(baud_period_)=(Proc_frequency/(3*Baudrate))-1

Useful Subroutines:
        init_uart_       Must be called after system reset.
        get_char_ar_    Waits for RX input and returns with it in ax1.
        out_char_ar_    Waits for last TX output and transmits data from ax1.
        turn_rx_on_      Must be called to enable the receipt of RX data.
        turn_rx_off_     Can be used to ignore input RX data.

Useful Flag:
      DM(flag_rx_ready)   If this DM location is all ones it indicates that
                     the UART is ready to rx new word. If it is zero
                     then data is being received. Can be used for xon
                     xoff flow control.

    Author:   Fares Eidi, 21-May-90, Analog Devices Inc.
    modified: Christoph D. Cavigioli, 17-Dec-90
    modified: Steven Cox, 20-Dec-91, extensive rewrite
    modified: Steven Cox, 31-Dec-91, Reset flag_rx_stop_yet to 1, Comments.
    modified: Steven Cox, 11-Feb-92, Added support for autobaud.
    modified: Philip Holdgate, 02-Apr-92, modified for Apps Note, cleaned up.

*************************************************************************}

.module             ana_UART;

{______The Constants below must be changed to modify uart parameters _______}

.const  tx_num_of_bits = 10;          { start bits + tx data bits + stop bits }
.const  rx_num_of_bits = 8;      { rx data bits (start&stop bits not counted)}
.const  RX_BIT_ADD = 0x0100;            { = 1 shifted left by rx_num_of_bits }
```

University of Cape Town

```
.const  TX_BIT_ADD = 0xfe00;    { = 0xffff shifted left by (tx data bits+1) }

{_____________Definitions of memory mapped control registers_____________}

.const  TSCALE=                 0x3ffb;
.const  TCOUNT=                 0x3ffc;
.const  TPERIOD=                0x3ffd;
.const  System_Control_Reg=     0x3fff;

{_____________________________________________________________________________}

.global init_uart_;                          { UART initialize baudrate etc. }
.global out_char_ar_;                          { UART output a character }
.global get_char_ar_;                    { UART wait & get input character }
.global turn_rx_on_;                        { UART enable the rx section }
.global turn_rx_off_;                      { UART disable the rx section }
.entry process_a_bit_;          { UART timer interrupt routine for RX and TX }

.global flag_rx_ready;
.global baud_period_;

.var    flag_tx_ready;       { flag indicating UART is ready for new tx word }
.var    flag_rx_ready;        { flag indicating UART is ready to rx new word }
.var    flag_rx_stop_yet;     { flag tells that a rx stop bit is not pending }
.var    flag_rx_no_word;    { indicates a word is not in the user_rx_buffer }
.var    flag_rx_off;          { indicates a that the receiver is turned off }
.var    timer_tx_ctr;      { divide by 3 ctr, timer is running @ 3x baudrate }
.var    timer_rx_ctr;      { divide by 3 ctr, timer is running @ 3x baudrate }
.var    user_tx_buffer;       { UART tx reg loaded by user before UART xmit }
.var    user_rx_buffer;       { UART rx reg read by user after word is rcvd }
.var    internal_tx_buffer; {formatted for serial word, adds start&stop bits}
                           { 'user_tx_buffer' is copied here before xmission }
.var    internal_rx_buffer;
.var    bits_left_in_tx; { num of bits left in tx buffer (not yet clkd out) }
.var    bits_left_in_rx; { number of bits left to be rcvd (not yet clkd in) }
.external baud_period_;
.var    baud_period_;                          { loaded by autobaud routine }
.var/dm/ram temp_ax0;
.var/dm/ram temp_ax1;
.var/dm/ram int_temp_ax0;
.var/dm/ram int_temp_ar;
.var/dm/ram int_temp_ay0;
.var/dm/ram int_temp_sr0;
.var/dm/ram int_temp_sr1;

{_____________________Initializing subroutine_____________________________}

init_uart_:
    dm (temp_ax0)=ax0;                                    {save ax0 register}
    ax0=0;
    dm(TSCALE)=ax0;              { decrement TCOUNT every instruction cycle }

    ax0=dm(baud_period_);                    { from autobaud or use constant}

    dm(TCOUNT)=ax0;
    dm(TPERIOD)=ax0;                  { interrupts generated at 3x baudrate }
    {ax0=0;
    dm(System_Control_Reg)=ax0;} { no bmwait, pmwait states, SPORT1 = FI/FO }

    ax0=1;
    dm(flag_tx_ready)=ax0;     { set the flags showing that UART is not busy }
    dm(flag_rx_ready)=ax0;
    dm(flag_rx_stop_yet)=ax0;
    dm(flag_rx_no_word)=ax0;
    dm(flag_rx_off)=ax0;                                  { rx section off }
```

University of Calgary

```
    set flag_out;                    { UART tx output is initialized to high }
    ifc=0x00ff;                      { clear all pending interrupts ADSP2181}
    nop;                                          { wait for ifc latency }
    imask=b#1001110001;                { enable TIMER interrupt handling }
                                              { also allow A/D interrupt}
    ena timer;                                       { start timer now }
    ax0=dm(temp_ax0);                              {restore ax0 register}
    rts;

{___________________Process_a_bit (TIMER interrupt routine)_____________________

    This routine is the heart of the UART.  It is called every timer
    interrupt (i.e. 3x baudrate).  This routine will xmit one bit at a
    time by setting/clearing the FLAG_OUT pin of the ADSP-2101. This routine
    will then test if the uart is already receiving. If not it will test
    flagin (rx) for a start bit and place the uart in receive mode if true.
    If already in receive mode it will shift in one bit at a time by reading
    the FLAG_IN pin.  Since the internal timer is running at 3x baudrate,
    bits need only be transmitted/received once every 3 timer interrupts.
_____________________________________________________________________________}

process_a_bit_:
    dm (int_temp_ax0)=ax0;
    dm (int_temp_ar)=ar;
    dm (int_temp_ay0)=ay0;
    dm (int_temp_sr0)=sr0;
    dm (int_temp_sr1)=sr1;
    ax0=dm(flag_tx_ready);    { if not in "transmit", go right to "receive" }
    ar=pass ax0;
    if ne jump receiver;

            {_________Transmitter Section_________}

    ay0=dm(timer_tx_ctr);              { test timer ctr to see if a bit }
    ar=ay0-1;                          { is to be sent this time around }
    dm(timer_tx_ctr)=ar;                     { if no bit is to be sent }
    if ne jump receiver;                { then decrement ctr and return }

    sr1=dm(internal_tx_buffer);    { shift out LSB of internal_tx_buffer }
    sr=lshift sr1 by -1 (hi);     { into SR1.  Test the sign of this bit }
    dm(internal_tx_buffer)=sr1;      { set or reset FLAG_OUT accordingly }
    ar=pass sr0;                       { this effectively clocks out the }
    if ge reset flag_out;         { word being xmitted one bit at a time }
    if lt set flag_out;                       { LSB out first at FLAG_OUT. }

    ay0=3;                           { reset timer ctr to 3, i.e. next bit }
    dm(timer_tx_ctr)=ay0;           { will be sent after 3 timer interrupts }

    ay0=dm(bits_left_in_tx);           { number of bits left to be xmitted }
    ar=ay0-1;                                 { is now decremented by one, }
    dm(bits_left_in_tx)=ar;          { indicating that one is now xmitted }
    if gt jump receiver;               { if no more bits left, then ready }

    ax0=1;                                { flag is set to true indicating }
    dm(flag_tx_ready)=ax0;                { a new word can now be xmitted }

            {_________Receiver Section_________}
receiver:
    ax0=dm(flag_rx_off);                  { Test if receiver is turned on }
    ar=pass ax0;
    if ne jump end_inter;

    ax0=dm(flag_rx_stop_yet);        { Test if finished with stop bit of }
    ar=pass ax0;                    { last word or not. if finished then }
```

University of Cape Town

```
    if ne jump rx_test_busy;              { continue with check for receive. }

    ay0=dm(timer_rx_ctr);               { decrement timer ctr and test to see }
    ar=ay0-1;                           { if stop bit period has been reached }
    dm(timer_rx_ctr)=ar;                                { if not return and wait }
    if ne jump end_inter;

    ax0=1;                              { if stop bit is reached then reset }
    dm(flag_rx_stop_yet)=ax0;                           { to wait for next word }
    dm(flag_rx_ready)=ax0;

    ax0=dm(internal_rx_buffer);                       { copy internal rx buffer }
    dm(user_rx_buffer)=ax0;                             { to the user_rx_buffer }

    ax0=0;                              { indicated that a word is ready in }
    dm(flag_rx_no_word)=ax0;                            { the user_rx_buffer }
    jump end_inter;

rx_test_busy:
    ax0=dm(flag_rx_ready);                { test rx flag, if rcvr is not busy }
    ar=pass ax0;                        { receiving bits then test for start.If it}
    if eq jump rx_busy;                  { is busy, then clk in one bit at a time }

    if flag_in jump rx_exit;           { Test for start bit and return if none }

    ax0=0;
    dm(flag_rx_ready)=ax0;              { otherwise, indicate rcvr is now busy }
    dm(internal_rx_buffer)=ax0;                         { clear out rcv register}

    ax0=4;                                { timer runs @ 3x baud rate, so rcvr }
    dm(timer_rx_ctr)=ax0;               { will only rcv on every 3rd interrupt }
                            { initially this ctr is set to 4.  This }
                                        { will skip the start bit and will }
                                  { allow us to check FLAG_IN at the center }
                                                  { of the received data bit }
    ax0=rx_num_of_bits;
    dm(bits_left_in_rx)=ax0;
rx_exit:
    jump end_inter;

rx_busy:
    ay0=dm(timer_rx_ctr);               { decrement timer ctr and test to see }
    ar=ay0-1;                         { if bit is to be rcvd this time around }
    dm(timer_rx_ctr)=ar;                  { if not return, else receive a bit }
    if ne jump end_inter;

rcv:                                                          { Shift in rx bit }
    ax0=3;                              { reset the timer ctr to 3 indicating }
    dm(timer_rx_ctr)=ax0;              { next bit is 3 timer interrupts later }
    ay0=RX_BIT_ADD;
    ar=dm(internal_rx_buffer);
    if not flag_in jump pad_zero;                   { Test RX input bit and }
    ar=ar+ay0;                                          { add in a 1 if hi }

pad_zero:
    sr=lshift ar by -1 (lo);              { Shift down to ready for next bit }
    dm(internal_rx_buffer)=sr0;

    ay0=dm(bits_left_in_rx);          { if there are more bits left to be rcvd }
    ar=ay0-1;                                     { then keep UART in rcv mode }
    dm(bits_left_in_rx)=ar;                                     { and return }
    if gt jump end_inter;                { if there are no more bits then.. }

                                                    { That was the last bit }
    ax0=3;                              { set timer to wait for middle of the }
```

University of Cape Town

```
    dm(timer_rx_ctr)=ax0;                                    { stop bit }
    ax0=0;                                  { flag indicated that uart is waiting }
    dm(flag_rx_stop_yet)=ax0;                    { for the stop bit to arrive }
end_inter:
    ax0=dm (int_temp_ax0);
    ar=dm (int_temp_ar);
    ay0=dm (int_temp_ay0);
    sr0=dm (int_temp_sr0);
    sr1=dm (int_temp_sr1);
    rti;

{____________________invoke_UART_transmit subroutine____________________

    This is the first step in the transmit process.  The user has now
    loaded 'user_tx_buffer' with the ascii code and has also invoked
    this routine.
_____________________________________________________________________________}
invoke_UART_transmit:
    ax0=3;                                  { initialize the timer decimator ctr }
    dm(timer_tx_ctr)=ax0;                   { this divide by three ctr is needed }
                                               { since timer runs @ 3x baud rate }

    ax0=tx_num_of_bits;                        { this constant is defined by the }
    dm(bits_left_in_tx)=ax0;                { user and represents total number of }
                                                  { bits including stop and parity }
                                             { ctr is initialized here indicating }
                                             { none of the bits have been xmitted }
    sr1=0;
    sr0=TX_BIT_ADD;                       { upper bits are hi to end txmit with hi }
    ar=dm(user_tx_buffer);                     { transmit register is copied into  }
    sr=sr or lshift ar by 1 (lo);          { the internal tx reg & left justified }
    dm(internal_tx_buffer)=sr0;                          { before it gets xmitted }

    ax0=0;                                    { indicate that the UART is busy }
    dm(flag_tx_ready)=ax0;
    rts;

{_____________________get an input character_____________________________
output:         ar
modifies:       ax0,ax1
_____________________________________________________________________________}

get_char_ar_:
        dm (temp_ax0)=ax0;
        dm (temp_ax1)=ax1;
wait2:  ax0=dm(flag_rx_no_word);
      ar=pass ax0;
        if ne jump wait2;                    { if no rx word input, then wait }

        ax1=dm(user_rx_buffer);                { get received ascii character }
      ax0=1;
        dm(flag_rx_no_word)=ax0;                                { word was read }
        ar=pass ax1;
        ax1=dm(temp_ax1);
        ax0=dm(temp_ax0);
      rts;

{_____________________output a character_____________________________

input:          ax1
modifies:       ax0, sr1, sr0, ar. Only need to save ax0 and ax1
_____________________________________________________________________________}

out_char_ar_:
        dm(temp_ax0)=ax0;
```

University of Calgary

```
        dm(temp_ax1)=ax1;
        ax1=ar;
wait1:  ax0=dm(flag_tx_ready);
      ar=pass ax0;
        if eq jump wait1;          { if tx word out still pending, then wait }
      dm(user_tx_buffer)=ax1;
        call invoke_UART_transmit;                              { send it out }
        ax0=dm(temp_ax0);
        ax1=dm(temp_ax1);
      rts;

{_____________________enable the RX section_____________________________________

modifies:       none
_____________________________________________________________________________}

turn_rx_on_:
        dm(temp_ax0)=ax0;
      ax0=0;
      dm(flag_rx_off)=ax0;
        ax0=dm(temp_ax0);
      rts;

{_____________________disable the RX section____________________________________

modifies:       none
_____________________________________________________________________________}

turn_rx_off_:
        dm(temp_ax0)=ax0;
      ax0=1;
      dm(flag_rx_off)=ax0;
        ax0=dm(temp_ax0);
      rts;

.endmod;
```

University of Cape Town

# A.4 Ana_ide.dsp

```
.MODULE           ana_ide;

{Written by Samuel Ginsberg
 This module contains the low level routines to interface to an ide type
 device, such as a CompactFlash or Flash drive.}

{Constants that represent the IDE commands}
.const ide_cmd_read= 0x0020;
.const ide_cmd_write=0x0030;
.const ide_cmd_id=0x00EC;

{Constants that represent the interconnection between the IDE control lines
 and the system bus}
.const ide_a0_line  =0x0001;
.const ide_a1_line  =0x0002;
.const ide_a2_line  =0x0004;
.const ide_cs1_line =0x0008;
.const ide_cs0_line =0x0010;
.const ide_rd_line  =0x0020;
.const ide_wr_line  =0x0040;
.const ide_reset_line =0x0080;

{Constants that represent the IDE registers as accessed in this system.
 These constants are derived from the system bus/control line
 interconnection}
.const ide_data =0x0008;
.const ide_err =0x0009;
.const ide_sec_cnt =0x000A;
.const ide_sector =0x000B;
.const ide_cyl_lsb =0x000C;
.const ide_cyl_msb =0x000D;
.const ide_head =0x000E;
.const ide_command =0x000F;
.const ide_status =0x000F;
.const ide_control =0x0016;
.const ide_astatus =0x0017;

{Variable Declarations}

.var/dm portvar_;                {keeps track of value on shared output port}
.var/dm lba1_;                                    {sector address variables}
.var/dm lba2_;
.var/dm lba3_;
.var/dm ide_dat_;             {data to be read from/written to an IDE register}
.var/dm buffer_ [256];         {data array to be read from/written to a sector}
.var/dm loopcnt;                                          {position in a sector}
.var/dm timeout;     {counter used to determine if an operation has timed out}
.var/dm address;                    {address of an IDE register to be accessed}
.var/dm ide_rd_ax0;               {temporary storage used to back up processor}
.var/dm ide_init_ax0;              {registers so that they don't get clobbered}
.var/dm ide_id_ax0;
.var/dm ide_read_i1;
.var/dm ide_read_ax0;
.var/dm ide_write_ax0;
.var/dm ide_err_ax0;
.var/dm ide_drq_ax0;
.var/dm ide_lba_ax0;
.var/dm dly_af;
.var/dm dly_ax0;
.var/dm temp_cntr;
```

University of Cape Town

```
{Global Variables used by other modules}
.global portvar_;
.global lba1_;
.global lba2_;
.global lba3_;
.global ide_dat_;
.global buffer_;

{Global Functions use by other modules}
.global ide_init_;
.global drive_id_;
.global setup_lba_;
.global read_sector_;
.global write_sector_;
.global make_drq_low_;

{============================================================================}
ide_init_:
{Power up initialization for the IDE device}
        dm (ide_init_ax0)=ax0;                          {save clobbered register}

        ax0=0x0000;
        IO(0x0003)=ax0;
        IO(0x0003)=ax0;
        IO(0x0003)=ax0;
        ax0=ide_reset_line;                     {pulse the drive's reset line}
        IO(0x0006)=ax0;
        IO(0x0006)=ax0;
        IO(0x0006)=ax0;
        call dly30;
        ax0=0;
        IO(0x0006)=ax0;
        IO(0x0006)=ax0;
        IO(0x0006)=ax0;
        call dly1;
init_loop:
        ax0=ide_head;                       {set up device as master,LBA mode}
        dm (address)=ax0;           {This is done by writing to the head and}
        ax0=0x00E0;                                             {device register}
        dm (ide_dat_)=ax0;
        call ide_wr;

        ax0=ide_status;         {check status reg bsy and rdy bits to see if}
        dm (address)=ax0;            {the device has finished initialization}
        call ide_rd;
        ax0=dm(ide_dat_);
        ar=tstbit 6 of ax0;
        if eq jump init_loop;  {poll device until initialization is complete}
        ar=tstbit 7 of ax0;
        if ne jump init_loop;

        ax0=dm(ide_init_ax0);               {restore the clobbered register}
        rts;

{============================================================================}
drive_id_:
{retrieve the identification data from the IDE device}
        dm(ide_id_ax0)=ax0;                         {backup clobbered register}

        call ide_busy;
        ax0=ide_command;
        dm(address)=ax0;
        ax0=ide_cmd_id;             {send the command to identify the device}
        dm (ide_dat_)=ax0;
        call ide_wr;
```

University of Cape Town

```
        call ide_drq;                {wait until the device has data waiting}
        call read_data_;                    {fetch the data from the device}

        ax0=dm(ide_id_ax0);                 {restore the clobbered register}
        rts;

{=============================================================================}
setup_lba_:
{setup the compactflash for LBA addressing. The block address is
 specified in the three 8 bit variables called lba1, lba2 and lba3.
 lba1 is the least significant.}

        dm(ide_lba_ax0)=ax0;                {backup the clobbered register}

       call ide_busy;
       ax0=ide_sec_cnt;   {set the sector counter register to 1. This system}
       dm(address)=ax0;              {only supports single sector transfers}
       ax0=0x0001;
       dm (ide_dat_)=ax0;
       call ide_wr;
       call dly1;

       call ide_busy;
       ax0=ide_cyl_msb;
       dm(address)=ax0;
       ax0=dm (lba3_);             {set up the address in the lba registers}
       dm (ide_dat_)=ax0;
       call ide_wr;
       call dly1;

       call ide_busy;
       ax0=ide_cyl_lsb;
       dm(address)=ax0;
       ax0=dm (lba2_);             {set up the address in the lba registers}
       dm (ide_dat_)=ax0;
       call ide_wr;
       call dly1;

       call ide_busy;
       ax0=ide_sector;
       dm(address)=ax0;
       ax0=dm (lba1_);             {set up the address in the lba registers}
       dm (ide_dat_)=ax0;
       call ide_wr;
       call dly1;

       call ide_busy;
       ax0=ide_head;
       dm(address)=ax0;
       ax0=0x00E0;              {set up the head and device register for LBA}
       dm (ide_dat_)=ax0;                          {mode, master device}
       call ide_wr;
       call dly1;

       ax0=dm(ide_lba_ax0);                {restore clobbered register}
       rts;

{=============================================================================}
read_sector_:
{read in the sector that was specified in the last setup_lba operation.}
        dm(ide_read_ax0)=ax0;

        call ide_busy;
        ax0=ide_command;
        dm(address)=ax0;
```

University of Cape Town

```
        ax0=ide_cmd_read;     {send the read sector command to the IDE device}
        dm (ide_dat_)=ax0;
        call ide_wr;
        call dly1;
        call ide_drq;  {wait for the device to signal that the data is ready}
        call read_data_;
        call dly1;

        ax0=dm(ide_read_ax0);                    {restore clobbered register}
        rts;

{=============================================================================}
read_data_:
{read 256 words from the IDE sector buffer and store in the array
 called buffer_. This function does not back up ax0 since this function is
 only called from other ide assembly functions.}

        dm(ide_read_i1)=i1;   {back up the clobbered memory pointer register}
        i1=^buffer_;  {place the address of the data buffer into the pointer}

        ax0=0x0100;          {retrieve 256 (hex100) words from the ide device}
        dm (loopcnt)=ax0;
read_data_loop:
        ax0=ide_data;
        dm (address)=ax0;
        call ide_rd;                          {read from the IDE data register}
        ax0=dm(ide_dat_);
        dm(i1,m1)=ax0;    {place the data into the location pointed to by i1}
        ay1= dm (loopcnt);           {i1 is used in automatic increment mode}
        ar=ay1-1;
        dm (loopcnt)=ar;                              {loop until all 256 words}
        if ne jump read_data_loop;                          {have been fetched}

        i1=dm(ide_read_i1);             {restore the memory pointer register}
        rts;

{=============================================================================}
write_sector_:
{write the contents of the array buffer_ into a sector on the
 compactflash. The address of the sector must be specified in a
 previous setup_lba_ operation.}
        dm(ide_write_ax0)=ax0;                   {back up clobbered register}

        call ide_busy;
        ax0=ide_command;
        dm(address)=ax0;
        ax0=ide_cmd_write; {issue the write sector command to the ide device}
        dm (ide_dat_)=ax0;
        call ide_wr;
        call dly1;
        call ide_drq;     {wait until the device is ready to accept the data}
        call write_data;               {send a sector of data to the device}
call dly1;

        ax0=dm(ide_write_ax0);                   {restore clobbered register}
        rts;

{=============================================================================}
write_data:
{write 256 words from the array called buffer_ into the IDE sector buffer.
 ax0 is not backed up since this function is only called from other ide
 assembly functions.}

        dm(ide_read_i1)=i1;   {back up the clobbered memory pointer register}
        i1=^buffer_;  {place the address of the data buffer into the pointer}
```

University of Cape Town

```
        ax0=0x0100;                     {send 256 words of data to the device}
        dm (loopcnt)=ax0;
write_data_loop:
        ax0=ide_data;
        dm (address)=ax0;
        ax0=dm (i1,m1);          {get data from the location pointed to by i1}
        dm (ide_dat_)=ax0;            {i1 is used in automatic increment mode}
        call ide_wr;               {write the data into the IDE data register}
        call dly3;          {delay to ensure that data hold time is exceeded}
        ay1= dm (loopcnt);
        ar=ay1-1;
        dm (loopcnt)=ar;
        if ne jump write_data_loop;     {loop until all data has been written}

        call make_drq_low_;      {if the drq line is still high then an error
                           has occurred. In order to prevent the error from
                                 corrupting future operations, dummy data is
                                                              sent to the device}
        i1=dm(ide_read_i1);             {restore the clobbered memory pointer}
        rts;

