MANCHESTER 1824
School of Electrical and Electronics Engineering
The University of Manchester

D4.6 Software defined toolbox containing fault models developed in Supergen Wind Phases 1 and 2 in a holistic drive train monitoring system for wind turbines.

Wind Turbine Condition Monitoring System

Alvaro Novoa, Daniel Carlson, Faqeeh Mannan,

Firdaus Khalid, Jack Wakley, Kevin Karikari,

Professor A. C. Smith, Dr. S. Durovic

Executive Summary

This project is motivated by the importance of wind energy and reducing the financial and operational impact of faults. Wind energy is an essential component of the solution to help reduce the carbon footprint of power generation. For maximum efficiency, wind turbines tend to be located in remote locations such as on offshore platforms. However, this remoteness leads to high maintenance costs and high downtime when faults occur. These factors highlight the importance of early fault detection and determine the motivation of the project.

The aim of the project has been to design and build a standalone wind turbine condition monitoring system (WTCMS). The system should have the capability to detect, identify and locate a fault in a wind turbine and communicate said information to remote supervision.

The objectives comprised all the tasks required to build an embedded monitoring system. Research on faults and fault detection techniques in wind turbine has facilitated the development of fault detection and analysis software. The fault detection techniques require data from specific sensors which were identified and acquired. Hardware and software was designed to convert the data from the sensors into a form which can be read by a controller. A software program coded into the controller provides a moving window of data for the fault analysis and detection software, the ability to view results on a detachable interface and trigger LEDs to indicate a fault. Another software program is coded for a detachable interface allowing remote viewing of data and control over the system. Power management hardware was designed to power system components and connected sensor. A prototype system and a small scale test rig were built to facilitate the development and testing of the final embedded system.

Most of the objectives explained above have been successfully achieved. The signal acquisition and power management hardware are ready to be assembled with the controller. This hardware runs the required software for the acquisition, processing and logging of data. A prototype system has been successfully built and has proved to be useful for preliminary testing of the fault analysis and detection program. System integration has not been successfully implemented due to the delays in delivery of some objectives, but this is expected to be complete before demonstration day.

Future work could include improvement in fault analysis and detection software, increased sampling frequency of signal acquisition hardware, addition of an LCD screen to compliment the LEDs, and designing more functions into the remote interface.

Contents

Introduction

1. Aims, Motivation & Objectives

1.1 Motivation

Wind turbines are an important source of renewable energy and have gained popularity in recent years. These turbines are generally sited in remote locations (such as off-shore) as that is where their yield is at its highest. Due to the remote locations of a lot of these wind turbines; the cost of maintenance for them is found to be extremely high. This issue has raised the demand for effective and reliable condition monitoring techniques.

A recent survey by Durham University shows that wind turbine technology is very mature with low failure frequency, but is inhibited by a high downtime [1]. **Figure 1** clearly shows the frequency and downtime of faults in different components of a wind turbine.