{=============================================================================}
ide_rd:
{read from an IDE register. Data is output in ide_dat_. The address of the
 register comes in via the variable called address.
 The function has been detuned from a speed point of view. This is for
 reliability. With the full speed version crashes could occur because of
 inadequate signal integrity.}

        dm(ide_rd_ax0)=ax0;                {save CPU registers in temp buffer}
        ax0=dm(portvar_);
        ar=ax0 and 0xEFFF;
        io (0x0004)=ar;           {clear the ide data bus ctrl bit to read in}
        io (0x0004)=ar;                     {from the bidirectional data bus}
        io (0x0004)=ar;
        dm (portvar_)=ar;
        cntr=0x0064;                  {delay while bus settles into new state}
        do end_dly1 until ce;
end_dly1:
        ax0=dm (address); {assert the register address, RD line not asserted}
        io(0x0006)=ax0;
        io(0x0006)=ax0;
        io(0x0006)=ax0;
        cntr=0x0064;
        do end_dly2 until ce;
end_dly2:
        ar= ax0 or ide_rd_line;
        io (0x0003)=ar;                                  {assert the RD line}
        io (0x0003)=ar;
        io (0x0003)=ar;
        cntr=0x0064;
        do end_dly3 until ce;
end_dly3:

        ax0=io(0x0007);
        ax0=io(0x0007);
        ax0=io(0x0007);
        dm (ide_dat_)=ax0;                    {read the data from the bus}
        cntr=0x0064;
        do end_dly4 until ce;
end_dly4:
        ax0=dm (address);
        io(0x0003)=ax0;                                {deassert the RD line}
        io(0x0003)=ax0;
```

University of Cape Town

```
        io(0x0003)=ax0;
        cntr=0x0064;
        do end_dly5 until ce;
end_dly5:
        ax0=0;
        io(0x0006)=ax0;                          {deassert all IDE address lines}
        io(0x0006)=ax0;
        io(0x0006)=ax0;
        cntr=0x0064;
        do end_dly6 until ce;
end_dly6:
        ax0=dm(ide_rd_ax0);              {restore the clobbered CPU registers}
        rts;

{=============================================================================}
ide_wr:
{write to an IDE register. The address comes in the variable called
 address_ and the data to be written comes in via the variable caled ide_dat_
 the speed has been detuned. See the notes above.}

        dm(ide_rd_ax0)=ax0;      {save clobbered CPU registers in temp buffer}
        ax0=dm (portvar_);
        ar=ax0 or 0x1000;
        io (0x0004)=ar;                {set the ide data bus ctrl bit to write}
        io (0x0004)=ar;                          {to the bidirectional data bus}
        io (0x0004)=ar;
        cntr=0x0064;
        do end_dly7 until ce;
end_dly7:

        ax0=dm (address);      {put out the register address, WR not asserted}
        io(0x0006)=ax0;
        io(0x0006)=ax0;
        io(0x0006)=ax0;
        cntr=0x00C8;
        do end_dly8 until ce;
end_dly8:

        ax0=dm (ide_dat_);
        io (0x0005)=ax0;                    {put out the data onto the ide bus}
        io (0x0005)=ax0;
        io (0x0005)=ax0;
        cntr=0x00C8;
        do end_dly9 until ce;
end_dly9:

        ax0=dm (address);
        ar= ax0 or ide_wr_line;
        io (0x0003)=ar;                                  {assert the WR line}
        io (0x0003)=ar;
        io (0x0003)=ar;
        cntr=0x00C8;
        do end_dly10 until ce;
end_dly10:

        ax0=dm(address);                                 {deassert WR line}
        io (0x0003)=ax0;
        io (0x0003)=ax0;
        io (0x0003)=ax0;
        cntr=0x0064;
        do end_dly11 until ce;
end_dly11:

        ax0=0;
        io (0x0006)=ax0;                          {deassert all ide lines}
```

University of Cape Town

```
        io (0x0006)=ax0;
        io (0x0006)=ax0;
        ax0=dm (portvar_);
        ar=ax0 and 0xEFFF;
        io (0x0004)=ar;          {clear the ide data bus ctrl bit to read in}
        io (0x0004)=ar;                                  {from the bidirectional bus}
        io (0x0004)=ar;

        ax0=dm(ide_rd_ax0);                    {restore the clobbered register}
        rts;

{=============================================================================}
ide_busy:
{wait for IDE device to deassert the bsy bit in the status register}

        dm (ide_drq_ax0)=ax0;                          {backup clobbered register}
        ax0=0x8FFF;          {maximum number of attempts before timeout occurs}
        dm (timeout)=ax0;
ide_busy_loop:
        ay1= dm (timeout);
        ar=ay1-1;
        dm (timeout)=ar;
        if eq rts;          {if timeout has ocurred exit. This can be detected
                              by the calling function by checking if timeout=0}

        ax0=ide_status;                          {read the ide status register}
        dm (address)=ax0;
        call ide_rd;
        ax0=dm (ide_dat_);
        ar=tstbit 7 of ax0;                {check if the busy bit is asserted}
        if ne jump ide_busy_loop;

        ax0=dm (ide_drq_ax0);                  {restore the clobbered register}
        rts;

{=============================================================================}
make_drq_low_:
{Force the drq bit in the status register by feeding data into the IDE data
 register. This is a safegaurd that is used to ensure that id an error occurs
 during a write sector operation the error is not carried over to subsequent
 device operations.}

      dm (ide_drq_ax0)=ax0;                        {back up clobbered register}

force_low:
        ax0=ide_status;
        dm (address)=ax0;
        call ide_rd;                                      {read the status bit}
        ax0=dm (ide_dat_);
        ar=tstbit 3 of ax0;                                {read the drq bit}
        if eq jump drq_low;                  {if drq is not asserted then exit}
        ax0=ide_data;              {If drq bit is asserterted send data to the}
        dm (address)=ax0;                                    {data register}
        ax0=0;
        dm (ide_dat_)=ax0;
        call ide_wr;
        jump force_low;                                {loop until drq is low}
drq_low:                                          {restore clobbered register}
      ax0= dm (ide_drq_ax0);
      rts;

{=============================================================================}

ide_drq:
{wait for IDE device to deassert the bsy bit and assert the DRQ bit in the
```

University of Cape Town

```
 status register. A timeout system protects against infinite loops should an
 error have occurred.}

        dm (ide_drq_ax0)=ax0;                    {back up clobbered register}
        ax0=0x8FFF;                  {set up the number of timeout attempts}
        dm (timeout)=ax0;
ide_drq_loop:
        ay1= dm (timeout);
        ar=ay1-1;
        dm (timeout)=ar;
        if eq rts;        {if timeout has ocurred exit. The calling function
                 can determine if this has happened by checking if timeout=0}

        ax0=ide_status;
        dm (address)=ax0;
        call ide_rd;                          {read the IDE status register}
        ax0=dm (ide_dat_);
        ar=tstbit 7 of ax0;                    {check if the device is busy}
        if ne jump ide_drq_loop;
        ar=tstbit 3 of ax0;               {check if the drq bit is asserted}
        if eq jump ide_drq_loop;
        call dly3;

        ax0=dm (ide_drq_ax0);                  {restore clobbered registers}
        rts;

{=========================================================================}
{This section contains a set of delay routines of various lengths. These
 routines use loops to determine period.}

dly30: dm(dly_ax0)=ax0;
       ax0=0x0BB8;
       jump dly;
dly3:  dm(dly_ax0)=ax0;
       ax0=0x012C;
       jump dly;
dly2:  dm(dly_ax0)=ax0;
       ax0=0x00C8;
       jump dly;
dly1:  dm(dly_ax0)=ax0;
       ax0=0x0064;
dly:   af=pass ax0;
       do end_dly until eq;
       af=af-1;
end_dly:
        ax0=dm(dly_af);
        af=pass ax0;
        ax0=dm(dly_ax0);
        rts;

        .ENDMOD;
```

University of Cape Town

# Appendix B

# C Program Files

## B.1 Ana.c

```
/*Written by Samuel Ginsberg
  This file contains the main core code of the Vibration analyzer. There are
  very many functions covering a wide range of functions*/

/*Include files. See the Linker group file for more libraries*/
#include <stdlib.h>
#include <math.h>                                    /*used for cos() and sin()*/

#define number_points 4096                         /*number of points in the FFT*/
#define pi 3.14149265
#define BASE 0x0001                                 /*base address for LCD port*/
#define s_scale 512
#define XMAX 127                          /*limits of (x,y) LCD graphics drawing*/
#define XMIN 0
#define YMAX 127
#define YMIN 0

#define fscaleacc 1                        /*scale factors for axis calibration.*/
#define fscalevel 1
#define fscaledis 1
#define tscaleacc 1
#define tscalevel 1
#define tscaledis 1

/*=========================================================================*/

/*Externally declared variables*/
extern int portvar;          /*holds the value on the port at address 0x0001*/
extern int state;                                 /*what state the analyzer is in*/
extern int change_peaks;       /*set if the change peaks button was pressed*/
extern int old_state;                            /*previous state of the analyzer*/
extern int show_peaks;                          /*set to 1 when peak finder is on*/
extern int full;                         /*set to 1 when the data arrays are full*/
extern int baud_period;     /*the period between bits for UART transmission*/
extern unsigned char clk_com;       /*command to send to the real time clock*/
extern unsigned char clk_dat;             /*data to get from/send to the clock*/
extern unsigned int triglev;   /*the input level at which triggering occurs*/
extern unsigned char lba1;           /*CompactFlash sector address variables*/
extern unsigned char lba2;
extern unsigned char lba3;
extern int buffer [3];                              /*CompactFlash sector buffer*/
extern int xreal [2];     /*data arrays. Each array is really 4096 elements*/
extern int ximag [2];
extern unsigned int soft0 [11];                    /*hotkey legend bitmap arrays*/
extern unsigned int soft1 [2];
extern unsigned int soft2 [2];
extern unsigned int soft3 [2];
extern unsigned int soft4 [2];
```

University of Cape Town

```
extern unsigned int soft5 [2];
extern unsigned int soft6 [2];
extern unsigned int soft7 [2];
extern unsigned int soft8 [2];
extern unsigned int soft9 [2];

/*Externally defined functions*/

extern void delay(unsigned long d);       /*delay proportional to "d" value*/
extern void dput(int byte);                 /*write data byte to LCD module*/
extern void cput(int byte);               /*write command byte to LCD module*/
extern void lcd_setup();    /*make sure control lines are at correct levels*/
extern void lcd_init();           /*initialize LCD memory and display modes*/
extern void lcd_print(char *string);     /*send string of characters to LCD*/
extern void lcd_clear_graph();              /*clear graphics memory of LCD*/
extern void lcd_clear_text();                    /*clear text memory of LCD*/
                            /*set memory pointer to (x,y) position (text)*/
extern void lcd_xy(int x, int y);
extern void lcd_put_char (char data);     /*put one character on the screen*/
                                        /*set single pixel in 128x128 array*/
extern void lcd_setpixel(int column, int row,int optype);
extern void lcd_putarray ();                      /*print a graph on the LCD*/
                                /*unpack the data coming out of the real FFT*/
extern void unpack (int *xreal,int *ximag,unsigned int N);
                                                  /*The real FFT function*/
extern void fft (int *xreal, int *ximag, unsigned int N, unsigned int nu);
extern double power (double x, double y);                 /*calculates x^y*/
          /*applies a window to the data before the FFT is tun on the data*/
extern void window (int *xreal, int* ximag, int win_type, int width);
                                    /*calculates the rms value of a signal*/
extern unsigned int find_rms (int *xreal, int *ximag);
                                  /*calculates the crest factor of a signal*/
extern unsigned int crest_factor (int *xreal, int *ximag,unsigned int rms);
extern int intlog (int arg);        /*calculates the log base 2 of a number*/
extern void savedata ();        /*saves the data arrays to the CompactFlash*/
extern void disp_saved_data ();       /*recalls data from the CompactFlash*/
extern void format ();                        /*formats the CompactFlash*/
extern unsigned int buttons ();                /*read in from the keypad*/
extern void init_uart ();                   /*initialize the UART emulator*/
extern unsigned int get_char_ar ();      /*read a byte from the serial port*/
extern void out_char_ar (unsigned int charout);  /*send a byte out the UART*/
extern void turn_rx_on ();                        /*enable UART reception*/
extern void turn_rx_off ();                      /*disable UART reception*/
extern void get_clk ();                 /*get data from the real time clock*/
extern void put_clk ();                 /*put data into the real time clock*/
extern void init_clk ();                  /*initialize the real time clock*/
extern void change_fs (unsigned int command);  /*change the ADC sample rate*/
extern void ide_init ();                        /*initialize the IDE device*/
extern void drive_id ();      /*get identification data from the IDE device*/
extern void setup_lba ();     /*set up the sector address on the IDE device*/
extern void read_sector ();             /*read a sector from the IDE device*/
extern void write_sector ();             /*write a sector to the IDE device*/

/*Functions defined within this module*/

void menu1 ();                   /*functions which implement the menu system*/
void menu2 ();
void menu3 ();
void menu4 ();
void menu5 ();
void menu6 ();
void menu7 ();
char* itoa (unsigned long in_num,char* input,int len);      /*int to string*/
void spectrum ();                          /*displays a vibration spectrum*/
void waveform ();                          /*displays a vibration waveform*/
```

University of Cape Town

```
void zoompan ();                    /*controls zooming and panning of graphics*/
void data_to_graph (int graphtype);              /*prepares data for display*/
void place_cursor (int optype);   /*places or removes a cursor on the graph*/
void update_cursor ();                          /*updates the cursor position*/
void freq_disp ();       /*displays the frequency represented by the cursor*/
void time_disp();              /*displays the time represented by the cursor*/
void find_peaks (int present);           /*finds the harmonics of the cursor*/
void place_circle (int x, int y,int optype);  /*puts marks on the harmonics*/
void setclk (int operation);      /*allows the user to set the time or date*/
void show_clk (int detail);                        /*shows the time or date*/
void clk_init ();                         /*initializes the real time clock*/
void clk_string (int detail);          /*pack the time or date into a string*/
void init_sample ();        /*sets up the sample rate to the power up value*/
void disp_soft (char pos, unsigned int* legend);  /*display a hotkey legend*/
void autorange ();                /*automatically set optimum vertical range*/
void triggerlevel ();                           /*set up the trigger level*/
void routemenu ();                            /*show the route manager menu*/
void loadroute ();                             /*download a route from a PC*/
void uploadroute ();                             /*upload a route to the PC*/
void selectroute ();                             /*set a route to be active*/
void deleteroute ();                                        /*delete a route*/
void walkroute ();                           /*walk the user through a route*/
void display_next_machine ();    /*prompt the user for the next route point*/
            /*wait for the user to release a button and press the next one*/
void waitforbutton (unsigned int maximum,unsigned int waitbutton);
void recall_mods ();            /*modify a user screen for flash data recall*/
void no_active_route();    /*display an error message if no route is active*/
void integ1hi ();                        /*manipulate the integrator system*/
void integ1lo ();
void integ2hi ();
void integ2lo ();

/*Variables declared in this module*/

char autorange_on;           /*equals 1 when the autorange feature is active*/
char overrange;      /*equals one when input data exceeds maximum allowable*/
char accelveldispl;        /*indicates if the display is in m/s^2, m/s or m*/
char active_file_selected;          /*set to 1 if a file has been selected*/
unsigned int nu=12;                  /*nu=(log (number_points))/(log (2));*/
unsigned int counter,pos;   /*general purpose counter and position tracking*/
unsigned int crest;                                          /*crest factor*/
unsigned int samples_per_sec;                       /*current sample rate*/
unsigned int rate_var;          /*holds the sample rate info in ADC format */
int spare, count, poscount, max;          /*general purpose global integers*/
int number_per_line=32;                              /*used as a zoom factor*/
int start_display=0;                       /*used to control pan position*/
int text_x,text_y;                            /*position of text cursor*/
int curs_x=10,curs_y=63;                    /*position of measurement cursor*/
int window_type=0;                  /*what type of windowing is used on data*/
int harmonic_disp=0;     /*set to 1 when harmonic markers must be displayed*/
int range;    /*holds the range info, i.e. which range the instrument is on*/
int fat_start;      /*The LBA address of the start of the flash drive's FAT*/
int fat_end;          /*The LBA address of the end of the flash drive's FAT*/
int curr_fat_entry;     /*The LBA address of the current entry's FAT record*/
int curr_route_entry; /*address of entry of active route, zero for no route*/
int curr_route_pos;                          /*where we are in the route*/
int walking_route;  /*=1 if busy walking route =0 if not busy walking route*/
int route_active_file;  /*when walking a route store waveforms to this file*/
int transformed;                        /*records if a fft was done on data*/
int rms;                                          /*RMS value of a waveform*/
float x,y;                              /* coordinates on graphics screen*/
double mag,mag1,mag2;       /*used to compute magnitude spectrum of the FFT*/

char number [6];                      /*string used to hold converted number*/
char clockstring [9];            /*string used to hold converted clock data*/
```

University of Cape Town

```
int screen_buf [4096];            /*buffer which holds last computed result*/
int maxima [128];      /*stores the exact positions of maxima in screen_buf*/
int circles [128];                  /*stores the location of circle markers*/
int string[320];                                  /*string to print to display*/
int grapharray [128]; /*array that is printed on graphics screen as a graph*/

/*============================================================================*/
void main()
{
 unsigned int keypad=0;                /*start by setting up some variables*/
 active_file_selected=0;
 curr_route_entry=0;
 baud_period=189;                              /*baud rate: 189 gives 56700bps*/

 init_uart ();                                      /*set up the UART emulator*/
 lcd_setup();               /*make sure control lines are at correct levels*/
 lcd_init();                     /*initialize LCD memory and display modes*/
 lcd_clear_text ();            /*display a message while booting the system*/
 lcd_clear_graph ();
 lcd_xy (2,8);
 lcd_print ("Samuel Ginsberg");
 clk_init ();                                 /*initialize the real time clock*/
 triglev=0;                             /*set the trigger level to its default*/
 range=1;                                        /*default autorange value*/
 autorange_on=1;                                /*autoranging defaults to on*/
 accelveldispl=1;                                /*defaults to acceleration*/
 integ1hi ();         /*set up integrators to produce acceleration waveform*/
 integ2hi ();
 init_sample();                /*set up the sample rate from power up value*/
 cput (0xA7);
 cput(0x9D);                                          /*graphics and text on*/
 state=2;                                               /*menu display mode*/
 old_state=2;

 lba1=0;                                          /*initialize the IDE device*/
 lba2=0;
 lba3=0;
 ide_init ();
 drive_id ();                /*get identification data from the IDE device*/

 menu1 ();                                             /*show the main menu*/


 while (1)                              /*outer loop for instrument operation*/

 {keypad=buttons();                                          /*read the keypad*/
  if ((state==1)||(state==3))                        /*if time domain display*/
     {if ((keypad==4)||(keypad==8))   /*left or right arrow buttons pressed*/
                          {update_cursor();        /*update cursor position*/
                           delay (4000);     /*delay gives the user control*/
                           time_disp();  /*display the time shown by cursor*/
                          }

      if ((keypad>8)&&(keypad<256))        /*if zoom or pan buttons pressed*/
                          {zoompan();          /*control zooming and panning*/
                           lcd_clear_graph ();         /*clear the graphics*/
                           disp_soft (1,soft1);      /*renew hotkey legends*/
                           disp_soft (3,soft4);
                           disp_soft (4,soft5);
                           update_cursor ();   /*also renews the data graph*/
                           delay (4000);                /*pause for control*/
                          }
     }
```

University of Cape Town

```
  if (state==1)                                    /*if monitoring time domain*/
       {                 /*save data to current file, if not walking a route*/
        if ((keypad==256)&&(walking_route==0)) savedata ();

        if (walking_route==1)                /*if the user is walking a route*/
            {switch (keypad)
                {case 256: savedata ();                          /*save button*/
                           display_next_machine ();
                           break;
                 case 512: display_next_machine ();              /*skip button*/
                           break;
                 case 1024: curr_route_pos-=2;                   /*back button*/
                            display_next_machine ();
                }
            }
        if (keypad==2) triggerlevel ();   /*trigger level adjustment system*/
       }

  if (state==2) menu1 ();  /*if state is 2 then the menu has been requested*/

  if (full)              /*if the data buffers are full of unprocessed data*/

    {
     if ((state!=2)&&(transformed==0))   /*if not in menu mode and new data*/
               {rms=find_rms (xreal,ximag);                /*calculate new rms*/
                crest=crest_factor (xreal,ximag,rms); /*calculate new crest*/
               }
     if (state<3) autorange ();         /*if monitoring data then autorange*/
     if (state==2) menu1 ();                                /*if in menu mode*/

     if ((state==1)||(state==3)) waveform ();      /*time domain display*/
     keypad=buttons();                                       /*read keypad*/
     if ((keypad>8)&&(keypad<256)&&(state!=2)) zoompan();    /*zoom and pan*/
     if ((keypad==2)&&(state==1)) triggerlevel ();      /*set trigger level*/

     if (state!=2) {place_cursor (1);      /*display assorted info for user*/
                    lcd_xy (0,13);
                    lcd_print ("RMS:");
                    lcd_print (itoa (rms,number,5));
                    lcd_xy (0,14);
                    lcd_print ("CREST:");
                    lcd_print (itoa (crest,number,5));
                    lcd_xy (12,13);
                    lcd_print ("RANGE:");
                    lcd_put_char (range+48);
                   }

     if ((state==0)||(state==4)) spectrum ();  /*spectrum modes*/

     if (state<3) full=0;/*when in recall mode don't let ADC overwrite data*/
    }                                                       /*end  if (full)*/
  }                                                          /*end while (1)*/
}                                                                /*end main*/

/*=========================================================================*/
void integ1hi ()
/*turn off integrator 1*/
{portvar=portvar|0x4000;        /*must not change the value on the LCD port*/
 pput (portvar);
}

void integ1lo ()
/*turn on integrator 1*/
{portvar=portvar&0xBFFF;
 pput (portvar);
```

University of Cape Town

```
}

void integ2hi ()
/*turn off integrator 2*/
{portvar=portvar|0x2000;
 pput (portvar);
}

void integ2lo ()
/*turn on integrator 2*/
{portvar=portvar&0xDFFF;
pput (portvar);
}

/*=========================================================================*/
void autorange ()

/*This function autoranges the data coming in from the ADC. The basic
  operation is as follows: Look through the whole data array and find
  the max positive or negative point. find the scale factor that makes
  the point lie in the allowable range. Adjust the range variable to match.
  allowable max output is 16000*/

{int count;
 float max;

 overrange=0;
 max=0;
 for (count=0;count<4096;count++)   /*look at the data and find the maximum*/
  {if (abs(xreal[count])>max) max=abs(xreal[count]);
   if (abs(ximag[count])>max) max=abs(ximag[count]);
  }

 if (autorange_on==1)                              /*if autoranging is enabled*/
  {if (max<160) range=200; /*choose a range based on the biggest data point*/
   if ((max>=160)&&(max<1600)) range=20;
   if ((max>=1600)&&(max<16000)) range=2;
   if (max>=16000) range=1;
  }

 for (count=0;count<4096;count++)   /*scale the data according to the range*/
     {xreal[count]*=range;
      xreal[count]/=2;
      ximag[count]*=range;
      ximag[count]/=2;
     }
 if ((max*range/2)>16000) overrange=1;     /*if data is too big set a flag to
                                             inform the rest of the system*/
}

/*=========================================================================*/
void triggerlevel ()
/*this function sets up the trigglev variable which tells the sampling
functions at what voltage to start filling the xreal and ximag arrays.*/

{int count,keypad=0;
 disp_soft (1,soft0);                  /*display appropriate hotkey legends*/
 disp_soft (1,soft6);
 disp_soft (2,soft0);
 disp_soft (3,soft7);
 disp_soft (4,soft8);
 if ((triglev*range/1024)<31)                  /*if triggerlevel can be shown*/
                                  /*trigger level setting depends on range*/
                          for (count=0;count<128;count+=2)
                                              /*draw the level indicator line*/
```

University of Cape Town

```
                            lcd_setpixel (count,(64-(triglev*range/1024)),1);
 else for (count=0;count<128;count+=2)  /*if trigger level is above maximum*/
                                          /*draw the level indicator line*/
                            lcd_setpixel (count,32,1);

 while (keypad!=1)                    /*until the user presses the quit button*/
   {keypad=buttons();
    if ((keypad==4)&&((triglev*range/1024)<31))
                                        /*if trigger level is to be raised*/
                                          triglev=triglev+(1024/range);
    if ((keypad==8)&&((triglev*range/1024)>0))
                                       /*if trigger level is to be lowered*/
                                      triglev=triglev-(1024/range);
    if ((keypad==4)||(keypad==8))          /*if trigger level was changed*/
              {lcd_clear_graph();                    /*clear all graphics*/
               disp_soft (1,soft6);                /*renew hotkey legends*/
               disp_soft (3,soft7);
               disp_soft (4,soft8);
               data_to_graph (2);                  /*convert data to graph*/
               lcd_putarray ();                     /*renew data graphics*/
               for (count=0;count<128;count+=2)  /*place level marker line*/
               if ((triglev*range/1024)<32)
               lcd_setpixel (count,(64-(triglev*range/1024)),1);
              }
   }
 while (buttons()==1);      /*so that it goes back to waveform mode cleanly*/
 delay (1000);                                          /*debounce delay*/
 state=1;                                /*go back to waveform display mode*/
}

/*=========================================================================*/
void disp_soft (char pos, unsigned int* legend)
/*This function puts the soft key legend at the bottom of the screen
  The variable pos tells where the legend is to be placed. pos==1
  places the legend above the leftmost key, pos==4 above the rightmost. There
  are a maximum of four hotkey legends at any time.
  legend is a pointer to the bitmap array with the text to display */

{int count1,count2,col,row,index;
 unsigned int current;
             /*calculate leftmost column on the graphics screen for hotkey*/
 col=(((32-(3*legend[0]))/2)+((pos-1)*32));

 for (index=1;index<=legend[0];index++)  /*legend[0] holds length of bitmap*/
  {current=legend[index];
   for (count1=0;count1<3;count1++)                        /*column counter*/
       {row=123;                          /*top row of hotkey legend*/
        for (count2=0;count2<5;count2++)                    /*row counter*/
            {if (current&0x4000) lcd_setpixel (col,row,1);  /*place pixel*/
             else lcd_setpixel (col,row,0);                 /*blank pixel*/
             current*=2;          /*rotate the current word left by one bit*/
             row++;
            }
        col++;
       }
  }
}

/*=========================================================================*/
void init_sample ()
/*set up the sample rate according to the value that was selected at last
  power down. This is stored in the RAM of the real time clock*/

{clk_com=0x00F1;            /*address of sample rate in the real time clock*/
 get_clk ();                              /*read the data from the clock*/
```

University of Cape Town

```
 delay(1000);
 rate_var=clk_dat;
 switch (clk_dat&0x00FF)              /*look up sample rate from clock data*/
      {case 0x0041: samples_per_sec=2000; break;
       case 0x0043: samples_per_sec=4000; break;
       case 0x0047: samples_per_sec=8000; break;
       case 0x004B: samples_per_sec=16000; break;
       case 0x0046: samples_per_sec=32000; break;
       case 0x004C: samples_per_sec=48000; break;
         default: clk_dat=0x004C;          /*if clock contains invalid data*/
                  samples_per_sec=48000;
                  clk_com=0x00F0;      /*put arbitary valid data into clock*/
                  put_clk ();
      }
 change_fs (0xC800+(clk_dat&0x00FF));                  /*change sample rate*/
}

/*=============================================================================*/
void freq_disp ()
/*display the frequency represented by the cursor and the magnitude
  represented by that position*/

{float freq,size;
 char outstring[9];
                          /*calculate the frequency represented by the point*/
 freq=maxima[curs_x]*(float)(samples_per_sec/81.92);
 lcd_xy (0,12);
 lcd_print (itoa (freq, outstring,8));
 lcd_print ("Hz");
 lcd_xy (18,12);
 size=screen_buf[maxima[curs_x]];      /*look up the magnitude of the point*/
 if (accelveldispl==1)                                    /*acceleration mode*/
          {size*=fscaleacc;          /*scale the magnitude for acceleration*/
           lcd_print ("g");
          }
 if (accelveldispl==2)                                      /*velocity mode*/
          {size*=fscalevel;          /*different scale factor for velocity*/
           lcd_print ("m/s");
          }
 if (accelveldispl==3)                                    /*displacement mode*/
          {size*=fscaledis;                /*displacement mode scale factor*/
           lcd_print ("m");
          }
          lcd_xy (12,12);
          lcd_print (itoa (size, outstring,6));    /*print scaled magnitude*/
}

/*=============================================================================*/
void time_disp ()
/*display the time represented by the cursor and the magnitude represented
 by that point*/

{float time,size;
 char outstring[8];
                              /*calculate the time represented by the point*/
 time=maxima[curs_x]*(float)(204800/samples_per_sec);
 lcd_xy (0,12);
 lcd_print (itoa (time, outstring,7));                     /*print the time*/
 lcd_print ("ms");
 lcd_xy (18,12);
 size=screen_buf[maxima[curs_x]];          /*look up the unscaled magnitude*/
 if (accelveldispl==1)                                    /*acceleration mode*/
          {size*=tscaleacc;     /*scale factor for time domain acceleration*/
           lcd_print ("g");
          }
```

University of Cape Town

```
 if (accelveldispl==2)                                          /*velocity mode*/
         {size*=tscalevel;         /*scale factor for time domain velocity*/
          lcd_print ("m/s");
         }
 if (accelveldispl==3)                                      /*displacement mode*/
         {size*=tscaledis;     /*scale factor for time domain displacement*/
          lcd_print ("m");
         }
         lcd_xy (12,12);
         lcd_print (itoa (size, outstring,6));    /*print scaled magnitude*/
}

/*=============================================================================*/
void find_peaks(int present)
/*This function puts or deletes a circular marker on all of the harmonics
  of the cursor position. Present==0 deletes all markers present!=0 places
  markers*/

{int count,pos,circlepos;
                                                    /*first delete old circles*/
 for (count=0;count<128;count++)
     {if (circles[count]!=0) place_circle (count,circles[count],0);
      circles[count]=0;                          /*clear out the circles array*/
            /*turn off harmonic display. This is turned on again if needed*/
      harmonic_disp=0;
     }
                  /*restore the screen display from where the circles were*/
 data_to_graph (2);
 lcd_putarray ();

 if ((present!=0)&&(grapharray[curs_x]>0))     /*if needed draw new circles*/
  {harmonic_disp=1; /*circles are to be drawn so assume that peak finder on*/
                                                  /*find fundamental frequency*/
   pos=start_display+((curs_x+1)*number_per_line);
   for (count=2*pos;count<4096;count+=pos)   /*counts through the harmonics*/

/*this next section is involved with finding out if a peak is a harmonic or
  not. There is a big possibility that frequencies may be represented by FFT
  outputs which are not exactly at harmonic points. This is because of
  limited frequency resolution. For this reason three points are checked per
  harmonic, thus ensuring that if a harmonic has been represented by an
  adjacent spectral element it will be recognized as the harmonic that it
  really is.*/

     {circlepos=((count-start_display)/number_per_line);
      if (circlepos>127) break;
      if (grapharray[circlepos]>0)
              {circles[circlepos]=(64-grapharray[circlepos]);
                continue;
              }
      if (grapharray[(1+circlepos)]>0)
              {circles[circlepos+1]=(64-grapharray[circlepos]);
                continue;
              }
      if (grapharray[(circlepos-1)]>0)
              {circles[(circlepos-1)]=(64-grapharray[circlepos]);
                continue;
              }

     circlepos=((count-start_display)/number_per_line)-1;
      if (grapharray[circlepos]>0)
              {circles[circlepos]=(64-grapharray[circlepos]);
                continue;
              }
      if (grapharray[(1+circlepos)]>0)
```

University of Cape Town

```
                {circles[circlepos+1]=(64-grapharray[circlepos]);
                     continue;
                    }
       if (grapharray[(circlepos-1)]>0)
                {circles[(circlepos-1)]=(64-grapharray[circlepos]);
                     continue;
                    }

      circlepos=((count-start_display)/number_per_line)-2;
       if (grapharray[circlepos]>0)
                {circles[circlepos]=(64-grapharray[circlepos]);
                     continue;
                    }
       if (grapharray[(1+circlepos)]>0)
                {circles[circlepos+1]=(64-grapharray[circlepos]);
                     continue;
                    }
       if (grapharray[(circlepos-1)]>0)
                {circles[(circlepos-1)]=(64-grapharray[circlepos]);
                     continue;
                    }
      }
    for(count=0;count<128;count++)                /*place new circle markers*/
        if (circles[count]>0) place_circle (count,circles[count],1);
   }
}

/*=====================================================================*/
void place_circle (int x, int y,int optype)
/*place or delete a circular marker. The position is spacified by x and y
  and optype determines if the marker is placed or deleted.*/

{int count;
/*a circular markerb is approximated by two diagonal lines of three pixels
  each and two additional pixels.*/

 for (count=0;count<=2;count++)
   {lcd_setpixel ((x-count),(y-2+count),optype);          /*diagonal lines*/
    lcd_setpixel ((x+count),(y+2-count),optype);
   }
 lcd_setpixel ((x+1),(y-1),optype);                    /*additional pixels*/
 lcd_setpixel ((x-1),(y+1),optype);
}

/*=====================================================================*/
void menu1 ()
/*The top menu function*/

{unsigned int keypad=0;
 lcd_clear_text ();
 lcd_clear_graph ();
 lcd_xy (3,0);
 lcd_print ("ANALYZE MENU");
 lcd_xy (0,2);
 lcd_print ("1. Monitor");
 lcd_xy (0,3);
 lcd_print ("2. Select data file");
 lcd_xy (0,4);
 lcd_print ("3. Flash utilities");
 lcd_xy (0,5);
 lcd_print ("4. Setup parameters");
 lcd_xy (0,6);
 lcd_print ("5. Display saved data");
 lcd_xy (0,7);
 lcd_print ("6. Upload file to PC");
```

University of Cape Town

```
 lcd_xy (0,8);
 if (old_state==1) lcd_print ("7. Return to waveform");
 if (old_state==0) lcd_print ("7. Return to spectrum");
 while (keypad<128) keypad=buttons();              /*wait for user keypress*/
  switch (keypad)
    {case 128: menu2 (); break;                                 /*monitor menu*/
     case 256: menu7 (); break;                               /*select data file*/
     case 512: menu6 (); break;                                /*flash utilities*/
     case 1024: menu3 (); break;                              /*setup parameters*/
     case 2048: menu4 (); break;                            /*display saved data*/
     case 4096: menu5 (); break;                                 /*upload to PC*/
     case 8192: state=old_state;           /*return to waveform or spectrum*/
    }
   lcd_clear_text ();
}

/*============================================================================*/
void menu2 ()
/*Monitor menu function*/

{int count;
 lcd_clear_text ();
 lcd_xy (4,0);
 lcd_print ("MONITOR MENU");
 lcd_xy (0,2);
 lcd_print ("1. Spectrum");
 lcd_xy (0,3);
 lcd_print ("2. Waveform");
 disp_soft (1,soft6);                                        /*hotkey display*/
 while (buttons()==128);   /*wait for user to remove finger from '1' button*/
 delay (250);                                              /*software debounce*/

 while (1)
  {if (buttons()==1)   {state=2;                                 /*quit button*/
                        break;
                       }
   if (buttons()==128) {state=0;                             /*spectrum selected*/
                        break;
                       }
   if (buttons()==256) {state=1;                             /*waveform selected*/
                        break;
                       }
  }

 lcd_clear_text();
 if (!(state==old_state))          /*clear out of date data off the screen*/
             {curs_x=64;
              lcd_clear_graph();
              for (count=0;count<4096;count++) screen_buf[count]=0;
              for (count=0;count<128;count++) grapharray[count]=0;
             }
 update_cursor();
}

/*============================================================================*/
void menu3 ()
/*setup parameters menu*/

{unsigned int temp; /*used to hold rate var to check if it has been changed*/
 lcd_clear_text ();
 lcd_xy (2,0);
 lcd_print ("SETUP PARAMETERS");
 lcd_xy (0,2);
 lcd_print ("1. Window type");
 lcd_xy (0,3);
```

University of Cape Town

```
 lcd_print ("2. Time/Date");
 lcd_xy (0,4);
 lcd_print ("3. Frequency range");
 lcd_xy (0,5);
 lcd_print ("4. Vertical range");
 lcd_xy (0,6);
 lcd_print ("5. Accel/Veloc/Displ");
 disp_soft (1,soft6);
waitforbutton (2048,1024);          /*wait for the '4' button to be released
                                                and new button pressed*/
  if (buttons()==1) {state=2; return;}                        /*quit button*/
  if (buttons()==256)
        {lcd_clear_text ();          /*set time or date menu*/
         lcd_xy (4,0);
         lcd_print ("SET TIME/DATE");
         lcd_xy (0,2);
         lcd_print ("1. Set time");
         lcd_xy (0,3);
         lcd_print ("2. Set date");
         disp_soft (1,soft6);
                        /*wait for user to remove finger from '1' button*/
         while (buttons()==256);
         delay (250);                                  /*software debounce*/
                                                    /*wait for a keypress */
         while (((buttons()<128)||(buttons()>256))&&(buttons()!=1))
                    {lcd_xy (0,14);
                     show_clk (2);
                    }
         if (buttons()==128) setclk (1);     /*allow the user to set time*/
         if (buttons()==256) setclk (2);     /*allow the user to set date*/
         if (buttons()==1) return;                        /*quit button*/
        }

  if (buttons()==128) {lcd_clear_text ();                 /*setup window type*/
                  lcd_xy (4,0);
                  lcd_print ("WINDOW TYPE");
                  lcd_xy (0,2);
                      lcd_print ("1. Rect");
                  lcd_xy (0,3);
                  lcd_print ("2. Triangular");
                  lcd_xy (0,4);
                  lcd_print ("3. Hamming");
                  lcd_xy (0,5);
                  lcd_print ("4. Hanning");
                      lcd_xy (2,(2+window_type));/*indicate current window*/
                  lcd_put_char ('*');
                     waitforbutton (1024,128);   /*wait for next keypress*/
                     if (buttons()==1) return;                    /*quit*/
                     if (buttons()==128) window_type=0;     /*rectangular*/
                     if (buttons()==256) window_type=1;      /*triangular*/
                     if (buttons()==512) window_type=2;         /*Hamming*/
                     if (buttons()==1024) window_type=3;        /*Hanning*/
                                 /*wait for user to stop pressing button*/
                     while ((buttons()>=128)&&(buttons()<=1024));
                    }                           /*end of if buttons()==128*/

   if (buttons()==512){lcd_clear_text ();          /*select frequency range*/
                  lcd_xy (2,0);
                  lcd_print ("FREQUENCY RANGE");
                  lcd_xy (0,2);
                  lcd_print ("1. 1 kHz");
                  lcd_xy (0,3);
                  lcd_print ("2. 2 kHz");
                  lcd_xy (0,4);
                  lcd_print ("3. 4 kHz");
```

University of Cape Town

```
                    lcd_xy (0,5);
                    lcd_print ("4. 8 kHz");
                    lcd_xy (0,6);
                    lcd_print ("5. 16 kHz");
                    lcd_xy (0,7);
                    lcd_print ("6. 24 kHz");
                    disp_soft (1,soft6);
                        switch (samples_per_sec)     /*indicate current rate*/
                   {case 2000: lcd_xy (2,2); break;
                    case 4000: lcd_xy (2,3); break;
                    case 8000: lcd_xy (2,4); break;
                    case 16000: lcd_xy (2,5); break;
                    case 32000: lcd_xy (2,6); break;
                    case 48000: lcd_xy (2,7); break;
                   }
                   lcd_put_char ('*');
                        waitforbutton (4096,512);
                        if (buttons()==1)                    /*quit button*/
                                        {state=2;
                                         return;
                                        }
                 /*store current rate so that we can see if rate_var changes*/
                        temp=rate_var;
                     if ((buttons()==128)&&(samples_per_sec!=2000))
                                    {samples_per_sec=2000;
                                     rate_var=0xc841;}
                     if ((buttons()==256)&&(samples_per_sec!=4000))
                                    {samples_per_sec=4000;
                                     rate_var=0xc843;}
                     if ((buttons()==512)&&(samples_per_sec!=8000))
                                    {samples_per_sec=8000;
                                     rate_var=0xc847;}
                     if ((buttons()==1024)&&(samples_per_sec!=16000))
                                     {samples_per_sec=16000;
                                      rate_var=0xc84B;}
                     if ((buttons()==2048)&&(samples_per_sec!=32000))
                                     {samples_per_sec=32000;
                                      rate_var=0xc846;}
                     if ((buttons()==4096)&&(samples_per_sec!=48000))
                                     {samples_per_sec=48000;
                                      rate_var=0xc84C;}
                                   /*only change rate if rate actually changed*/
                     if (temp!=rate_var) change_fs (rate_var);
                                      /*store sample rate in real time clock*/
                     clk_dat=rate_var&0x00FF;
                                     /*wait for user to stop pressing button*/
                         while ((buttons()>=128)&&(buttons()<=4096));
                        /*give the clock time to settle after button scanning*/
                         delay (100000);
                     clk_com=0x00F0;
                     put_clk ();
                        }                              /*end of if buttons()==512*/

if (buttons()==1024) {lcd_clear_text ();              /*set vertical range*/
                      lcd_xy (4,0);
                      lcd_print ("VERTICAL RANGE");
                      lcd_xy (0,2);
                      lcd_print ("1. Big input");
                      lcd_xy (0,3);
                      lcd_print ("2. Medium input");
                      lcd_xy (0,4);
                      lcd_print ("3. Small input");
                      lcd_xy (0,5);
                      lcd_print ("4. Tiny input");
                      lcd_xy (0,6);
```

University of Cape Town

```
                        lcd_print ("5. Autorange");
                        if (range==1) lcd_xy (2,2);    /*show current range*/
                        if (range==2) lcd_xy (2,3);
                        if (range==20) lcd_xy (2,4);
                        if (range==200) lcd_xy (2,5);
                        if (autorange_on==1) lcd_xy (2,6);
                        lcd_put_char ('*');
                        disp_soft (1,soft6);
                        waitforbutton (2048,1024);/*wait for next keystroke*/
                          if (buttons()==1) return;          /*quit button*/
                              /*if the user sets range turn off autoranging*/
                          if (buttons()==128) {autorange_on=0;
                                               range=1;
                                              }
                          if (buttons()==256) {autorange_on=0;
                                               range=2;
                                              }
                          if (buttons()==512) {autorange_on=0;
                                               range=20;
                                              }
                          if (buttons()==1024) {autorange_on=0;
                                                range=200;
                                               }
                          if (buttons()==2048) autorange_on=1;
                                    /*wait for user to stop pressing button*/
                         while ((buttons()>=128)&&(buttons()<=4096));
                       }                              /*end of if buttons()==1024*/

  if (buttons()==2048) {lcd_clear_text ();    /*set up accel/velocity/displ*/
                        lcd_xy (4,0);
                        lcd_print ("ACCEL/VELOC/DISPL");
                        lcd_xy (0,2);
                        lcd_print ("1. Acceleration");
                        lcd_xy (0,3);
                        lcd_print ("2. Velocity");
                        lcd_xy (0,4);
                        lcd_print ("3. Displacement");
                                          /*indicate current setting*/
                        if (accelveldispl==1) lcd_xy (2,2);
                        if (accelveldispl==2) lcd_xy (2,3);
                        if (accelveldispl==3) lcd_xy (2,4);
                        lcd_put_char ('*');
                        disp_soft (1,soft6);
                        waitforbutton (512,2048);      /*wait for keystroke*/
                          if (buttons()==1) return;          /*quit button*/
                          if (buttons()==128) {accelveldispl=1;
                                               integ1hi (); /*turn off both*/
                                               integ2hi ();   /*integrators*/
                                              }
                          if (buttons()==256) {accelveldispl=2;
                                               integ1lo ();/*one integrator*/
                                               integ2hi ();   /*on, one off*/
                                              }
                          if (buttons()==512) {accelveldispl=3;
                                               integ1lo ();   /*integrators*/
                                               integ2lo ();       /*both on*/
                                              }
                                    /*wait for user to stop pressing button*/
                        while ((buttons()>=128)&&(buttons()<=4096));
                       }                                  /*if buttons==2048*/
   state=2;                                          /*go back to main menu*/
}

/*=========================================================================*/
void menu4 ()
```

University of Cape Town

```
/*This is the display saved data menu.*/

{int temp_autorange_on;
 lcd_clear_text ();
                        /*if no file currently active give an error message*/
 if (active_file_selected==0) {lcd_xy (0,1);
                               lcd_print ("No file!");
                               delay (1000000);
                               lcd_xy (0,1);
                               lcd_print ("               ");
                               state=2;
                               return;
                              }
 lcd_xy (2,0);
 lcd_print ("DISPLAY SAVED DATA");
 lcd_xy (0,2);
 lcd_print ("1. View record - time");
 lcd_xy (0,3);
 lcd_print ("2. View record - freq");
 waitforbutton (256,2048);                        /*wait for user's keystroke*/
 if (buttons ()==1) return;
 if (buttons()==128) state=3;     /*state=3 means get record in time domain*/
 if (buttons()==256) {state=4;                    /*get record in freq domain*/
                      transformed=0;     /*mark the data as being unprocessed
       this is needed so that we don't process the same data more than once*/
                     }
 full=1;                        /*mark the main data arrays as full so that the
                                                          ADC can't overwrite them*/
 disp_saved_data ();     /*get the data from the flash into the data arrays*/
 temp_autorange_on=autorange_on;   /*store status of the autorange function*/
 autorange_on=0;              /*turn off autorange and recall the range from*/
 range=string[19];                                        /*the stored data*/
 autorange ();                                 /*apply the range to the data*/
 overrange=0;                               /*there should never be an autorange*/
 autorange_on=temp_autorange_on;             /*reinstate the autoranger status*/
}

/*=========================================================================*/
void menu5 ()
/*Upload a single record to the host PC*/

{if (active_file_selected==0)     /*check to see if there is an active file*/
                              {lcd_clear_text ();
                               lcd_xy (0,1);
                               lcd_print ("No file!");
                               delay (1000000);
                               state=2;
                               return;
                              }
 upload_file ();                                    /*send the file to the PC*/
 state=2;
}

/*=========================================================================*/
void menu6 ()
/*Flash utilities menu*/

{int count=0,temp;

 lcd_clear_text ();
 lcd_xy (4,0);
 lcd_print ("Flash utilities");
 lcd_xy (0,2);
 lcd_print ("1. Format flash");
 lcd_xy (0,3);
```

University of Cape Town

```
lcd_print ("2. Select data file");
lcd_xy (0,4);
lcd_print ("3. Kill record");
lcd_xy (0,5);
lcd_print ("4. Route Manager");
lcd_xy (0,6);
lcd_print ("5. Get record from PC");
lcd_xy (0,7);
lcd_print ("6. Make a record");

disp_soft (1,soft6);

waitforbutton (4096,512);                          /*wait for a keypress*/

if (buttons()==128)                                /*format flash device*/
                    {disp_soft (1,soft0);         /*clear off clear button*/
                     format();
                    }

if (buttons()==2048)                                /*get record from PC*/
                     {lcd_clear_text ();
                      lcd_xy (2,0);
                      lcd_print ("Name loose record");
                      lcd_xy (0,2);
                      lcd_print ("To load a loose");
                      lcd_xy (0,3);
                      lcd_print ("record name from a PC");
                      lcd_xy (0,4);
                      lcd_print ("attach the RS232 link");
                      lcd_xy (0,5);
                      lcd_print ("and press 'SET' on");
                      lcd_xy (0,6);
                      lcd_print ("this unit and 'load'");
                      lcd_xy (0,7);
                      lcd_print ("on the PC.");
                      lcd_xy (0,9);
                      lcd_print ("If you do not wish to");
                      lcd_xy (0,10);
                      lcd_print ("load from the PC hit");
                      lcd_xy (0,11);
                      lcd_print ("'QUIT' on this unit.");
                      disp_soft (2,soft3);           /*display set hotkey*/

                     while (buttons()==2048);     /*wait for next keypress*/
                     while (buttons()<1);
                     if (buttons()==2)               /*set button pressed*/
                       {disp_soft (1,soft0);        /*clear hotkey legends*/
                        disp_soft (2,soft0);
                        lcd_clear_text ();
                        lcd_xy (1,8);
                        lcd_print ("Waiting for data...");
                        turn_rx_on ();     /*prepare to receive data through
                                                        the serial port*/
                                          /*wait for a full load of data*/
                         while ((buttons()!=1)&&(count<252))
                          {string[count]=get_char_ar ();    /*receive data*/
                           count++;
                          }
                        turn_rx_off();             /*turn off UART receiver*/
                        lcd_xy (0,10);
                        lcd_print ("Data received");
                        create_record ();              /*create the record*/