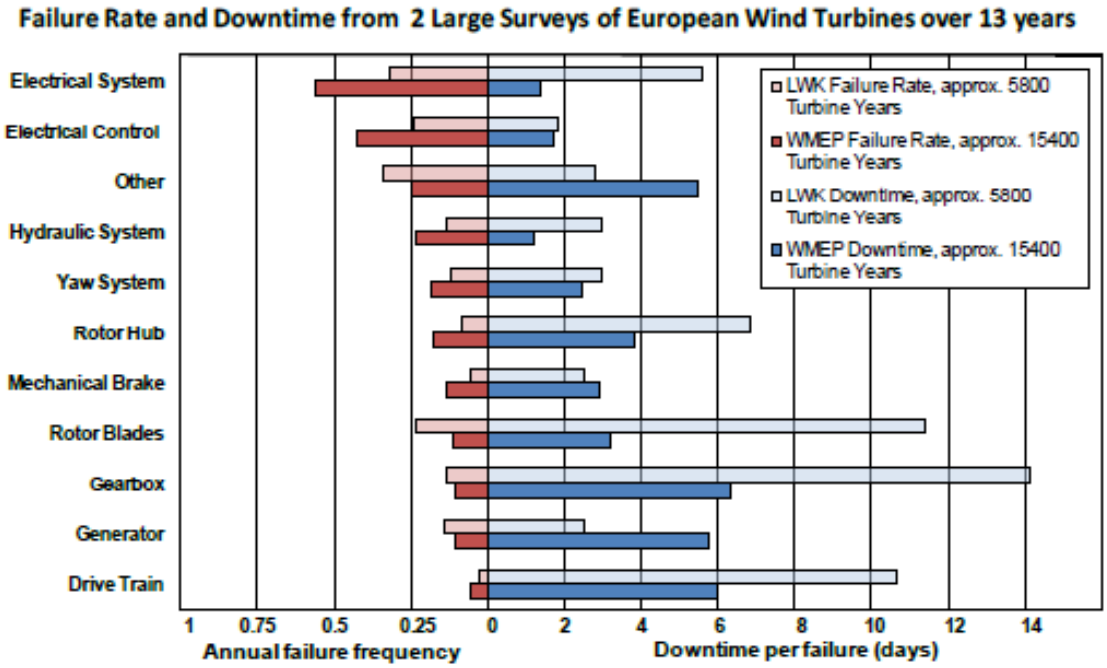


Figure 1: Wind turbine sub assembly failure rate and downtime per failure for two surveys [1]

The high downtime is often a result of complex repair procedures in remote locations. Without condition monitoring, when a fault occurs supervisors must first send a maintenance crew to the turbine to identify the faults location. Following that, repair may include specialist equipment such as cranes and support vessel with the potential for delays caused by unfavorable weather or wave conditions [1]. During these steps it is likely that no energy will be produced as without knowledge of the fault type the risk to operate cannot be taken. Obviously this reduces the efficiency and productivity of the system and increases overall costs.

A solution to this problem is to use an efficient condition monitoring system. These systems should be able to detect impending faults before they occur by analysing signals read from sensors connected to the wind turbine components. Additionally, by detecting the location of faults, the need for a thorough investigation can be eliminated. Being able to detect or even predict where and when a fault will occur will help reduce the downtime especially if a possible fault is detected in an early stage before complete system failure. With an indication of the nature and location of a fault the most effective measures can be taken; such as scheduled maintenance. With this

knowledge it may also be possible to avoid the wind turbine's operation being stopped during maintenance. With a high performance evaluation of the wind turbine, the time and cost for maintenance can be significantly reduced. This has become the foundation of motivation for the development of the group's project.

1.2 Aims and Objectives

The aim of the project is to build a stand-alone online condition monitoring system (CMS) which can determine early bearing and electrical fault detection and also the fault's location in the wind turbine system. The system should be flexible, transferrable and able to remotely communicate with an on-shore information centre. The detection of these faults is done using various sensors within the box to gather data on its condition. The data gathered by the sensors is then processed by frequency domain analysis and detection algorithms based on the theoretical knowledge of specific fault frequencies gained through research.

Additional complimentary data such as temperature and torque will also be acquired to help maintenance teams gauge the conditions which are affecting the wind turbine.

All functionality will be packaged in a small enclosure with appropriate means to communicate the results of signal analysis. There will also be a function which allows the maintenance the ability to connect their laptop to the CMS for additional checks and to run the tests using a custom-made Graphical User Interface (GUI).

To achieve these aims, a number of objectives were set out and need to be fulfilled.

The initial project specification shown in **Appendix G1: Original Project Description** gave a broad description of the main objectives. From this the scope of the project was narrowed to produce a final set of specifications with some additions and some refinements to the originals. With these adjustments, the tasks ahead were given more focus from the original set and with more measurable targets:

1. Acquire a range of sensors for evaluation with desirable specifications for their purpose for both fault analysis techniques and some auxiliary analysis.
2. Identify the main fault types to focus on and develop fault analysis and detection software to analyse signals with FFTs over time, and raise an alarm if changes or trends indicate a potential hazard has occurred or is going to occur in the system.
3. Develop a layout for a standalone condition monitoring system with sensor interface, on-board processing, power supply and remote control and viewing interface.
4. Design a custom DAQ PCB to appropriately convert sensor signals and filter as may be desirable.
5. Design a power management PCB to power all sensors and system components.
6. Develop code to run on a suitable off-the-shelf microprocessor board that interfaces and formats incoming data from the sensor interface board, presenting a moving time window to the developed fault analysis and detection software.
7. Develop other software that provides functions desirable in a condition monitoring system and make the control of these functions and the viewing of live/recorded data possible with a detachable interface.
8. Design a GUI and software to run on the detached interface allowing the viewing of data and control over the system.
9. Develop a small scale rig to aid development and demonstration of the proposed system.