                        }                               /*end if buttons==2*/
                    }                                /*end name loose record*/
```

University of Cape Town

```
 if (buttons()==512) delete_record ();          /*delete the current record*/

 if (buttons()==1024) routemenu ();     /*go into the route management menu*/

 if (buttons()==256) menu7();                        /*select data file menu*/

 if (buttons()==4096)            /*create a record and automaticaly name it*/
                     {lcd_clear_text ();
                      lcd_xy (3,0);
                      lcd_print ("Make a record");
                      lcd_xy (0,2);
                      lcd_print ("Your record name is:");
                      lcd_xy (0,3);

                      while (buttons()==4096);
     /*The record name is the time and date in the order YY/MM/DD-HH:mm:SS*/

                     clk_string (1);                  /*get the date string*/
                     for (count=0;count<8;count++)
                                    /*pack date into record name string*/
                       string[count]=clockstring[count];
                     string[8]='-';
                     clk_string(0);                   /*get the date string*/
                     for (count=0;count<8;count++)
                              /*pack time string into record name string*/
                       string[count+9]=clockstring[count];

             /*display the name and clear out the rest of the name string*/
                     for (count=0;count<252;count++)
                                {if (count>16) string [count]=0;
                                 lcd_put_char (string[count]);
                                }

                     create_record ();                /*create the record*/

                     while (buttons()!=1);     /*until quit button pressed*/
                    }                             /*auto name loose record*/
}

/*=====================================================================*/
void menu7 ()
/*select a data record and make it active*/

{unsigned int count,temp,keypad=0;
 lcd_clear_text ();
 lcd_xy (2,0);
 lcd_print ("SELECT DATA FILE");
 disp_soft (1,soft6);                                        /*quit hotkey*/
 disp_soft (2,soft3);                                         /*set hotkey*/
 disp_soft (3,soft7);                                  /*down arrow hotkey*/
 disp_soft (4,soft8);                                    /*up arrow hotkey*/

 while (buttons()==256);   /*wait for user to remove finger from '2' button*/
 delay (250);                                          /*software debounce*/
    get_fat_start();/*start displaying record names from the top of the FAT*/
    count=fat_start;
                                  /*search for the first used FAT entry*/
    while ((used_fat_entry(count)==0)&&(count<=fat_end)) count++;
                    /*if we hit the end of the FAT then the flash is empty*/
    if (count==fat_end) count=fat_start;

      while (1)
      { display_fat_entry (count);                 /*show the record's name*/
        keypad=0;
```

University of Cape Town

```
        while ((keypad<1)||(keypad>8)) keypad=buttons();
        if (keypad==1) break;                                   /*quit button*/
        if (keypad==2)                                           /*set button*/
                      {                 /*make entry shown on screen active*/
                       curr_fat_entry=count;
                       active_file_selected=1;/*we now have an active file*/
                       break;
                      }
        if ((keypad==4)&&(count>fat_start))       /*show previous FAT entry*/
                     {temp=count;
                            /*search for previous FAT entry if any exists*/
                      do count--;
                      while ((used_fat_entry(count)==0)&&(count>fat_start));
                                          /*if no previous entry exists*/
                      if (used_fat_entry(count)==0) count=temp;
                     }

        if ((keypad==8)&&(count<(fat_end-1)))         /*show next FAT entry*/
                    {temp=count;
                     do count++;
                            /*search for previous FAT entry if any exists*/
                     while ((used_fat_entry(count)==0)&&(count<(fat_end-1)));
                                       /*if there are no more FAT entries*/
                     if (used_fat_entry(count)==0) count=temp;
                    }
        delay (250);
      }                                                          /*end while 1*/
}                                                       /*end select data record*/

/*=========================================================================*/
void routemenu ()
/*route manager menu*/

{lcd_clear_text ();
 lcd_xy (4,0);
 lcd_print ("Route Manager");
 lcd_xy (0,2);
 lcd_print ("1. load route from PC");
 lcd_xy (0,3);
 lcd_print ("2. Upload route data");
 lcd_xy (0,4);
 lcd_print ("3. Select route");
 lcd_xy (0,5);
 lcd_print ("4. Delete route");
 lcd_xy (0,6);
 lcd_print ("5. Walk the route");

 waitforbutton (2048,1024);                          /*wait for user keypress*/
 if (buttons()==128) loadroute ();                      /*load route from PC*/
 if (buttons()==256) uploadroute ();                 /*upload entire route to PC*/
 if (buttons()==512) selectroute ();                    /*select an active route*/
 if (buttons()==1024) deleteroute ();                          /*delete a route*/
 if (buttons()==2048) walkroute ();                /*walk the user through a route*/

}

/*=========================================================================*/
void loadroute()
/*Load a route from the PC. The PC sends the route name as well as all of
  the record names which make up the route*/

{int count,empty_route,route_address,pos,number_records,record_number;
 long int record_address;

 lcd_clear_text ();      /*load route from PC*/
```

University of Cape Town

```
lcd_xy (2,0);
lcd_print ("Load route from PC");
lcd_xy (0,2);
lcd_print ("To load a route from");
lcd_xy (0,3);
lcd_print ("a PC attach the RS232");
lcd_xy (0,4);
lcd_print ("link and press 'SET'");
lcd_xy (0,5);
lcd_print ("on this unit and");
lcd_xy (0,6);
lcd_print ("'load'on the PC.");
lcd_xy (0,8);
lcd_print ("If you do not wish to");
lcd_xy (0,9);
lcd_print ("load from the PC hit");
lcd_xy (0,10);
lcd_print ("'QUIT' on this unit.");
disp_soft (2,soft3);
while (buttons()==128);                    /*wait for user to release button*/
while (buttons()<1);
if (buttons()==2)                                                /*set button*/
 {disp_soft (1,soft0);                                 /*clear hotkey legends*/
  disp_soft (2,soft0);
  lcd_clear_text ();
  get_fat_start();
  for (count=1;count<(fat_start-1);count+=2) /*search for blank route spot*/
    {lba1=count;
     lba2=0;
     lba3=0;
     setup_lba ();
     empty_route=1;
     read_sector ();            /*read the first sector in the route entry*/
                               /*check to see if that route entry is vacant*/
     for (pos=0;pos<256;pos++) if (buffer[pos]!=0) empty_route=0;
     if (empty_route==1) break;
    }
    if (empty_route==0) {lcd_clear_text();             /*if no free routes*/
                         lcd_xy (2,7);
                         lcd_print ("No free routes!");
                         disp_soft (1,soft6);
                         while (buttons()!=1);
                         return;
                        }

    route_address=count;

    lcd_xy (1,8);
    lcd_print ("Waiting for data...");
    turn_rx_on ();                                  /*turn UART receiver on*/
                                                 /*get route name and length*/
    for (count=0;count<12;count++) buffer[count]=get_char_ar ();
    number_records=buffer[10];              /*how many records in the route*/
    pos=12;
    turn_rx_off();                                 /*turn UART receiver off*/
    lba1=route_address;          /*write the information to the route entry*/
    lba2=0;
    lba3=0;
    setup_lba ();
    write_sector ();
    out_char_ar ('S'); /*tell host PC that unit is ready for next transfer*/
                                    /*receive the full number of records*/
    for (record_number=0;record_number<number_records;record_number++)
      {if (record_number==122)/*122 records in first sector of route entry*/
                              {route_address++;
```

University of Cape Town

```
                                pos=0;
                               }
        turn_rx_on ();                  /*receive a record name through UART*/
        for (count=0;count<252;count++) string[count]=get_char_ar ();
        turn_rx_off();
                            /*create a record and get its FAT entry address*/
        record_address=create_record ();
        lba1=route_address;  /*update route entry to include the new record*/
        lba2=0;
        lba3=0;
        setup_lba ();
        read_sector ();
                    /*write the pointers to the record into the route entry*/
        buffer[pos]=(char)(record_address&0x000000FF);
        buffer[pos+1]=(char)((record_address/256)&0x000000FF);
        pos+=2;
        setup_lba ();
        write_sector();             /*write the route entry back onto flash*/
        out_char_ar ('G');                    /*signal host for next record*/
       }
  }                                                          /*if buttons==2*/
}                                                               /*load route*/

/*=========================================================================*/
void uploadroute ()
/*upload a complete route to the host PC*/

{long int temp_fat_entry;
 int num_records;
 temp_fat_entry=curr_fat_entry; /*back up curr_fat_entry else we clobber it*/
 if (curr_route_entry==0)                  /*check that a route is selected*/
                         {no_active_route();
                          return;
                          }
 curr_route_pos=0;
 do
 {lba2=0;
  lba3=0;
  if (curr_route_pos<123)
         /*if the record that is being sent is in first route entry sector*/
                {lba1=curr_route_entry;
                 setup_lba ();
                 read_sector ();
                  /*set up curr_fat_entry to point to next file to be sent*/

curr_fat_entry=buffer[(2*curr_route_pos)+12]+256*buffer[(2*curr_route_pos)+13];
                }
   else
      /*if the record that is being sent is in the second route entry sector*/
       {lba1=curr_route_entry+1;
        setup_lba ();
        read_sector ();
                  /*set up curr_fat_entry to point to next file to be sent*/
        curr_fat_entry=buffer[(2*(curr_route_pos-
123))]+256*buffer[(2*(curr_route_pos-123))+1];
       }
  upload_file ();                                          /*send the file*/
  curr_route_pos++;                  /*move to the next record in the route*/
 }
 while (curr_route_pos<buffer[10]);
 curr_fat_entry=temp_fat_entry;    /*restore the current fat entry variable*/
}

/*=========================================================================*/
void selectroute ()
```

University of Cape Town

```
/*allows the user to view all available records and make one active*/

{unsigned int count,temp,keypad=0;
 lcd_clear_text ();
 lcd_xy (2,0);
 lcd_print ("SELECT ROUTE");
 disp_soft (1,soft6);                                        /*quit hotkey*/
 disp_soft (2,soft3);                                         /*set hotkey*/
 disp_soft (3,soft7);                                      /*up arrow hotkey*/
 disp_soft (4,soft8);                                    /*down arrow hotkey*/

 while (buttons()==512);   /*wait for user to remove finger from '3' button*/
 delay (250);                                             /*software debounce*/

 get_fat_start();
 count=1;

 while (1)                                    /*search through route entry table*/
   {if ((used_route_entry(count)==0)&&(count<(fat_start-1))) count+=2;
    display_route_entry (count);
    keypad=0;
    while ((keypad<1)||(keypad>8)) keypad=buttons();
    if (keypad==1) break;                                        /*quit button*/
    if (keypad==2)                                               /*set button*/
                  {curr_route_entry=count;          /*set up the new route*/
                   curr_route_pos=0;                /*set position to zero*/
                   walking_route=0;                /*not walking route yet*/
                   break;
                  }
    if ((keypad==4)&&(count>2))                                  /*down arrow*/
                {temp=count;
                 do count-=2;   /*search for next route entry if one exists*/
                 while ((used_route_entry(count)==0)&&(count>2));
                 if ((count==1)&&(used_route_entry(count)==0)) count=temp;
                }
    if ((keypad==8)&&(count<(fat_start-1)))                       /*up arrow*/
       {temp=count;
        do count+=2;        /*search for previous route entry if one exists*/
        while ((used_route_entry(count)==0)&&(count<(fat_start-1)));
        if ((count>=(fat_start-1))&&(used_route_entry(count)==0)) count=temp;
       }
        delay (250);
   }
}

/*=========================================================================*/
void deleteroute ()
/*Delete a route. Forst the individual records are deleted and then the route
  entry is erased.*/

{int pos,count,number_records1,number_records2=0;
 lcd_clear_text ();
 lcd_xy (0,0);
 if (curr_route_entry==0) {no_active_route();   /*check for an active route*/
                           return;}
   else {lcd_print ("Deleting route...");
         lba1=curr_route_entry;
         lba2=0;
         lba3=0;
         setup_lba ();
         read_sector ();
         number_records1=buffer[10];  /*determine how many records in route*/
                        /*if there is more than 1 sector in the route entry
                           determine how many records are in each sector*/
         if (number_records1>122) {number_records2=number_records1-122;
```

University of Cape Town

```
                                    number_records1=122;
                                   }
                                                    /*find where the records are*/
        for (pos=0;pos<244;pos++) string[pos]=buffer[(pos+12)];
                                        /*remove the records in first sector*/
        for (count=0;count<(2*number_records1);count+=2)
            {lba1=string[count];          /*set up the record to be removed*/
             lba2=string[count+1];
             lba3=0;
               /*check to see if the record is active. If so deactivate it*/
             if
(((char)(curr_fat_entry&0x00FF)==lba1)&&(char)(((curr_fat_entry/256)&0x00FF)==lb
a2)) active_file_selected=0;
             setup_lba ();
             buffer [3]='?';           /*mark the record's data as invalid*/
             buffer [4]='?';                    /*mark the record as blank*/
             write_sector();
            }
                                  /*read the second sector of the route entry*/
        lba1=curr_route_entry+1;
        lba2=0;
        lba3=0;
        setup_lba ();
        read_sector ();
                                   /*remove the records in the second sector*/
                                                /*find where the records are*/
        for (pos=0;pos<256;pos++) string[pos]=buffer[pos];
                                                          /*remove the records*/
        for (count=0;count<(2*number_records2);count+=2)
            {lba1=string[count];              /*set up the record's address*/
             lba2=string[count+1];
             lba3=0;
                       /*if file to be removed is active then deactivate it*/
             if
(((char)(curr_fat_entry&0x00FF)==lba1)&&(char)(((curr_fat_entry/256)&0x00FF)==lb
a2)) active_file_selected=0;
             setup_lba ();
             buffer [3]='?';                  /*mark record data as invalid*/
             buffer [4]='?';                              /*delete record*/
             write_sector();
            }

        lba1=curr_route_entry;                /*now remove the route entry*/
        lba2=0;
        lba3=0;
        setup_lba ();
                          /*clear out the whole first sector of route entry*/
        for (pos=0;pos<256;pos++) buffer[pos]=0;
        write_sector ();
        lba1=curr_route_entry+1;
        lba2=0;
        lba3=0;
        setup_lba ();
        write_sector ();          /*clear out second sector of route entry*/
        curr_route_entry=0;                              /*no active route*/
        curr_route_pos=0;                  /*not in the middle of any route*/
        walking_route=0;                           /*not walking any route*/
       }
}

/*=========================================================================*/
void walkroute ()
/*Walk the user through the route. There is an option to start walking a
  route, stop walking a route in the middle or resume a paused route.*/
```

University of Cape Town

```
{lcd_clear_text ();
 lcd_xy (3,0);
 if (curr_route_entry==0) {no_active_route();      /*check for active route*/
                           return;
                          }
 lcd_print ("Walk the Route");
 lcd_xy (0,2);
 lcd_print ("1. Start route");
 lcd_xy (0,3);
 lcd_print ("2. Resume route");
 lcd_xy (0,4);
 lcd_print ("3. Stop route");

 waitforbutton (512,2048);                        /*wait for user keypress*/

 if (buttons()==128) curr_route_pos=0;     /*start route from position zero*/
 if (buttons()==256) curr_route_pos--;              /*resume a paused route*/

 if ((buttons()==256)||(buttons()==128))    /*if walking is to (re)commence*/
                     {walking_route=1;       /*set flag to indicate walking*/
                      display_next_machine ();                /*prompt user*/
                      state=1;                        /*monitor time domain*/
                      lcd_clear_text();
                      curs_x=64;
                      lcd_clear_graph();             /*obliterate old reading*/
                      for (count=0;count<4096;count++) screen_buf[count]=0;
                      for (count=0;count<128;count++) grapharray[count]=0;
                      update_cursor();
                     }                              /*buttons==256||buttons==128*/

 if (buttons()==512) {curr_route_pos=0;          /*stop route button pressed*/
                      walking_route=0;
                     }

}
/*=============================================================================*/
void display_next_machine ()

{int fat_address;
 lba2=0;
 lba3=0;
       /*if the route entry for this machine is in first route entry sector*/
 if (curr_route_pos<123)
    {lba1=curr_route_entry;
     setup_lba ();
     read_sector ();
                                          /*set up FAT address of record*/

fat_address=buffer[(2*curr_route_pos)+12]+256*buffer[(2*curr_route_pos)+13];
    }
 else                                     /*route entry in second sector*/
    {lba1=curr_route_entry+1;
     setup_lba ();
     read_sector ();
     fat_address=buffer[(2*(curr_route_pos-123))]+256*buffer[(2*(curr_route_pos-
123))+1];
    }
 lcd_clear_text ();
 lcd_clear_graph ();
 lcd_xy (1,0);
        /*check if the last reading taken was the last record in the route*/
 if (curr_route_pos<buffer[10]) curr_route_pos++;/*if not go to next record*/
 else {lcd_print ("Route Complete!");
       delay (3000000);
       state=2;
```

University of Cape Town

```
      return;
     }
 lcd_print ("Go to this machine:");    /*prompt user with next machine name*/
 display_fat_entry (fat_address);
 lcd_xy (0,15);
 lcd_print ("and press ready.");
 route_active_file=fat_address;         /*set up record to be recorded into*/
 while (buttons()!=2048);
 while (buttons()==2048);
 delay (1000);
 lcd_clear_text();
 update_cursor ();
}

/*=============================================================================*/
void no_active_route()
/*Show the user an error message telling that no route is active*/

{lcd_clear_text();
 lcd_xy (0,0);
 lcd_print ("No active route!");
 delay (3000000);
 return;
}

/*=============================================================================*/
void waitforbutton (unsigned int maximum,unsigned int waitbutton)
/*waits for the quit button or any other button between 128 and
  minimum inclusive. used to detect when a menu option has been chosen.*/

{unsigned int keypad=0;
 while (buttons()==waitbutton);
 delay (500);

 while ((keypad!=1)&&(!((keypad>=128)&&(keypad<=maximum))))
        keypad=buttons();
}


/*=============================================================================*/
void clk_init ()
/*boot up initialization routine for real time clock. If the clock is halted,
  start it. Set the clock into 24hr mode and clear the write protect bit so
  that it may have data stored in it.*/

{clk_com=0x0081;          /*check the clock halt flag. If it is set, clear it*/
   get_clk ();
   delay(1000);
   if ((clk_dat&0x0080)==0x0080) clk_dat=clk_dat&0x007F;
   clk_com=0x0080;
   put_clk();
   clk_com=0x0085;                        /*set the clock into 24 hour mode*/
   get_clk ();
   delay(1000);
   if ((clk_dat&0x0080)==0x0080) clk_dat=clk_dat&0x007F;
   clk_com=0x0084;
   put_clk ();
   clk_com=0x008E;                            /*clear the write protect bit*/
   clk_dat=0;
   put_clk();
}

/*=============================================================================*/
void setclk (int operation)
/*Allows the user to set the time or date in the real time clock*/
```

University of Cape Town

```
{int curs_pos;
 unsigned int keypad;
 lcd_clear_text ();
 lcd_xy (0,12);
 lcd_print ("Press quit to exit");
 disp_soft (3,soft3);                                          /*quit softkey*/
 lcd_xy (0,13);
 if (operation==1)                                               /*set time*/
   {lcd_print ("Press set to set time");
                  /*display the time at the bottom of the screen*/
     while ((buttons()!=1)&&(buttons()!=4)) {lcd_xy (7,7);
                                             show_clk (0);
                                            }
     if (buttons()==1) return;                                  /*quit button*/
     lcd_xy (0,13);
     lcd_print ("arrows to move cursor");
     lcd_xy (0,14);
     lcd_print ("number keys set time");

     disp_soft (3,soft0);                                  /*display quit hotkey*/
     disp_soft (3,soft4);                                /*display arrow hotkeys*/
     disp_soft (4,soft5);
     curs_pos=0;                             /*first time field (tens of hours)*/
     while (buttons()!=1)                                  /*menu button to quit*/
     {lcd_xy (7,7);
      cput (0x009F);                                       /*cursor on and blink*/
      cput (0x00A2);                                             /*3 line cursor*/
      dput (0x0007+curs_pos);          /*position cursor at field to be set*/
      dput (0x0007);
      cput (0x0021);
      show_clk (0);                                      /*show the current time*/
      keypad=buttons();
      if (keypad==0) continue;
      if ((keypad==4)&&(curs_pos>0))                                 /*left arrow*/
                 {curs_pos--;   /*move cursor left. Skip seperating colons*/
                  if ((curs_pos==2)||(curs_pos==5)) curs_pos--;
                  while (buttons()==4);
                  delay (100);
                 }
      if ((keypad==8)&&(curs_pos<7))                               /*right arrow*/
                 {curs_pos++;  /*move cursor right. Skip seperating colons*/
                  if ((curs_pos==2)||(curs_pos==5)) curs_pos++;
                  while (buttons()==8);
                  delay (100);
                 }
      if ((keypad>=64)&&(keypad<=32768))     /*the procedure to change time*/
       {if (curs_pos==0)
                          {clk_com=0x0085;             /*set tens of hours*/
                     get_clk();
                           delay(1000);
                           clk_dat=clk_dat&0x00CF;
                                                  /*pack data into clock*/
                           if (keypad==128) clk_dat=clk_dat|0x0010;
                                    /*pack data into clock if it is legal*/
                           if ((keypad==256)&&((clk_dat&0x000F)<4))
                                                  clk_dat=clk_dat|0x0020;
                           clk_com=0x0084;     /*command to write to clock*/
                     put_clk();
                    }
                /*set hours (units) or minutes (units) or seconds (units)*/
        if ((curs_pos==1)||(curs_pos==4)||(curs_pos==7))
                    {if (curs_pos==1) clk_com=0x0085;
                     if (curs_pos==4) clk_com=0x0083;
                     if (curs_pos==7) clk_com=0x0081;
```

University of Cape Town

```
                    get_clk();
                          delay(1000);
                    clk_dat=clk_dat&0x00F0;
                                    /*decode keypad output into numbers*/
                          clk_dat+=(intlog(keypad)-6);
                          clk_com--;   /*clear the read bit to write data*/
                    put_clk();
                   }
                                    /*set tens of minutes or tens of seconds*/
      if (((curs_pos==3)||(curs_pos==6))&&((keypad>=64)&&(keypad<=2048)))
                   {if (curs_pos==3) clk_com=0x0083;
                    if (curs_pos==6) clk_com=0x0081;
                    get_clk();
                          delay(1000);
                    clk_dat=clk_dat&0x008F;
        /*decode keypad output into numbers and shift it left four places*/
                    clk_dat+=(intlog(keypad)-6)*16;
                    clk_com--;
                    put_clk();
                   }
     }
    }
    cput (0x009C);                                        /*turn cursor off*/
  }                                                    /*end if operation==1*/

if (operation==2)                                              /*set date*/
  {lcd_print ("Press set to set date");
    while ((buttons()!=1)&&(buttons()!=4)) {lcd_xy (6,7);      /*show date*/
                                  show_clk (1);}
    if (buttons()==1) return;                                /*quit button*/
    lcd_xy (0,13);
    lcd_print ("arrows move cursor");
    lcd_xy (0,14);
    lcd_print ("number keys set date");
    disp_soft (3,soft0);                              /*quit and arrow hotkeys*/
    disp_soft (3,soft4);
    disp_soft (4,soft5);
    curs_pos=0;
    while (buttons()!=1)                                /*menu button to quit*/
    {lcd_xy (6,7);
     cput (0x009F);                                     /*cursor on and blink*/
     cput (0x00A2);                                          /*3 line cursor*/
     dput (0x0008+curs_pos);              /*position cursor on current field*/
     dput (0x0007);
     cput (0x0021);
     show_clk (1);
     keypad=buttons();
     if (keypad==0) continue;
     if ((keypad==4)&&(curs_pos>0))                              /*left arrow*/
                   {curs_pos--;       /*move cursor left, skip seperators*/
                    if ((curs_pos==2)||(curs_pos==5)) curs_pos--;
                    while (buttons()==4);
                    delay (100);
                   }
     if ((keypad==8)&&(curs_pos<7))                             /*right arrow*/
                   {curs_pos++;      /*move cursor right, skip seperators*/
                    if ((curs_pos==2)||(curs_pos==5)) curs_pos++;
                    while (buttons()==8);
                    delay (100);
                   }
      if ((keypad>=64)&&(keypad<=32768))    /*The procedure to change date*/
      {if (curs_pos==0)
                         {clk_com=0x008D;             /*set tens of years*/
                    get_clk();
                          delay(1000);
```

University of Cape Town

```
                    clk_dat=clk_dat&0x000F;
                /*decode the keypad into numbers and shift left four places*/
                    clk_dat+=(intlog(keypad)-6)*16;
                        clk_com=0x008C;     /*command to write to clock*/
                    put_clk();
                   }
        if ((curs_pos==1)||(curs_pos==4)||(curs_pos==7))
                                            /*set years months and days*/
                         {if (curs_pos==1) clk_com=0x008D;          /*year*/
                          if (curs_pos==4) clk_com=0x0089;         /*month*/
                          if (curs_pos==7) clk_com=0x0087;          /*date*/
                    get_clk();
                        delay(1000);
                              /*check for legality of months and days*/
                    if
((curs_pos==4)&&((clk_dat&0x0010)==0x0010)&&(keypad>256)) continue;
                    if
((curs_pos==7)&&((clk_dat&0x0030)==0x0030)&&(keypad>128)) continue;
                    clk_dat=clk_dat&0x00F0;
                        clk_dat+=(intlog(keypad)-6);    /*decode keypad*/
                    clk_com--;
                    put_clk();
                   }

        if ((curs_pos==6)&&(keypad>=64)&&(keypad<=512))  /*set 10s of date*/
                        {clk_com=0x0087;
                         get_clk();
                         delay(1000);
                         clk_dat=clk_dat&0x008F;
                         clk_dat+=(intlog(keypad)-6)*16;  /*decode keypad*/
                         if (((clk_dat&0x000F)>1)&&(keypad>=512)) continue;
                         clk_com--;              /*command to write to clock*/
                         put_clk();
                        }
                                                  /*set 10s of months*/
       if ((curs_pos==3)&&(keypad>=64)&&(keypad<=128))
                    {clk_com=0x0089;
                     get_clk ();
                        delay(1000);
                     clk_dat=clk_dat&0x00EF;
                                        /*decode and shift keypad code*/
                        if (keypad==128) clk_dat+=16;
                     if ((clk_dat&0x000F)>=3) continue;
                     clk_com--;
                     put_clk();
                    }
      }
     }
     cput (0x009C);                                 /*turn cursor off*/
   }                                              /*end if operation==2*/
}

/*=========================================================================*/
void show_clk (int detail)
/*displays the real time clock on the screen at the current text position.
  detail==0->time only detail==1->date only detail==2->both time and date*/

{if ((detail==0)||(detail==2))                      /*put time on screen*/
   {clk_string(0);
    lcd_print (clockstring);
    lcd_print ("  ");
   }

 if ((detail==1)||(detail==2))                      /*put date on screen*/
   {lcd_print ("20");
```

University of Cape Town

```
    clk_string(1);
    lcd_print (clockstring);
    lcd_print ("  ");
   }
}

/*=========================================================================*/
void clk_string (int detail)
/*packs the real time clock data into a string called clockstring. The data
 is packed in so that it is ready for display. If detail is zero the time is
 packed into the string, if detail is 1 the date is packed.*/

{int temp,pos=0;
 if (detail==0)                                        /*put time into string*/
   {for (clk_com=133;clk_com>128;clk_com-=2)     /*loop through time digits*/
    {get_clk ();
    delay(1000);
    /*decode clock data into string. Each byte from the clock is two digits*/
     clockstring [pos]=(((clk_dat&0x0070)/16)+0x0030);
     clockstring[pos+1]=((clk_dat&0x000F)+0x0030);
     clockstring[pos+2]=':';                  /*insert seperator into string*/
     pos+=3;
    }
    clockstring[8]=0;              /*insert null terminator for easy display*/
   }

 if (detail==1)                                        /*put date into string*/
   {for (temp=139;temp>134;temp-=2)          /*loop through all date fields*/
    {clk_com=temp;
     if (clk_com==139) clk_com=141;                        /*skip day field*/
     get_clk ();
     delay(1000);
         /*decode clock data into characters. Each clock byte is two digits*/
     clockstring [pos]=(((clk_dat&0x0070)/16)+0x0030);
     clockstring[pos+1]=((clk_dat&0x000F)+0x0030);
     clockstring[pos+2]='/';                              /*insert seperator*/
     pos+=3;
    }
    clockstring[8]=0;              /*insert null terminator for easy display*/
   }
}

/*=========================================================================*/
void zoompan ()
/*This function reads the zoom and pan keys and sets up the zoom and pan
  variables to give correct zoom and position*/

{                                 /*zoom in pressed and not yet fully zoomed in*/
 if ((buttons()==16)&&(number_per_line>1)) number_per_line/=2;
                                 /*zoom out pressed and not yet fully zoomed out*/
 if ((buttons()==32)&&(number_per_line<=16)) number_per_line*=2;
                      /*pan left button pressed so pan 0.25 of the screen left*/
 if (buttons()==64) start_display-=((128*number_per_line)/4);
                           /*some combinations of zooming and panning may cause
                            screen overflow this protects against the problem*/
 if (start_display<0) start_display=0;
                          /*pan right pressed so pan right 0.25 of the screen*/
 if (buttons()==128) start_display+=((128*number_per_line)/4);
                  /*protect against array overflow due to zooming and panning*/
 if (start_display>(4095-128*number_per_line)) start_display=4095-
128*number_per_line;
 if ((start_display+spare+(number_per_line*127))>4095) start_display=0;
}

/*=========================================================================*/
```

University of Cape Town

```
void data_to_graph (int graphtype)
/*convert the 4096 element data array into a 128 element array to show on the
  screen. This involves using the zoom and pan factors, as well as data
  compression*/

{int loop;
   /*for spectrum each pixel represents the maximum point of the data that is
                                                    compressed into that pixel*/
 for (poscount=0;poscount<128;poscount++)
 {max=-1;
  for (spare=0;spare<number_per_line;spare++)
  {if (abs(screen_buf[(start_display+spare+(number_per_line*poscount))])>max)
      {max=screen_buf[(start_display+spare+(number_per_line*poscount))];

           /*we also need to store exactly where the maximum point was. This
       allows the system to give numerical readings with far greater accuracy
                                                   than the screen can represent*/
       maxima[poscount]=start_display+spare+(number_per_line*poscount);
      }
  }
   grapharray[poscount]=max;
 }

          /*The time domain displays use a different method of compression*/
 if ((state==1)||(state==3))
   {for (poscount=0;poscount<128;poscount++)
     /*showing only the first sample looks better than showing the average.*/
    {grapharray[poscount]=screen_buf[start_display+number_per_line*poscount];
    }                                                  /*loop iterated 128 times*/
   }                                                              /*state==1*/
}

/*=============================================================================*/
void spectrum ()
/*This function is responsible for processing raw ADC data into a usable
  spectrum display*/

{if (state==0) lcd_xy (5,0);
    else recall_mods ();      /*modify display slightly for spectrum recall*/

 lcd_print ("Spectrum");
 disp_soft (1,soft1);                              /*display hotkey legends*/
 disp_soft (2,soft2);
 disp_soft (3,soft4);
 disp_soft (4,soft5);
              /*only transform data that has not already been transformed*/
 if ((transformed==0)||(state<4))
  {if ((state==0)||(state==4))
      {transformed=1;                                /*mark data as transformed*/
       window (xreal,ximag,window_type,10);    /*apply a window to the data*/
       fft (xreal,ximag,number_points,nu);                    /*FFT the data*/
       unpack (xreal,ximag,number_points); /*unpacking is a part of the FFT*/
      }
   if (!(state==2))
      {for (pos=0;pos<number_points;pos++)   /*calculate magnitude spectrum*/
         {mag1=xreal[pos];
          mag2=ximag[pos];
          screen_buf[pos]=256*(sqrt(power(mag1,2)+power(mag2,2)));
         }                                                        /*for pos=0*/
      }
  }
  data_to_graph (1);/*convert magnitude spectrum to 128 point display array*/

  lcd_clear_graph();                              /*clear graphics off screen*/
  disp_soft (1,soft1);                                 /*show hotkey legends*/
```

University of Cape Town

```
  disp_soft (2,soft2);
  disp_soft (3,soft4);
  disp_soft (4,soft5);

  lcd_xy (5,7);
  if (overrange==0)                                  /*check for over range data*/
                {lcd_print ("          ");  /*clear old over range message*/
                 lcd_putarray ();              /*display the spectrum graph*/
                 update_cursor ();             /*update the cursor position*/
                 freq_disp(); /*show the frequency and magnitude of cursor*/
                 find_peaks(show_peaks);/*show the harmonics of the cursor*/
                }
     else lcd_print ("Over Range");
}

/*=========================================================================*/
void waveform ()
/*convert and display a time domain on the screen, together with all other
  features, such as time, magnitude etc.*/

{int count=0,loop;
 if (state==1) lcd_xy (5,0);
    else recall_mods ();  /*modify the screen display if recalling waveform*/

 lcd_print ("Waveform");
 if ((buttons()==256)&&(walking_route==0))          /*if save button pressed*/
                                          savedata ();
 if (walking_route==1)                                    /*walking a route*/
      {if (buttons()==256) {savedata ();                      /*save button*/
                            display_next_machine ();
                           }
       if (buttons()==512) display_next_machine ();          /*skip button*/
       if (buttons()==1024) {curr_route_pos-=2;              /*back button*/
                             display_next_machine ();
                            }
      }
           /*load data into screen buffer and convert it to display format*/
 for (count=0;count<4096;count++) screen_buf [count]=xreal [count];
 data_to_graph (1);
 lcd_clear_graph();                                 /*clear graphics display*/
 disp_soft (1,soft1);
 if (state==1)                    /*only display trigger hotkey if monitoring*/
             disp_soft (2,soft9);
 disp_soft (3,soft4);
 disp_soft (4,soft5);
 lcd_xy (5,7);
 if (overrange==0)                                  /*check for over range data*/
                  {lcd_print ("          ");    /*clear over range message*/
                   lcd_putarray ();                        /*print new graph*/
                   update_cursor ();          /*update the cursor position*/
                   time_disp();           /*show cursor time and magnitude*/
                  }
   else lcd_print ("Over Range");
}

/*=========================================================================*/
void recall_mods ()
{unsigned int old_rate;

 lcd_xy (0,1);
 lcd_print (string);
 lcd_print ("  ");
 lcd_print ("20");
 lcd_print (string+9);
 lcd_xy (2,0);
```

University of Cape Town

```
 lcd_print ("Recall ");

  old_rate=samples_per_sec;
  switch (string[18])
        {case 0x41: samples_per_sec=2000; break;
         case 0x43: samples_per_sec=4000; break;
         case 0x47: samples_per_sec=8000; break;
         case 0x4B: samples_per_sec=16000; break;
         case 0x46: samples_per_sec=32000; break;
         case 0x4C: samples_per_sec=48000; break;
        }
  rate_var=string [18]+0xc800;        /*setup sample rate*/
  if (samples_per_sec!=old_rate) change_fs(rate_var);
 }

/*=========================================================================*/

void update_cursor ()
 /*put the cursor on*/
{
 place_cursor (0);
 if ((buttons()==4)&&(curs_x>=1)) curs_x--;
 if ((buttons()==8)&&(curs_x<=126)) curs_x++;

 curs_y=64-grapharray[curs_x];
 place_cursor(1);
}

/*=========================================================================*/

void place_cursor (int optype)
{int count;
 for (count=-2;count<=2;count++)
   {lcd_setpixel ((curs_x+count),(curs_y+2),optype);
    lcd_setpixel ((curs_x+count),(curs_y),optype);
    lcd_setpixel ((curs_x+count),(curs_y-2),optype);
    lcd_setpixel ((curs_x+2),(curs_y+count),optype);
    lcd_setpixel ((curs_x),(curs_y+count),optype);
    lcd_setpixel ((curs_x-2),(curs_y+count),optype);
   }


/* for (count=-2;count<=2;count++)
   lcd_setpixel ((curs_x+count),(curs_y),optype);
 for (count=-2;count<=2;count++)
   lcd_setpixel ((curs_x+count),(curs_y-2),optype);

 lcd_setpixel (curs_x,(curs_y+1),optype);
 lcd_setpixel ((curs_x+2),(curs_y+1),optype);
 lcd_setpixel ((curs_x-2),(curs_y+1),optype);
 lcd_setpixel ((curs_x+2),(curs_y-1),optype);
 lcd_setpixel ((curs_x-2),(curs_y-1),optype);
 lcd_setpixel ((curs_x),(curs_y-1),optype);*/
 if (optype==0)
 /*restore the screen display from where the cursor was.  */
   {data_to_graph (2);
    lcd_putarray ();}

 }

/*=========================================================================*/
char* itoa (unsigned long in_num ,char* input, int len)
{/*converts a number to a string.
 len is the length of the string minus 1 eg arbstring [5] needs len=4
 this function assumes that the input was pre-multiplied by 100.*/
```

University of Cape Town

```
 int i;
 unsigned long temp;

  temp=in_num/10;
  input[(len-1)]=48+in_num-10*temp;

  in_num=temp;
  temp=in_num/10;
  input[(len-2)]=48+in_num-10*temp;

 input[len-3]='.';

 for (i=4;i<=len;i++)
  {in_num=temp;
   temp=in_num/10;
   input[(len-i)]=48+in_num-10*temp;
  }
 input [len]=0;
 return input;
}

/*=============================================================================*/
```

University of Cape Town

## B.2 Ana_math.c

```
/*Written by: Samuel Ginsberg

 This file contains the mathematical functions for the vibration analyzer.
 The functions are as follows:

 fft: This function performs an fft on data passed into it.
 bitrev: This function is the bit reversal used in the fft.
 power: This returns one argument to the power of the other argument.
 intlog: Returns a logarithm base 2 of its input.
 unpack: Performs the post fft unpacking for the real fft.
 window: multiplies the input by a selectable window function.
 find_rms: Calculates the RMS value of the input array.
 crest_factor: Calculates the crest factor of the input array.

==========================================================================*/

void fft (int *xreal, int *ximag, unsigned int N, unsigned int nu);
unsigned int bitrev (unsigned int nu, unsigned int j);
double power (double x, double y);
int intlog (int arg);
void unpack (int *xreal,int *ximag,unsigned int N);
void window (int *xreal, int* ximag, int win_type, int width);
unsigned int find_rms (int *xreal, int *ximag);
unsigned int crest_factor (int *xreal, int *ximag,unsigned int rms);

/*==========================================================================*/

#define number_points 4096                          /*number of fft points*/
#define pi 3.14149265
#include <math.h>                                          /*cos(),sin()*/

/*==========================================================================*/

extern int xreal [2];                /*data array for even numbered samples*/
extern int ximag [2];                 /*data array for odd numbered samples*/
extern int state;                          /*what state the analyzer is in.*/
extern int change_peaks;     /*set if the change peaks button was pressed.*/
extern int old_state;       /*analyzer state before the keyboard interrupt.*/
extern int show_peaks;
        /*zero when peak finder is off, non zero when peak finder is active*/

/*==========================================================================*/

extern void update_cursor ();              /*updates the cursor on the screen*/
extern void freq_disp ();     /*puts the current cursor frequency on screen*/
extern void find_peaks (int present);     /*places circles on harmonic peaks*/
extern void zoompan ();           /*zooms and pans the screen over the data*/
extern unsigned int buttons ();
                          /*This returns the value read in from the keypad.*/
extern void lcd_clear_graph();               /* clear graphics memory of LCD*/
extern void data_to_graph (int graphtype);
                    /*converts a big data array to a small graphable array*/
extern void delay(unsigned long d);       /* delay proportional to "d" value*/

/*==========================================================================*/
/*
The program below implements an FFT algorithm. The algorithm operates
in place. The data vector is only 16 bits wide. The maximum input range
into the FFT is from -25000 to +25000 (a range of 50000. This is because the
number representation only allows a range of -32768 to 32767 and input
numbers get multiplied and added, thus they need room to grow.
```

University of Cape Town

```
It must be noted that the fft is a real fft. This means that there are 8192
data points and only a 4096 point fft. This saves memory space. Because of
this special packing is required for the data goin into the fft and unpacking
is required for the data coming out of the fft.
Pack the data (total 2N points) into the fft with sample element 0 in
xreal[0], sample element 1 in ximag[0]. This means that xreal has
samples 2k k=0,1,2...N-1 and ximag has samples 2k+1 k=0,1,2..N-1
The output needs to be unpacked. The function unpack (..) converts the
values that come out of the fft into the correct real and imaginary parts
of the transform.

The code is an adaptation of the algorithm found in E Oren Brigham's book
called The Fast Fourier Transform. As an aid to understanding the code
presented there I used the book Informal Introduction to Algol 68 by
CH Lindsey and SG Van Der Meulen as well as Programming in Standard
Fortran 77 by Balfour and Marwick.  */

/*=========================================================================*/

void fft (int *xreal, int *ximag, unsigned int N, unsigned int nu)
{unsigned int n2,k,nu1,i,p,keypad;
 float arg,cosine,sine,treal,timag;
 int temp1,temp2,l;

 n2=N/2;
 nu1=nu-1;
 k=0;
 for (l=1;l<=nu;l++)
 {

   /*The next section is not a part of the mathematical transform.
    The reason for its inclusion is that the fft takes many seconds. while
    the user is waiting for fft results they may wish to zoom, pan, quit out
    of the fft, turn the peak finder on or off and move the cursor. This is
    all done inside the fft loop so that the response times are acceptable.*/

  /*is peak finder on or off? has the state been toggled?
    The toggle peak finder function is in an interrupt service routine.*/
  lcd_xy (12,14);
  if (show_peaks==0) lcd_print ("        ");
           else lcd_print ("harmonic");
  if (change_peaks==1) {find_peaks (show_peaks);
                        change_peaks=0;}                    /*change_peaks==1*/


 label:

  /*provision for zooming and panning*/
  keypad=buttons();
  if ((keypad==16)||(keypad==32)||(keypad==64)||(keypad==128))
                     {zoompan();
                      data_to_graph (2);
                      lcd_clear_graph ();
                      update_cursor();
                      find_peaks (show_peaks);
                           }                                /*zoom and pan*/

  if (state==2) return;  /*get to menu quickly*/

  /*if the arrow keys were pressed update the cursor position*/
  if ((keypad==4)||(keypad==8)) {update_cursor();
                                 delay (4000);
                                 freq_disp();
                                 find_peaks (show_peaks);
```

University of Cape Town

```
                                }                                   /*cursor position*/

    /*continue with the fft process.*/
   for (i=1;i<=n2;i++)
   {
    p=2*bitrev (nu,(k/((int)ldexp (1,nu1))));
    arg=((pi*p)/N);
    cosine=(float) cos (arg);
    sine=(float) sin (arg);
    treal=xreal[k+n2]*cosine+ximag[k+n2]*sine;
    timag=ximag[k+n2]*cosine-xreal[k+n2]*sine;
    xreal[k+n2]=(int)(xreal[k]-treal)/2;  /*note that the algorithm has been*/
    ximag[k+n2]=(int)(ximag[k]-timag)/2;  /*modified to allow the data array*/
    xreal[k]=(int)(xreal[k]+treal)/2;              /*to be of integer type.*/
    ximag[k]=(int)(ximag[k]+timag)/2;
    k++;
   }                                                    /*for (i=1;i<=n2;i++)*/
   k+=n2;
   if (k<N) goto label;
   k=0;
   nu1--;
   n2/=2;
  }                                                     /*for (l=1;l<=nu;l++)*/
  for (k=0;k<N;k++)
  {i=bitrev (nu,k);
   if (i>k)
   {temp1=xreal[k];
    temp2=ximag[k];
    xreal[k]=xreal[i];
    ximag[k]=ximag[i];
    xreal[i]=temp1;
    ximag[i]=temp2;
   }                                                            /*if i>k*/
  }                                                     /* for(k=0;k<N;k++)*/
}      /*void fft (int *xreal, int *ximag, unsigned int N, unsigned int nu)*/

/*=========================================================================*/
unsigned int bitrev (unsigned int nu, unsigned int j)
{/*bitrev is used to bitreverse a 12 bit number. The bit reversal is needed
   by the fft algorithm. The function works by shifting the input number to
   the right 12 times and testing the least significant bit. The result of
   the test is put into the output number, which is shifted right every time
   the input is shifted left.*/

unsigned int j2,ibitr,count;
 ibitr=0;
 for (count=1;count<=12;count++)
  {j2=j/2;
   ibitr=((ibitr*2)+(j-(2*j2)));
   j=j2;}                                   /*for (count=1;count<=12;count++)*/
  return (ibitr);
 }                  /*unsigned int bitrev (unsigned int nu, unsigned int j)*/

/*=========================================================================*/

double power (double x, double y)
/*returns x to the power of y. The function uses repeated multiplication*/
{int count=0;
 double temp;
 temp=x;
 if (y==0) return (1);
 for (count=0;count<(y-1);count++)
  x=x*temp;
 return (x);
}                                       /*double power (double x, double y)*/
```

University of Cape Town

```
/*=========================================================================*/

int intlog (int arg)
{/*This function reurns the log to the base 2 of the input.*/
 switch (arg)
     {case 32768: return (15);
      case 16384: return (14);
      case 8192: return (13);
      case 4096: return (12);
      case 2048: return (11);
      case 1024: return (10);
      case 512: return (9);
      case 256: return (8);
      case 128: return (7);
      case 64: return (6);
      case 32: return (5);
      case 16: return (4);
      case 8: return (3);
      case 4: return (2);
      case 2: return (1);
      case 1: return (0);
     }                                                    /*switch (arg)*/
 return (0);
}                                                  /*int intlog (int arg)*/

/*=========================================================================*/

void unpack (int *xreal,int *ximag,unsigned int N)
{/*The fft function returns its data in scrambled form. This function
   unscrambles the fft result into a usable format. The algorithm was
   taken from E Oren Brigham's book "The Fast Fourier Transform".*/

 int Rn,RNn,In,INn,n;
 float arg,cos1,cos2,sin1,sin2;

 for (n=1;n<(N/2);n++)
 {Rn=xreal[n];
  RNn=xreal[(N-n)];
  In=ximag[n];
  INn=ximag[(N-n)];
  arg=pi*n/N;
  cos1=(float) cos (arg);
  sin1=(float) sin (arg);
  arg=pi*(N-n)/N;
  cos2=(float) cos (arg);
  sin2=(float) sin (arg);

 xreal [n]=(int) (((Rn+RNn)/2)  +  cos1*(In+INn)/2  -  sin1*(Rn-RNn)/2);
 xreal [(N-n)]=(int) ((RNn+Rn)/2  +  cos2*(INn+In)/2  -  sin2*(RNn-Rn)/2);
 ximag [n]=(int) ((In-INn)/2  -  sin1*(In+INn)/2  -  cos1*(Rn-RNn)/2);
 ximag [(N-n)]=(int) ((INn-In)/2  -  sin2*(INn+In)/2  - cos2*(RNn-Rn)/2);
 }                                                  /*for (n=1;n<(N/2);n++)*/
 xreal [0]=0;                                              /*Correct DC bin*/
}                        /*void unpack (int *xreal,int *ximag,unsigned int N)*/
/*=========================================================================*/

void window (int *xreal, int* ximag, int win_type, int width)
{/*This function takes in xreal and ximag packed as for the real fft. The
   function returns xreal and ximag, packed the same way, but windowed. The
   variable win_type determines the window type applied. type=0: rectangular.
   type=1: triangle. type=2: Hamming. type=3: Hanning.
   The variable width gives the single sided window rolloff width as a
   percentage. e.g. width=1 with number_samples=4096 will window
   2*4096*1/100=82 samples on each end of the data array.*/
```

University of Cape Town

```
int count,number;

for (count=0;count<4096;count++)  /*First apply some prescaling for the fft*/
 {ximag[count]/=16;
  xreal[count]/=16;
 }                                          /*for (count=0;count<4096;count++)*/

if (win_type==0) return;                 /*rectangular window requires no action*/

number=(int)number_points*width/50; /*calculate how many samples are affected
                                                    by the windowing operation*/

if (win_type==1)                                                /*triangular window*/
 {for (count=0;count<number;count+=2)
  {xreal[count]=(xreal[count]*count)/number;   /*linear rolloff at the ends*/
   ximag[count]=(ximag[count]*(count+1))/number;
   xreal[number_points-1-count]=(xreal[number_points-1-count]*(count+1))/number;
   ximag[number_points-1-count]=(ximag[number_points-1-count]*count)/number;
  }                                       /*for (count=0;count<number;count+=2)*/
 return;
 }                                                                  /*win_type==1*/

if (win_type==2)                                                  /*Hamming window*/
                                      /*The Hamming window shape is specified as
                                         x(n)=0.54-0.46*Cos (2*pi*n/(N-1)) */
 {for (count=0;count<number;count+=2)
  {xreal[count]=(xreal[count]*(0.54-0.46*cos(6.28319*count/(number-1))));
   ximag[count]=(ximag[count]*(0.54-0.46*cos(6.28319*(count+1)/(number-1))));
   xreal[number_points-1-count]=(xreal[number_points-1-count]*(0.54-
0.46*cos(6.28319*(count+1)/(number-1))));
   ximag[number_points-1-count]=(ximag[number_points-1-count]*(0.54-
0.46*cos(6.28319*(count)/(number-1))));
  }                                       /*for (count=0;count<number;count+=2)*/
  return;
 }                                                                  /*win_type==2*/

if (win_type==3)                                                 /*Hanning window*/
                                      /*The Hanning window is specified as
                                         x(n)=0.5*(1-Cos(2*pi*n/(N-1))) */
 {for (count=0;count<number;count+=2)
  {xreal[count]=(xreal[count]*.5*(1-cos(6.28319*count/(number-1))));
   ximag[count]=(ximag[count]*.5*(1-cos(6.28319*(count+1)/(number-1))));
   xreal[number_points-1-count]=(xreal[number_points-1-count]*.5*(1-
cos(6.28319*(count+1)/(number-1))));
   ximag[number_points-1-count]=(ximag[number_points-1-count]*.5*(1-
cos(6.28319*count/(number-1))));
  }                                       /*for (count=0;count<number;count+=2)*/
  return;
 }                                                                  /*win_type==3*/
}          /*void window (int *xreal, int* ximag, int win_type, int width)*/

/*=========================================================================*/

unsigned int find_rms (int *xreal, int *ximag)
{/*Calculate the RMS value of the input arrays. The calculation works by
   squaring each value, averaging the squares and taking the square root
   of that average.*/

 int count;
 double total=0;
 for (count=0;count<(number_points);count++)
  {total=total+power(xreal[count],2);
   total=total+power(ximag[count],2);
  }                              /*for (count=0;count<(number_points);count++)*/
```

University of Cape Town

```
 total=total/(2*number_points);
 return (int)(sqrt(total));
}                          /*unsigned int find_rms (int *xreal, int *ximag)*/

/*=============================================================================*/

unsigned int crest_factor (int *xreal, int *ximag,unsigned int rms)
{/*Calculate the Crest factor of the input array. Crest factor is defined
   as the ratio of the peak to the RMS.
   The function works by looking through the input array for the peak and
   dividing that peak by the RMS value obtained from the find_rms function.*/

 int count;
 float peak=0;

 for (count=0;count<(number_points);count++)
 {if (abs(xreal[count])>peak) peak=abs(xreal[count]);
  if (abs(ximag[count])>peak) peak=abs(ximag[count]);
 }                           /*for (count=0;count<(number_points);count++)*/
 return (int)(100*(peak/rms));
}     /*unsigned int crest_factor (int *xreal, int *ximag,unsigned int rms)*/

/*=============================================================================*/
```

University of Cape Town

## B.3 Ana_lcd.c

```
/*
Partially written by Samuel Ginsberg. See the notice below for full
credits.

This file contains all of the LCD drivers for the vibration analyzer.
The bottom of the file contains all the lowest level code, i.e. code
that manipulates hardware lines etc. The closer the beginning of the file
that a function is, the higher the level of abstraction it is.

Program to control a T6963C-based 240x64 pixel LCD display
partially Written by John P. Beale May 3-4, 1997  beale@best.com
Based on information from Steve Lawther,
"Writing Software for T6963C based Graphic LCDs", 1997 which is at
http://ourworld.compuserve.com/homepages/steve_lawther/t6963c.pdf
and the Toshiba T6963C data sheet, also on Steve's WWW page
and info at: http://www.citilink.com/~jsampson/lcdindex.htm
             http://www.cs.colostate.edu/~hirsch/LCD.html
             http://www.hantronix.com/
==========================================================================*/


/*====== Definitions concerning LCD internal memory and graphing==========*/

#define s_scale 512                  /*scale constant for data graph display*/
#define G_BASE 0x0200                     /* base address of graphics memory*/
#define T_BASE 0x0000                      /* base address of text memory   */
#define BYTES_PER_ROW 30                 /* how many bytes per row on screen*/

/*==========================================================================*/

extern int portvar;          /*This variable holds the value presented on the
                                                        port at address 0x0001.*/
int myvar;                                /*global variable to communicate with
                                                    hardware via assembler code*/
extern int grapharray [128];               /*array which holds data to graph*/
extern int text_x, text_y;       /*indicate current text position on screen*/

/*Function declarations. These are arranged in order of descending
  abstraction, ie the lower a function is in the list the closer it is
  to the hardware.*/

void lcd_putarray ();    /*plot a graph of the data in the grapharray array*/
void lcd_print(char *string);             /*send string of characters to LCD*/
void lcd_clear_graph();                       /*clear graphics memory of LCD*/
void lcd_clear_text();                          /*clear text memory of LCD*/
void lcd_xy(int x, int y);    /*set memory pointer to (x,y) position (text)*/
void lcd_put_char (char data);  /*put a single text character on the screen*/
void lcd_setpixel(int column, int row,int optype);
                            /* set or clear a single pixel in 128x128 array*/
void lcd_setup();           /*make sure control lines are at correct levels*/
void lcd_init();                 /*initialize LCD memory and display modes*/
void dput(int byte);                        /*write data byte to LCD module*/
void cput(int byte);                    /*write command byte to LCD module*/
void outp (int data);                    /*put data onto the LCD data lines*/
void pput(int value);                  /*send a number to the LCD hardware*/
void cehi ();                                        /*pull LCD ce line high*/
void celo ();                                         /*pull LCD ce line low*/
void rdhi ();                                        /*pull LCD rd line high*/
void rdlo ();                                         /*pull LCD rd line low*/
void wrhi ();                                        /*pull LCD wr line high*/
void wrlo ();                                         /*pull LCD wr line low*/
```

University of Cape Town

```
void cdhi ();                                          /*pull LCD cd line high*/
void cdlo ();                                           /*pull LCD cd line low*/

/*=========================================================================*/

void lcd_putarray ()
{/*This function plots a graph on the screen. The input comes in in the 128
   element array, grapharray. Each element of grapharray represents one
   pixel. The input array is signed. Zero places the point halfway down the
   screen.
   In order that the data fits onto the screen the vertical axis is scaled
   by a factor of s_scale.
   The function features an interpolation system so that a clear graph is
   obtained.*/

 int col,interp,n,x,y;
 /*col: which column  (array element) is being processed.
   n: how many points to fill in when interpolating.
   x: which column to put interpolation point in.
   y: which row to put interpolation point in.*/

 for (col=0;col<128;col++) grapharray [col]/=s_scale; /*scale incoming data*/

 for (col=0;col<128;col++)
 {lcd_setpixel (col,(64-grapharray[col]),1);
 /*we want linear interpolation to fill in the pixels to get a nice display*/
  n=(grapharray[(col+1)]-grapharray[col]);
  if (n>63) n=63;          /*The interpolation range is limited to +-63. This*/
  if (n<-63) n=-63;   /*limits the mess which interpolation makes of out of*/
                                                          /*range inputs.*/
  if ((abs (n)>=1)&&(col<127))      /*if we need to place at least one
            interpolation point, and it's not the last column on the screen*/
      for (interp=1;interp<abs(n);interp++)
       {x=(col+(interp/abs(n)));
      if (n>0) y=grapharray[col]+interp;
      else y=grapharray[col]-interp;
      lcd_setpixel (x,(64-y),1);
       }                                                 /*end for interp*/

 }                                                         /*end for col*/
}                                                         /*end put_array*/

/*=========================================================================*/

void lcd_print(char *string)
{/*Send string of characters to LCD. The normal C conventions for strings
   apply here.*/
int i;
      for (i=0;i<strlen(string);i++)
       lcd_put_char (string[i]);

}                                                       /* end lcd_print*/

/*=========================================================================*/

void lcd_clear_graph()
{/*clear graphics memory of LCD. This leaves the text unaffected.*/

 int i;

 dput(G_BASE%256);
 dput(G_BASE>>8);
 cput(0x24);  /*addrptr of LCD at address G_BASE, the graphics base address*/
                    /*graphics area is 30 bytes per row*128 rows=3840 bytes*/
 for (i=0;i<3840;i++)
```

University of Cape Town

```
   {dput(0);
    cput(0xc0);                                  /*write data, increment pointer*/
   }                                                          /*end for(i)*/
}                                                        /*end lcd_clear_graph()*/

/*=============================================================================*/

void lcd_clear_text()
{/*Clear text memory of the LCD. This leaves the graphics unaffected.*/
 int i;

 dput(T_BASE%256);
 dput(T_BASE>>8);
 cput(0x24);    /*addrptr of LCD at address T_BASE,the start of text memory*/
               /*Text area is 30 bytes per row and 128/8 rows, ie 480 bytes*/
 for (i=0;i<480;i++)
  {dput(0);
   cput(0xc0);                                /* write data, increment pointer*/
  }                                                           /*end for(i)*/
}                                                         /*lcd_clear_text()*/

/*=============================================================================*/

void lcd_xy(int x, int y)
{/*Set memory pointer to (x,y) position. This makes the next text appear at
   that position. The position is quoted as a column and row. This function
   does the memory mapping.*/
 int addr;

  text_x=x;
  text_y=y;
                                 /*column and row converted to memory address*/
  addr = T_BASE + (y * BYTES_PER_ROW) + x;
  dput(addr%256);
  dput(addr>>8);
  cput(0x24);                                          /*set LCD addr. pointer*/
}                                                                 /*lcd_xy()*/

/*=============================================================================*/

void lcd_put_char (char data)
{/*Place a character on the screen at the current position. This function
   takes in ASCII rather than the LCD's character set.*/

 int c;
 c=data-0x20;                        /*convert ASCII to LCD character address*/
 if (c<0) c=0;
 dput (c);
 cput (0xc4);           /*write characters. Don't increment memory pointer as
                                        this seems to mess up the graphics.*/
 text_x++;                           /*manually increment the text position.*/
 lcd_xy (text_x,text_y);
}                                                       /*lcd_put_char(char)*/

/*=============================================================================*/

void lcd_setpixel(int column, int row,int optype)
{/*Sets/clears a single pixel in 128X128 array. If optype=0 it clears the
   pixel, if optype=1 it sets the pixel. The pixel is specified by a column
   and row. The column/row to memory address conversion is also done here.

   Notice that in this case the LCD is set for 6*8 characters. This also
   means that each byte in graphics memory only represents 6 pixels*/

 int addr;              /* memory address of byte containing pixel to write*/
```

University of Cape Town

```
                                    /*convert column/row to LCD memory address*/
  addr =  G_BASE + (row*BYTES_PER_ROW)  + (column/6);
  dput(addr%256);
  dput(addr>>8);
  cput(0x24);                                        /*set LCD address pointer*/
  cput(0xf8 | (5-(column%6)) );                   /*set bit-within-byte command*/
                      /*if optype is zero we clear the  bit within the byte*/
  if (optype==0) cput(0xf0 | (5-(column%6)) );
}                                                          /*end lcd_setpixel()*/

/*=============================================================================*/

void delay(unsigned long d)
{/*Makes a delay proportional to "d" value*/

unsigned long i;
double a;

 a = 1.000;
 for (i=0;i<d;i++) a = a / 1.001;       /*do lots of floating point divides
                                         because we are trying to waste time*/

}                                                               /*end delay()*/

/* ============================================================================
 * Low-level I/O routines to interface to LCD display
 * based on two routines:
 *
 *          dput(): write data byte
 *          cput(): write control byte
 * ============================================================================*/

void lcd_setup()
{/*Make sure control lines are at correct levels when starting the LCD*/

 cehi ();                                                 /*disable LCD module*/
 rdhi ();                                          /*disable reading from LCD*/
 wrhi ();                                            /*disable writing to LCD*/
 cdhi ();                                               /*command/status mode*/
}                                                          /*end lcd_setup()*/

/*=============================================================================*/

void lcd_init()
{/*Initialize LCD memory and display modes. This function also sets up the
   graphics and text memory areas, how many bytes are used per line, it
   enables simeltaneous graphics and text and sets the LCD up so that where
   graphics overlaps text the pixel is the result of a logical OR (i.e. the
   pixel is on.*/

 dput(G_BASE%256);
 dput(G_BASE>>8);
 cput(0x42);                          /*set graphics memory to address G_BASE*/

 dput(BYTES_PER_ROW%256);
 dput(BYTES_PER_ROW>>8);
 cput(0x43);                          /*BYTES_PER_ROW bytes per graphics line*/

 dput(T_BASE%256);
 dput(T_BASE>>8);
 cput(0x40);                           /*text memory starts at address T_BASE*/

 dput(BYTES_PER_ROW%256);
 dput(BYTES_PER_ROW>>8);
```

University of Cape Town

```
 cput(0x41);                              /*BYTES_PER_ROW bytes per text line*/

 cput(0x80);   /*mode set: Graphics OR Text, ROM CGen used to generate text*/

 cput(0x95);                                          /*Graphics & Text ON*/

}                                                          /*end lcd_init()*/

/*============================================================================*/

void dput(int byte)
{/*write data byte to LCD module*/
  delay(10);
  cdlo ();
  wrlo ();                                      /*activate LCD's write mode*/
  outp(byte);                                    /*write value to data port*/
  celo ();                                       /*pulse enable LOW > 80 ns*/
  cehi ();                                             /*return enable HIGH*/
  wrhi ();                                  /*restore Write mode to inactive*/

}                                                            /*end dput()*/

/*============================================================================*/

void cput(int byte)
{/*write command byte to LCD module*/

  delay(10);
  outp(byte);                          /* present data to LCD on LCD data pins*/

  cdhi ();                                            /*control/status mode*/
  rdhi ();                                  /*make sure LCD read mode is off*/
  wrlo ();                                        /*activate LCD write mode*/
  celo ();                  /*pulse ChipEnable LOW, > 80 ns, enables LCD I/O*/
  cehi ();                                              /*disable LCD I/O*/
  wrhi ();                                       /*deactivate write mode*/

}                                                              /*cput()*/

/*============================================================================*/

void outp (int data)
{/*Put out data onto the LCD data lines. The LCD data lines and control
   lines are mapped to the same address, and so some care is needed not to
   scramble the control lines when putting out data.*/

 portvar=portvar&0xFF00;
 portvar=portvar|data;
 pput (portvar);
}

/*============================================================================*/

/*Sends a number to the port that connects to the LCD. A global variable
  called temp_ax0 is used to back up the DSP's ax0 register to prevent it
  from being clobbered. (ax0 is not a scratch register)*/

int temp_ax0;

void pput(int value)
{myvar=value;          /*myvar is global which allows it to communicate with
                                                      assembly instructions*/
 asm ("dm(temp_ax0_)=ax0;");                                  /*save ax0*/
 asm ("ax0=dm(myvar_);");
 asm ("IO(0x0001)=ax0;");            /*put the number into IO address space*/
```

University of Cape Town

```
 asm ("ax0=dm(temp_ax0_);");                                    /*restore ax0*/
}

/*=========================================================================*/
/*The following routines were all explained in the header at the top of this
  file.*/

void cehi ()
{portvar=portvar|0x0400;
 pput (portvar);
}

void celo ()
{portvar=portvar&0xFBFF;
 pput (portvar);
}


void rdhi ()
{portvar=portvar|0x0200;
 pput (portvar);
}

void rdlo ()
{portvar=portvar&0xFDFF;
 pput (portvar);
}

void wrhi ()
{portvar=portvar|0x0100;
 pput (portvar);
}

void wrlo ()
{portvar=portvar&0xFEFF;
 pput (portvar);
}

void cdhi ()
{portvar=portvar|0x0800;
 pput (portvar);
}

void cdlo ()
{portvar=portvar&0xF7FF;
 pput (portvar);
}

/*=========================================================================*/
```

University of Cape Town

## B.4 Ana_flash.c

```
/*Written By: Samuel Ginsberg

This file contains the high level CompactFlash access functions. Functions
for disk management are here. These include the disk formatter and record
management facilities.  */

/*These variables are all declared in other modules*/
extern unsigned char lba1;                          /*Sector address variables*/
extern unsigned char lba2;
extern unsigned char lba3;
extern fat_start;      /*address of the start of the FAT area on the device*/
extern fat_end;          /*address of the end of the FAT area on the device*/
extern range;  /*variable which determines which range the instrument is in*/
extern curr_fat_entry;     /*address of the FAT entry for the active record*/
extern clk_com;                    /*command to send to the real time clock*/
extern clk_dat;                  /*data to send to/from the real time clock*/
extern char active_file_selected; /*flag which is set if a record is active*/
extern int walking_route; /*flag which tells if the user is walking a route*/
extern int route_active_file;  /*address of FAT entry for current route pos*/
extern string[320];                              /*a general purpose string*/
extern buffer [3];                  /*sector buffer actually 256 words long*/
extern xreal [3];               /*main data arrays actually 4096 words long*/
extern ximag [3];
extern screen_buf [3];                                        /*screen buffer*/
extern clockstring[9];             /*string into which clock data is packed*/

/*These functions are all defined in other modules*/
extern void setup_lba ();      /*set the lba address into the ide registers*/
extern void write_sector ();         /*write a sector onto the CompactFlash*/
extern void read_sector ();          /*read a sector from the CompactFlash*/
extern void drive_id ();       /*get the identification data from the flash*/
extern void lcd_clear_text ();                    /*clear text off the LCD*/
extern void lcd_clear_graph ();                 /*clear graphics off the LCD*/
extern void data_to_graph (int graphtype);       /*convert data array format*/
extern void lcd_putarray ();                    /*display a graph on the LCD*/
extern void clk_string (int detail);         /*read from the real time clock*/
extern unsigned int soft6 [2];                          /*quit hotkey legend*/
extern char* itoa (unsigned long in_num, char* input, int len);
                                                /*convert numbers to strings*/
/*Functions defined in this module*/
void get_fat_start ();     /*determine the position of the start of the FAT*/
void write_mbr ();              /*write the master boot record to the flash*/
void clear_ret ();                            /*clear the route entry table*/
void setup_fat ();                       /*set up the File allocation table*/
int used_fat_entry (int fat_entry_addr); /*determine if a FAT entry is used*/
void display_fat_entry();                        /*show the name of a record*/
void delete_record ();                                  /*delete a record*/
void upload_file ();                  /*upload a record via the serial port*/
int used_route_entry (int route_entry_addr);
                                    /*determine if a route entry is blank*/
void display_route_entry (int fat_entry_addr);      /*display a route entry*/
void getsec ();                   /*setup the lba address and read a sector*/
void savedata ();                                /*save data into a record*/
void disp_saved_data ();                /*retrieve the data from a record*/
void format ();                                    /*format the device*/
long int create_record ();                        /*create a new record*/

/*=====================================================================*/
void savedata ()
/*This function takes in the xreal and ximag arrays and saves them
 to the flash memory.*/
```

University of Cape Town

```
{int loop,count,temp;

                  /*first check if there is an active file to save data to*/
 if ((active_file_selected==0)&&(walking_route==0))
                              {lcd_xy (0,1);
                               lcd_print ("No file!");
                               delay (1000000);
                               lcd_xy (0,1);
                               lcd_print ("               ");
                               }

 else                                          /*if there is an active file*/
    {if (walking_route==0)              /*see if the record is stand alone*/
       /*if stand alone load up its FAT entry address from curr_fat_entry*/
                             {lba1=(curr_fat_entry&0xFF);
                              lba2=((curr_fat_entry>>8)&0xFF);
                             }
     else                                            /*or part of a route*/
/*if it is part of a route load up its address from the route and position*/
                             {lba1=(route_active_file&0xFF);
                              lba2=((route_active_file>>8)&0xFF);
                              }

     lba3=0;
     getsec ();                                          /*read the sector*/
     buffer[3]=0;                    /*mark the data in the sector as valid*/
     setup_lba ();
     write_sector ();                  /*write the sector back to the Flash*/

        lba1=buffer[0];              /*set up the address of the actual data*/
        lba2=buffer[1];              /*from the data read from the FAT entry*/
        lba3=buffer[2];

           /*place a time stamp in the information sector for that record*/
        clk_string (0);
        for (count=0;count<8;count++)
             buffer[count]=clockstring[count];
         buffer[8]=0;           /*add in a null terminator for easy display*/

           /*place a date stamp in the information sector for that record*/
         clk_string(1);
         for (count=0;count<8;count++)
               buffer[count+9]=clockstring[count];
         buffer[17]=0;          /*add in a null terminator for easy display*/


         clk_com=0x00F1;  /*save the sample rate in the information sector*/
         get_clk();
         buffer[18]=clk_dat;
         buffer[19]=range; /*save the range info in the information sector*/
         setup_lba ();
         write_sector ();

             /*read in the actual data from the data segment of the record
               use two nested loops: the outer works through the 32 sectors
                              while the inner loop works through each sector*/
         for (count=0;count<32;count++)
             {
              lba1++;                         /*lba address might roll over out*/
              if (lba1==256) {lba1=0;                  /*of a byte boundary*/
                   lba2++;
                   if (lba2==256) {lba2=0;
                                   lba3++;
                                  }
```

University of Cape Town

```
            }
          for (loop=0;loop<128;loop++)
              {buffer [(2*loop)]=(int)(xreal [(count*128)+loop]);
               buffer [((2*loop)+1)]=(int)(ximag [(count*128)+loop]);
              }
          setup_lba();
          write_sector();                    /*write the data into the sector*/
      }
    }
}

/*=========================================================================*/
void disp_saved_data ()
/*retrieve data from a record and place it into the xreal and ximag arrays.*/

{int loop,count,pos;

 lba1=(curr_fat_entry&0xFF);   /*set up the sector address of the FAT entry*/
 lba2=((curr_fat_entry>>8)&0xFF);
 lba3=0;
 getsec ();
 if (buffer[3]==0x003F)              /*see if the data in that record is valid*/
    {for (loop=0;loop<4096;loop++)       /*if not fill the data arrays with*/
         {xreal [loop]=0;     /*zeros so as not to present false information*/
          ximag [loop]=0;
         }
     for (loop=0;loop<8;loop++)
         {string [loop]='-';              /*fill the time string with dashes*/
          string [loop+9]='-';   /*because no valid time stamp is available*/
         }
     string[8]=0;
     string[17]=0;
     return;
    }

    lba1=buffer[0];          /*set up the address of the information sector*/
    lba2=buffer[1];
    lba3=buffer[2];

    getsec ();                                  /*read the information sector*/
                    /*pack the information into a string for later display*/
    for (count=0;count<255;count++) string [count]=buffer[count];

                  /*read the data from the data segment into the data arrays
                 two loops are used, one to interate through the32 sectors of
                          data and another to iterate within each sector*/
    for (count=0;count<32;count++)
        {lba1++;
         if (lba1==256) {lba1=0;
                         lba2++;
                         if (lba2==256) {lba2=0;
                                         lba3++;
                                        }
                        }
         getsec ();
         for (loop=0;loop<128;loop++)
             {pos=(count*128)+loop;
              xreal [pos]=buffer [(2*loop)];
              ximag [pos]=buffer [((2*loop)+1)];
              screen_buf [pos]=0;
             }
        }
}

/*=========================================================================*/
```

University of Cape Town

```
void format ()
{/*format the drive in accordance with the standard set out for
 vibration data.*/

  lcd_clear_text ();
  lcd_xy (0,0);
  lcd_print ("Formatting...");
  active_file_selected=0;  /*because there are no records left after format*/

  write_mbr ();            /*set up sector zero with the master boot record*/
  clear_ret ();                          /*clear out the route entry table*/
  setup_fat ();                                          /*set the FAT up*/
  disp_soft (1,soft6); /*when the Flash is formatted allow the user to quit*/
  while (buttons()!=1);
}

/*=============================================================================*/
void write_mbr ()
{/*write sector zero with the drive identifier, fat start address end
   address etc.*/

 long int sectors;                      /*how many sectors are on the device*/
 int routes,size;    /*how many routes can be stored and how many megabytes*/
 long int records,temp;             /*how many records the device can store*/

 drive_id ();                /*get the device to present its identification*/
 sectors=buffer[1]*buffer[3]*buffer[6];    /*determine the number of setors*/
 fat_start=sectors/1500;    /*determine the address of the start of the FAT*/
 if (fat_start>255) fat_start=255;         /*maximum size route entry table*/
 fat_end=(sectors/34)+fat_start;     /*calculate the end address of the FAT*/
 size=sectors/20;  /*calculate size. size is premultiplied by 100 for itoa.*/
 lcd_xy (0,2);
 lcd_print ("Usable space: ");
 itoa (size,clockstring,6);           /*convert size to a string for display*/
 clockstring [3]=0;           /*notice that we use clockstring as a general*/
 lcd_print (clockstring);                              /*purpose string here*/
 lcd_print ("MB");

 lcd_xy (0,3);
 lcd_print ("Max routes: ");
 routes=50*(fat_start-1);       /*50 times because each route is two sectors
                                  and itoa expects 100* premultiply.*/
 itoa (routes,clockstring,6);
 clockstring [3]=0;
 lcd_print (clockstring);

 lcd_xy (0,4);
 lcd_print ("Max records: ");        /*each record occupies 1 FAT sector so*/
 records=fat_end-fat_start;        /*this gives the total number of records*/
 itoa ((100*records),clockstring,8);
 clockstring [5]=0;
 lcd_print (clockstring);
                                                 /*Set up identification sector*/
 buffer[0]=0x56;                                                        /*V*/
 buffer[1]=0x49;                                                        /*I*/
 buffer[2]=0x42;                                                        /*B*/
 buffer[3]=0x52;                                                        /*R*/
 buffer[4]=0x41;                                                        /*A*/
 buffer[5]=0x54;                                                        /*T*/
 buffer[6]=0x49;                                                        /*I*/
 buffer[7]=0x4F;                                                        /*O*/
 buffer[8]=0x4E;                                                        /*N*/
 buffer[9]=0;
 buffer[10]=0;
 buffer[11]=fat_start;            /*This sector also gives reference points*/
```

University of Cape Town

```
 buffer[12]=fat_end;              /*so that other functions can locate data*/
 lba1=0;
 lba2=0;
 lba3=0;
 setup_lba ();
 write_sector ();
}

/*=========================================================================*/
void clear_ret ()
{/*clear out the route entry table (ret) by writing all zeros to
   all the entries in the table.*/

 unsigned char count;
 int pos;

 for (pos=0;pos<256;pos++) buffer[pos]=0;  /*set the sector buffer to zeros*/

  lba2=0;               /*the route entry table size is 255 sectors maximum*/
  lba3=0;
  for (count=1;count<fat_start;count++)      /*the table starts at sector 1*/
      {lba1=count;
       setup_lba();
       write_sector ();             /*write the sector buffer to the sector*/
      }
}

/*=========================================================================*/
void setup_fat ()
{/*set up record entry table (fat) by writing the sectors to which the
   entries point and marking them all as blank with question marks*/

unsigned long int currsec;
unsigned int count,temp;

 for (count=fat_start;count<fat_end;count++)

   {lba1=(count&0xFF);          /*break the sector address into two seperate*/
    lba2=((count>>8)&0xFF);          /*bytes. Max FAT size is 2^16 sectors*/
    lba3=0;

    currsec=fat_end+((count-fat_start)*33);    /*33 sectors per data record*/
    buffer[0]=(char)(currsec&0xFF); /*set up the address pointer in the FAT*/
    buffer[1]=(char)((currsec>>8)&0xFF);
    buffer[2]=(char)((currsec>>16)&0xFF);
    buffer[3]=0x3F;                 /*mark the data in the record as invalid*/
    buffer[4]=0x3F;                          /*mark the record as deleted*/
    buffer[5]=(lba1+48);        /*useful for checking that the correct sector
                                                          was retrieved*/
    setup_lba ();
    write_sector ();
   }
}

/*=========================================================================*/
void get_fat_start ()
{/*returns the LBA address of the start of the FAT and the end of the FAT
 and leaves the global array buffer filled with the other info from the
 master boot record.*/
 lba1=0;
 lba2=0;
 lba3=0;
 setup_lba ();
 read_sector();
 fat_start=buffer[11];
```

University of Cape Town

```
 fat_end=buffer[12];
}

/*=========================================================================*/
int used_fat_entry (int fat_entry_addr)
{/*checks to see if the fat entry specified in the input argument
 is used. If it is a 1 is returned, if not a 0 is returned*/

 lba1=(char)(fat_entry_addr&0xFF);
 lba2=(char)((fat_entry_addr>>8)&0xFF);
 lba3=0;

  getsec ();

  if (buffer[4]==0x3F) return 0;                /*check the deletion marker*/

  return 1;
}

/*=========================================================================*/

void display_fat_entry (int fat_entry_addr)
{/*Put the details of a record on the screen. This effectively shows the
   record name to the user.*/

  int count,line;

  lba1=(char)(fat_entry_addr&0xFF);         /*load up the address of the FAT*/
  lba2=(char)((fat_entry_addr>>8)&0xFF);          /*entry to be displayed*/
  lba3=0;
  getsec ();

  if (buffer[4]!='?')                       /*check that the FAT entry is used*/
        for (line=0;line<11;line++)
            {for (count=4;count<25;count++)      /*display name information*/
                 {lcd_xy ((count-4),(line+2));      /*character by charcter*/
                  lcd_put_char (buffer[((line*21)+count)]);
                 }
            }
}

/*=========================================================================*/
long int create_record ()
{/*creates a new record at some available spot on the flash. The function
   looks for a free spot (marked by a '?' in the FAT) and loads the
   record name into it. The record name comes in as an array called string.
   the first 252 entries of string hold the name data. The function returns
   a long int which holds the LBA address of the FAT entry for the record
   just created.*/

   int count;
   int pos;

   get_fat_start();            /*get the start and end addresses of the FAT*/

   for (count=fat_start;count<fat_end;count++)   /*look sequentially at FAT*/
    {lba1=(char)count&0xFF;                    /*until a blank enttry is found*/
     lba2=(char)((count>>8)&0xFF);
     lba3=0;
     getsec ();
     if (buffer[4]==0x3F) break;   /*if a blank entry is found stop looking*/
    }

   /*Notice that the buffer holds the LBA address of the information sector*/
            /*copy the name into the buffer to be written to the FAT entry*/
```

University of Cape Town

```
   for (pos=0;pos<252;pos++) buffer [pos+4]=string[pos];

    lba1=(char)count&0xFF;      /*set up to write back into the sector*/
    lba2=(char)((count>>8)&0xFF);
    lba3=0;

    setup_lba();
    write_sector();

 return count;       /*return the address of the FAT entry that was created*/
}

/*=========================================================================*/
void delete_record ()
/*delete a record by marking it as blank.*/

{lba1=(char)curr_fat_entry&0xFF;
 lba2=(char)((curr_fat_entry>>8)&0xFF);
 lba3=0;
 getsec ();

 buffer[3]=0x003F;                                  /*mark the data as invalid*/
 buffer[4]=0x003F;                                 /*mark the record as deleted*/

 setup_lba();
 write_sector();
 active_file_selected=0;                    /*The active file no longer exists*/
}

/*=========================================================================*/
void upload_file ()
/*upload the selected data file to the PC via the serial port.*/

{int count,sec_count;

 lba1=(char)curr_fat_entry&0xFF;
 lba2=(char)((curr_fat_entry>>8)&0xFF);
 lba3=0;
 getsec ();                          /*fetch the address of the actual data*/

 lba1=buffer[0];
 lba2=buffer[1];
 lba3=buffer[2];
                                    /*send the name of the record to the PC*/
 for (count=3;count<256;count++) out_char_ar (buffer [count]);

 for (sec_count=0;sec_count<33;sec_count++)
   {getsec ();                          /*upload all 33 sectors of information*/
    for (count=0;count<256;count++)   /*split the words into bytes and send*/
        {out_char_ar (buffer[count]&0x00FF);
         out_char_ar ((buffer[count]>>8)&0x00FF);
        }
    lba1++;                                                 /*go to the next sector*/
    if (lba1==256) {lba1=0;          /*if byte rollover occurs deal with it*/
                    lba2++;
                    if (lba2==256) {lba2=0;
                                    lba3++;
                                   }
                   }
   }
}

/*=========================================================================*/
int used_route_entry (int route_entry_addr)
/*returns 1 if route is used, zero if route is unused. A route is defined as
```

University of Cape Town

```
  unused if all of the words in the first sector are set to zero.*/

{int count;
 lba1=route_entry_addr;       /*maximum route entry table size is 255 words*/
 lba2=0;
 lba3=0;
 getsec ();
 for (count=0;count<256;count++) if (buffer[count]!=0) return (1);
 return (0);
}

/*=========================================================================*/
void display_route_entry (int fat_entry_addr)
/*Put the details of a route on the screen. The function shows the
  lba address of the start of the actual data as well as the title info
  which the user input for that reading.*/

{int count,line;

 lba1=(char)(fat_entry_addr&0xFF);
 lba2=0;
 lba3=0;
 getsec ();

 for (count=0;count<10;count++)
     {lcd_xy ((count),2);
      lcd_put_char (buffer[count]);    /*print out the 10 letter route name*/
     }
}

/*=========================================================================*/
void getsec ()
/*set the lba address up and read the sector. This is just used to simplify
  other functions*/

{setup_lba ();
 read_sector ();
}

/*=========================================================================*/
```

University of Cape Town

# Appendix C

# Compilation, Assembly and linker files

## C.1 System Builder File, Ana.sys

```
.system ana_ach;
.adsp2181;
.mmap0;
.seg/pm/ram/abs=0/code/data int_pm[16384];
.seg/dm/ram/abs=0/data int_dm[16351];
.endsys;
```

## C.2 Linker Group File, Ana.grp

```
ana_uart
ana_clk
ana_io
ana_ide
ana_fl~1
ana_lcd
ana_math
ana
-a ana.ach
-e ana
-c
-lib stdlib.h
-lib string.h
-lib math.h
-end
```

## C.3 Builder Batch File Sam.bat

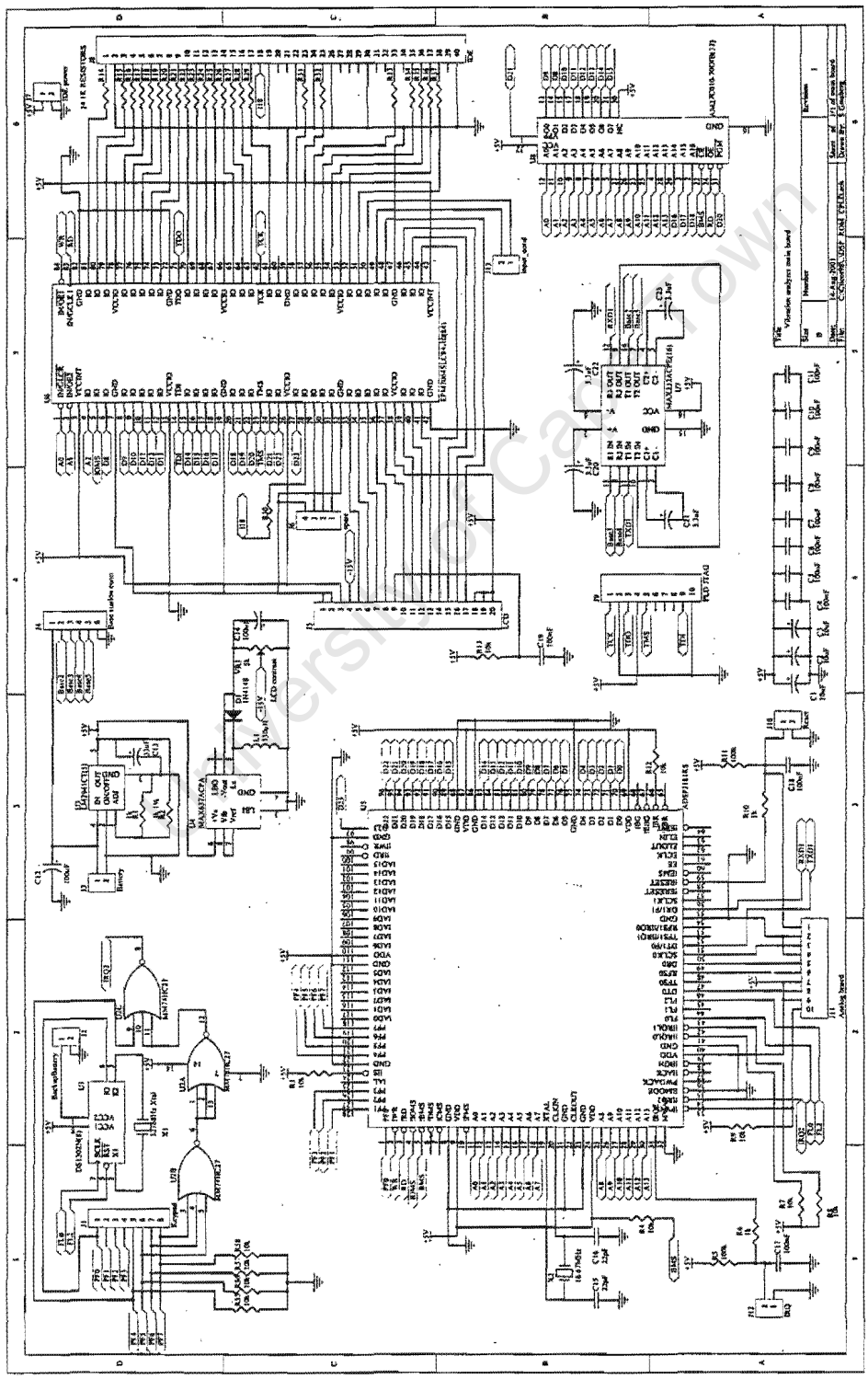
```
bld21 -c ana.sys
asm21 ana_hdr.dsp -c -s -2181 -o run_hdr
asm21 ana_uart.dsp -c -s -2181
asm21 ana_clk.dsp -c -s -2181
asm21 ana_io.dsp -c -s -2181
asm21 ana_ide.dsp -c -s -2181
g21 ana_fl~1.c  -c -mreserved=i2,i3 -fno-inline
g21 ana_lcd.c -c -mreserved=i2,i3 -fno-inline
g21 ana_math.c -c -mreserved=i2,i3 -fno-inline
g21 ana.c -c -mreserved=i2,i3 -fno-inline
ld21 -group ana.grp
spl21 ana anarom -loader -2181 -i
```

University of Cape Town

# Appendix D

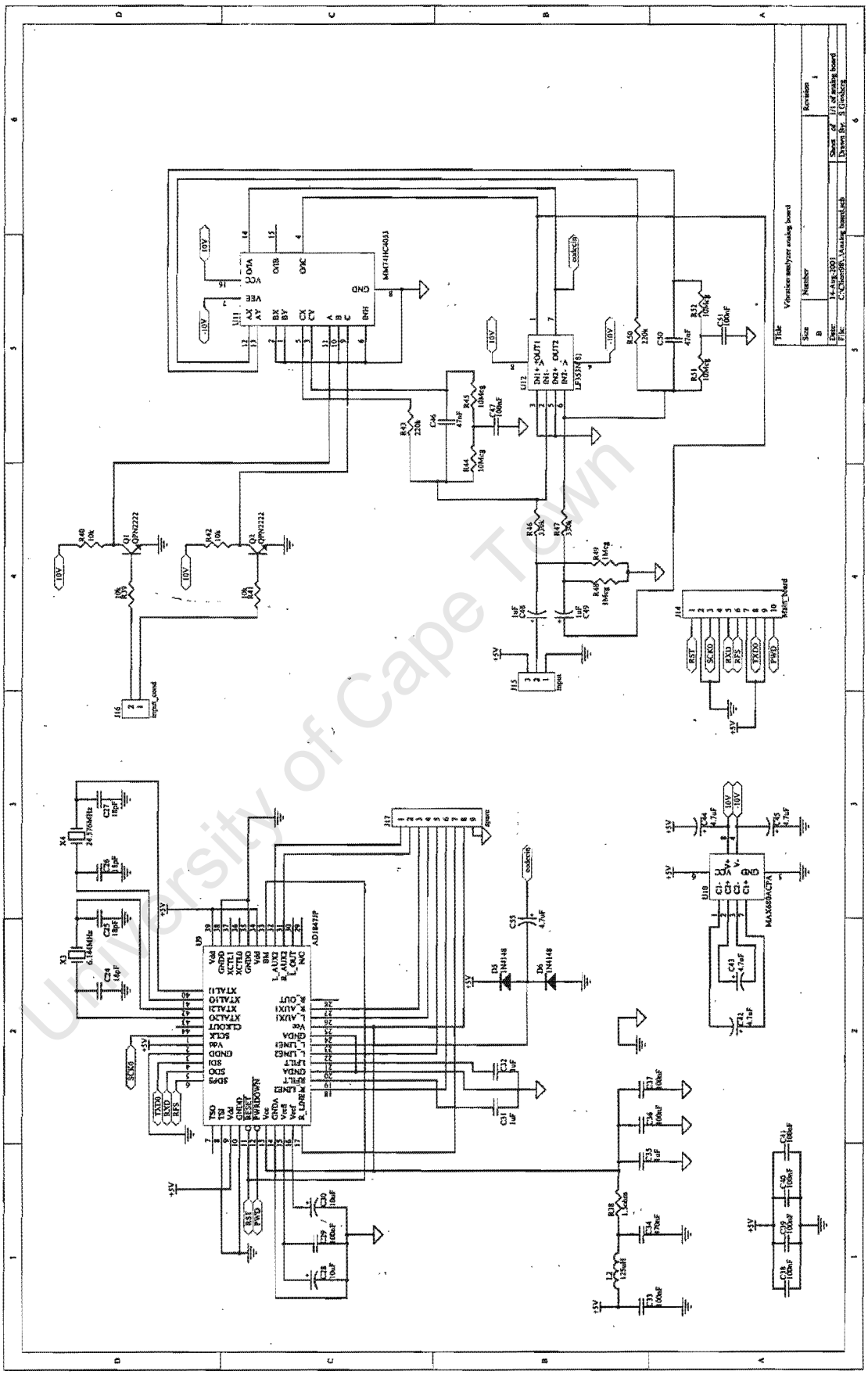
# Circuit Diagrams

## D.1 Main Board Diagram



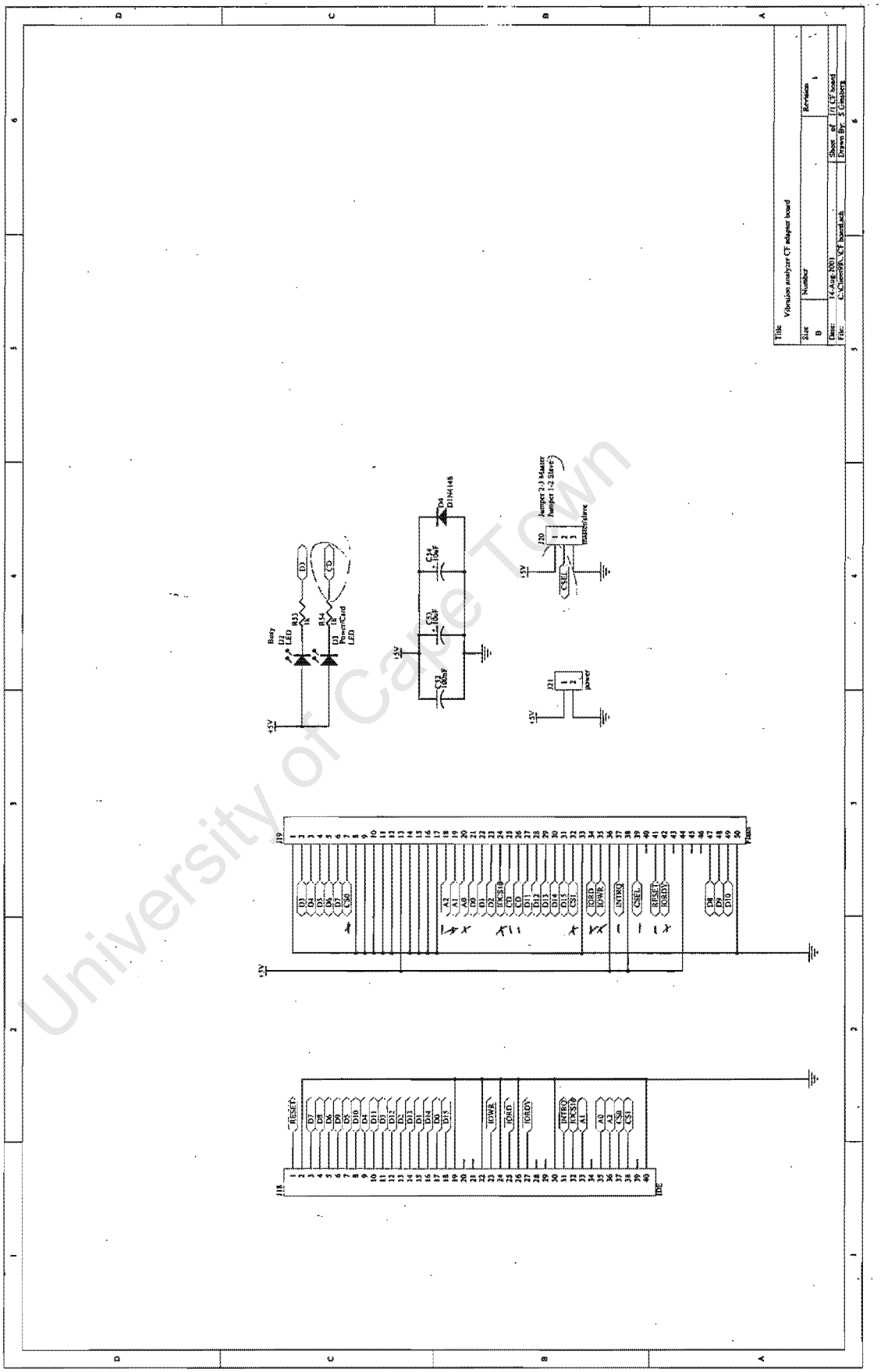
University of Cape Town

# D.2 Analog Board Diagram

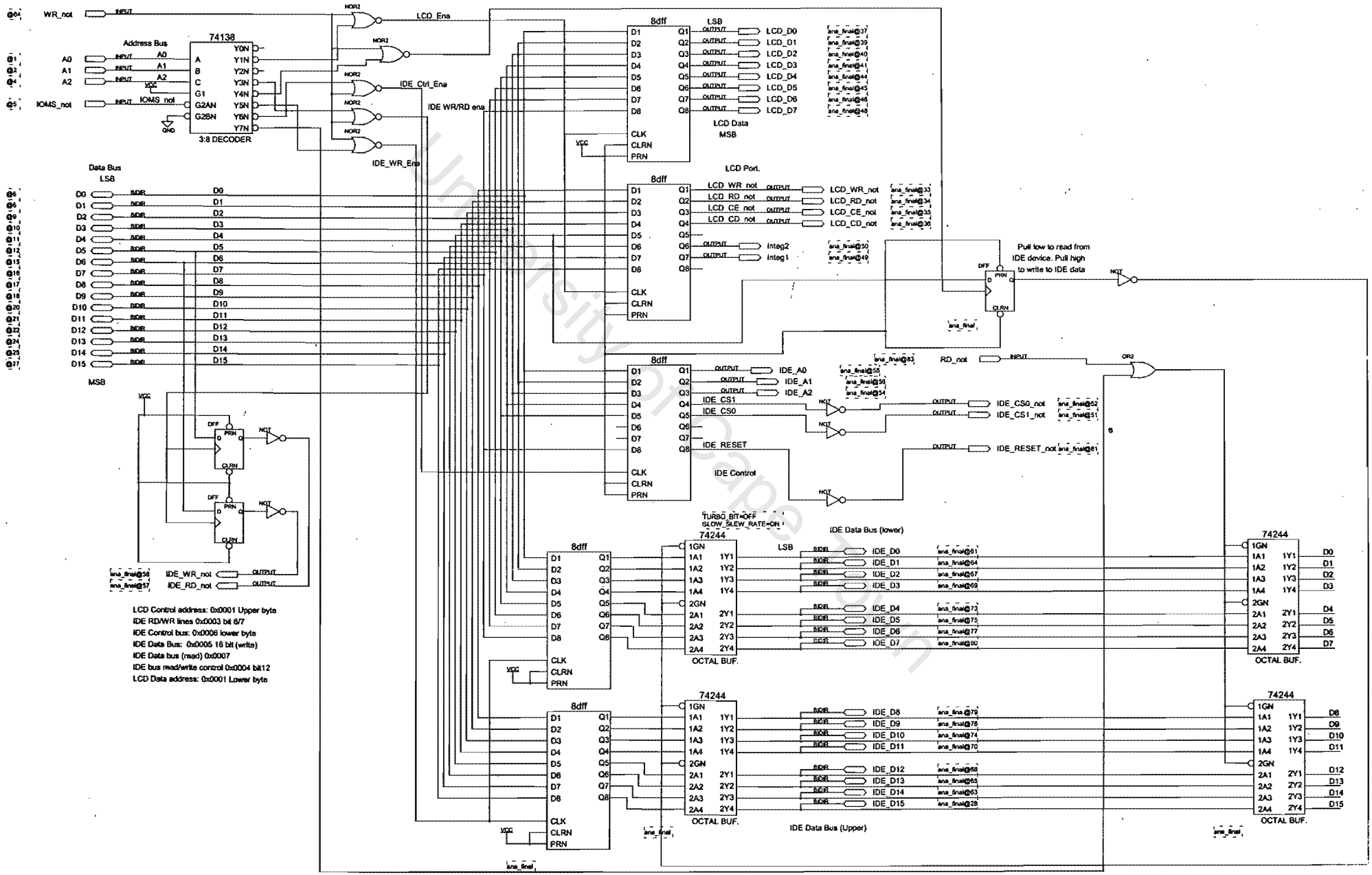


University of Cape Town

# D.3 CompactFlash Board Diagram




University of Cape Town



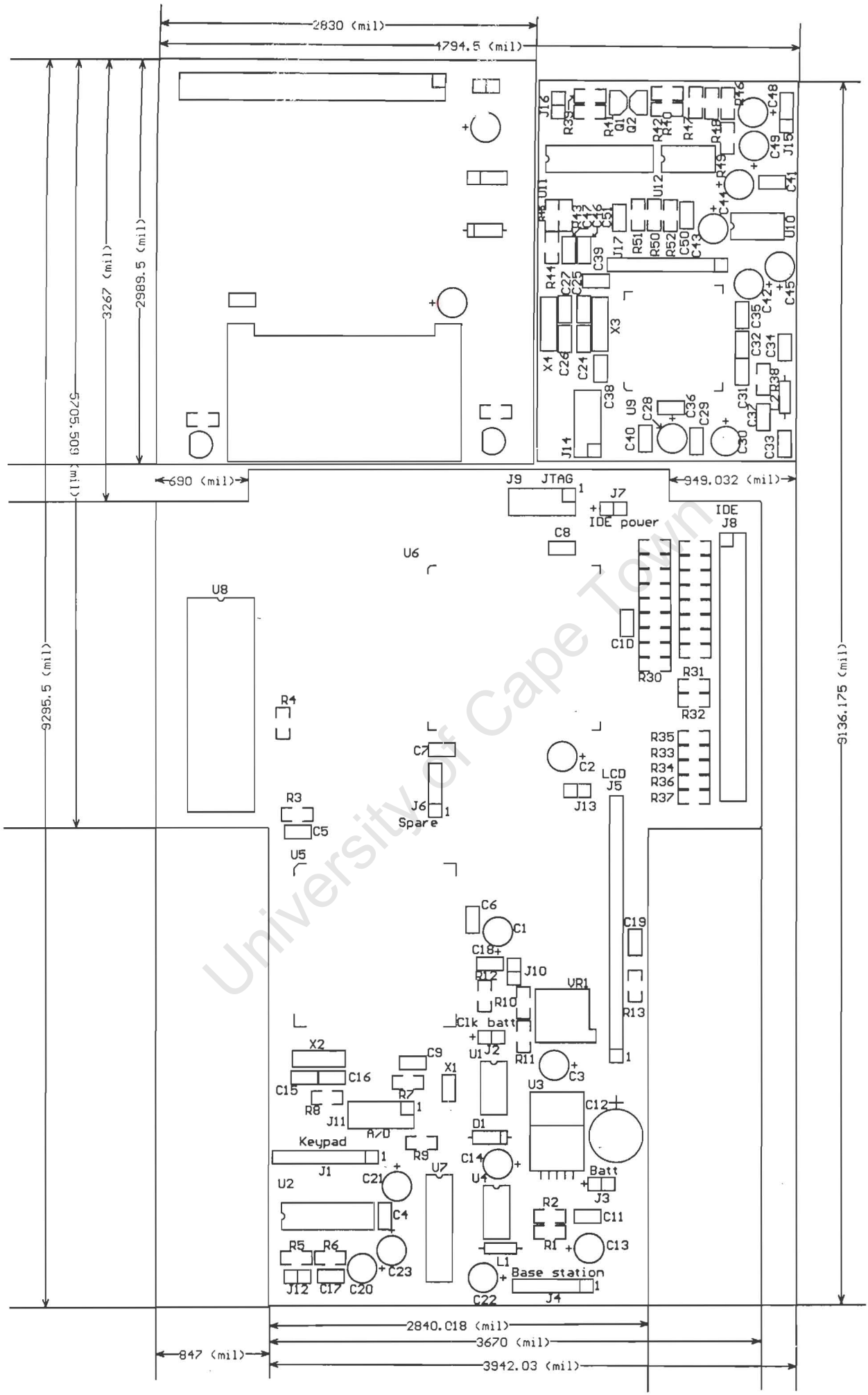

University of Cape Town

# Appendix E

# Printed Circuit Board Layouts

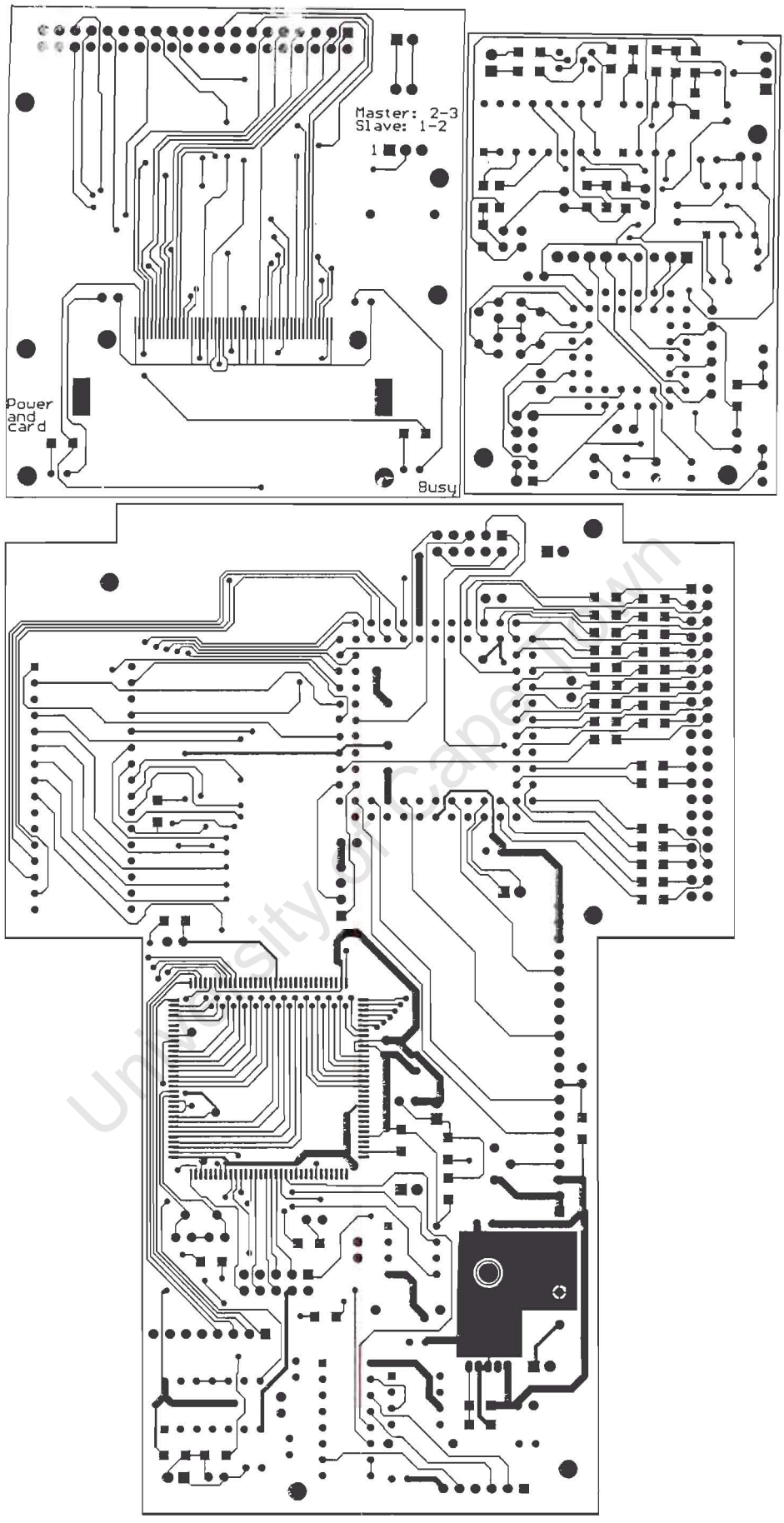
## E.1 Silkscreen for System Boards

University of Cape Town



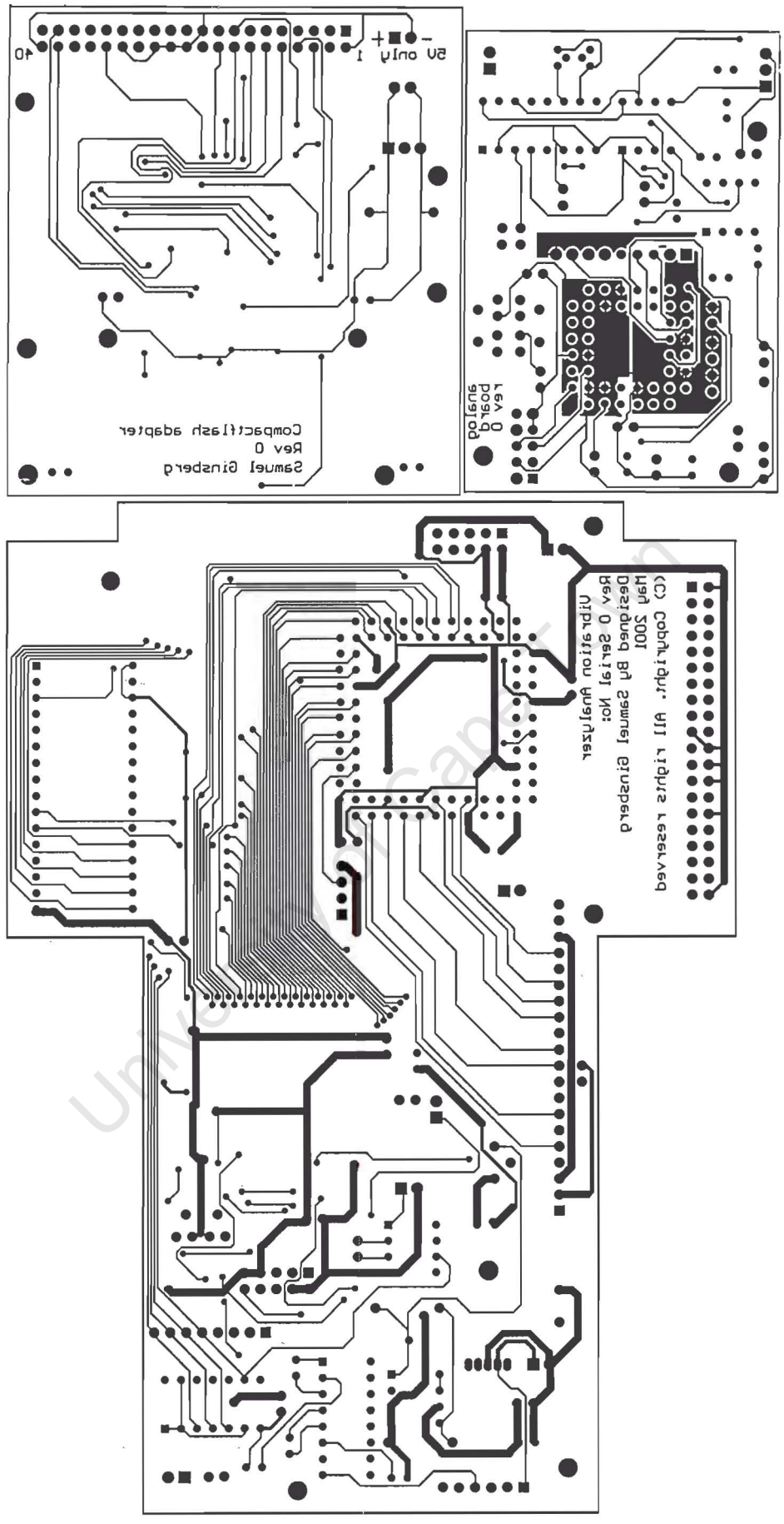

University of Cape Town

E.2 Top (Component side) track for system boards




University of Cape Town

## E.3 Bottom (Solder side) track for system boards

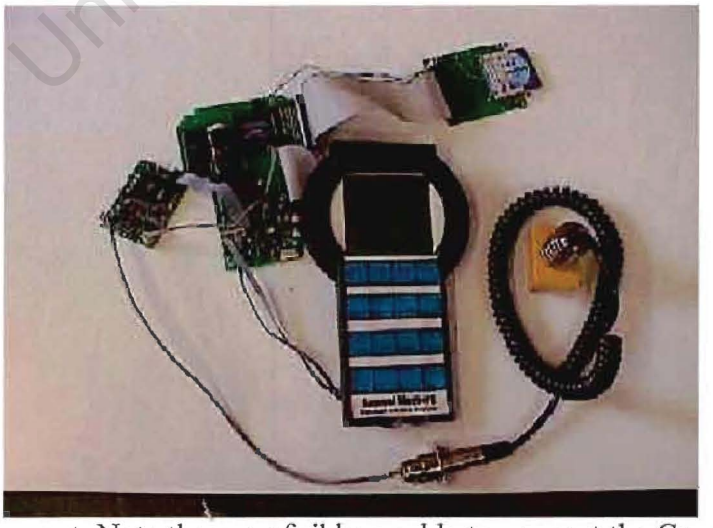



University of Cape Town

# Appendix F

# Photographs of Completed System

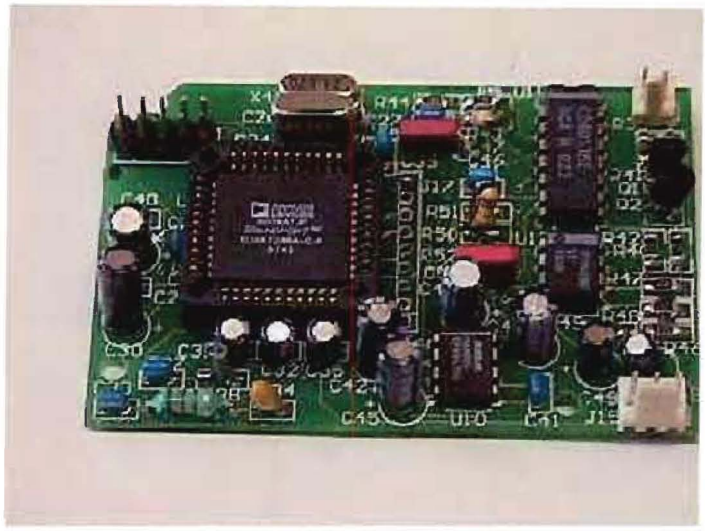
The completed analyzer in its casing. Note the membrane keypad fitted into the casing and the accelerometer mounted on its yellow magnetic clip.



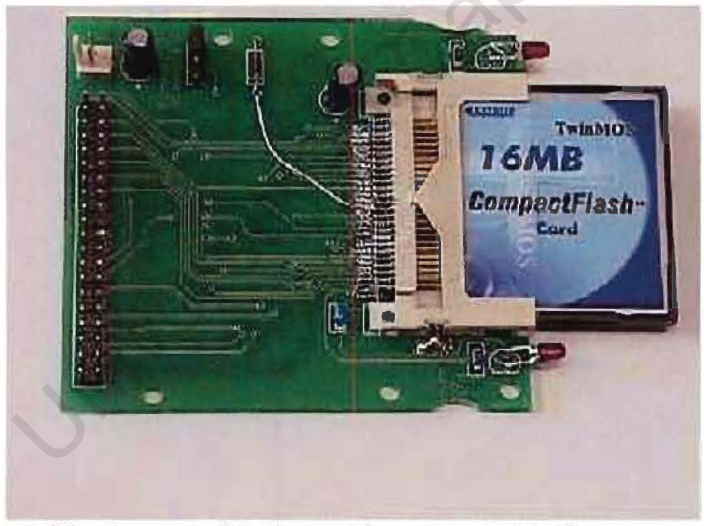
The disassembled instrument. Note the use of ribbon cable to connect the CompactFlash and display modules to the main board.



University of Cape Town

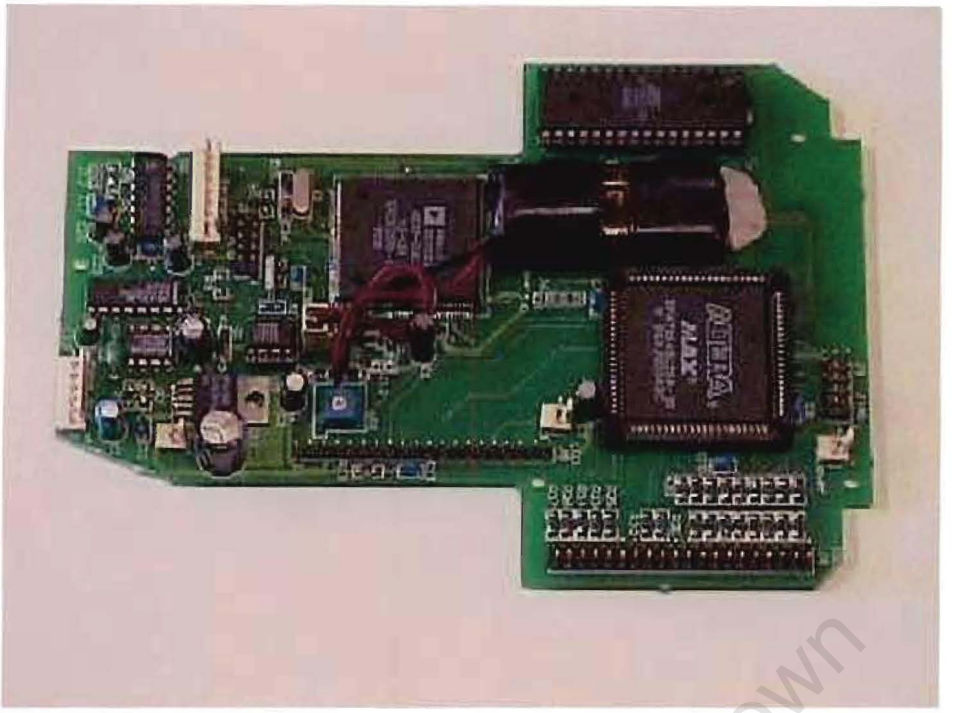


The analog board. The large square device is the CODEC, while the other integrated circuits are the analog preprocessing circuitry and power supply converter. The two CODEC crystals are at the top of the board.

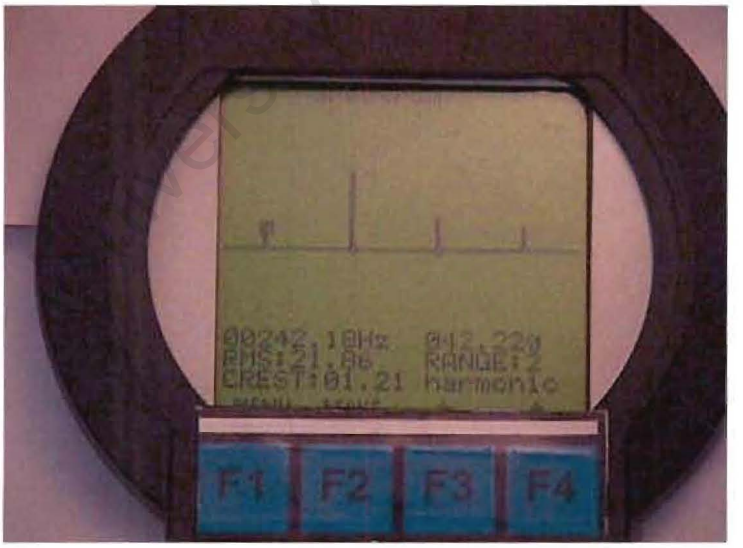



The CompactFlash board. The CompactFlash card is not seated in its connector. The two LEDs indicate power and card activity. The connector on the left is a standard 40 pin IDE connector.

University of Cape Town



The main board. The device at the very top of the board is the system ROM. The Altera MAX device is the programmable logic device and the other large device is the Digital Signal Processor. The serial port interface circuitry is to the left of the board, while the power supply circuitry is just below that. The Real Time Clock is located next to the blue potentiometer.




The display given by the instrument when in spectrum monitoring mode. Note the cursor on the fundamental and the small markers on the harmonics. The Frequency and amplitude of the cursor position are shown, along with the range, RMS value and crest factor of the signal. The hotkey legends are displayed at the very bottom of the screen.