2. Survey of Related Work

Wind turbine condition monitoring systems continue to improvement as technology advances. Many companies are now investing in the condition monitoring system market, and as a result the competition in design and price of these products been increasing.

Research obtained on some of the industry commercialised products has been used in the project design considerations due to its resultant useful features and attractive options. For instance, large companies such as Bruel and Kjaer have always used vibration and envelope analysis for fault detection in bearings or in the gearbox of a wind turbine [1]. The main method of fault detection for these types of fault was identified as the FFT spectral analysis due to its high frequency definition and simple implementation. The main wind turbine components supported by current equipment was identified as the bearing and gearbox components.

The following table (**Table 1**) offers a brief review on the products developed by the main active companies in the market. A summary with more detailed information can be found in the Supergen Wind consortium web page and in their surveys**Invalid source specified.**. Further information on currently commercialised wind turbine systems discovered from this can be found in **Overall Summary: section 4**.

Product and Company		The product details		
Products	Manufacturer	Main Components Monitored	Monitoring Technology	Analysis Method
Ascent	Commtest	Main shaft, Gearbox, Generator	Vibration (Accelerometer)	FFT frequency domain analysis
Vibro	Bruel and Kjaer	Main bearing, generator, gearbox, nacelle	Vibration	Time domain FFT frequency analysis
CMS	Nordex	Main bearing, gearbox, generator	Vibration (Accelerometer)	Time domain based on initial footprint
Condition based maintenance system (CMB)	GE (Bently Nevada)	Main bearing, generator, gearbox, nacelle	Vibration (Accelerometer)	FFT domain analysis Acceleration enveloping
Windcon 3.0	SKF	Blade, main bearing, gearbox, generator	Vibration (Accelerometer)	FFT frequency domain analysis

Table 1: Table of the developed condition monitoring system [1]

3. Project Overview

Figure 2 shows a high level view of this projects final design as a group of interconnected blocks.

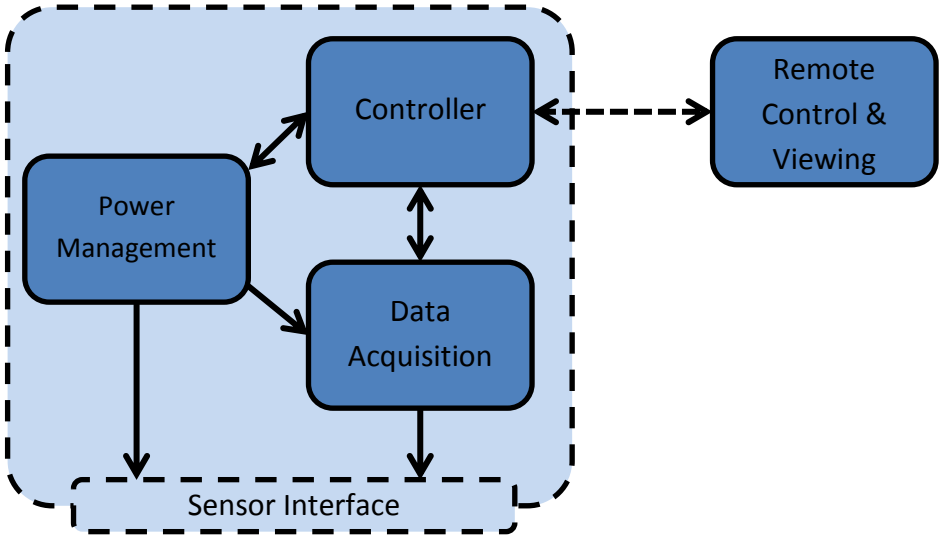


Figure 2: High level component layout of the Condition Monitoring System and detachable Control and Viewing Platform

The main functionality of each block is explained below.

3.1 Controller

The controller provides most of the systems functionality and governs the operation of intelligent subsystem. It also provides an interface for external control and viewing.

It maintains a record of all settings and can be updated with new settings that configure the tasks of acquisition, analysis and logging of data on the controller. These are used to configure the connected custom DAQ hardware so that it can start sampling connected sensors and streaming this data back to the controller. After this, its main function is to interface with the custom DAQ, receiving its already filtered raw signals, converting them into their appropriate units and presenting them in a moving window to the on-board fault analysis and detection software.

The fault analysis and detection software to run on the controller is a piece of MATLAB code, developed separately and then integrated into the controller software. Its design is based on the principle that when there is a fault, the spectral energy of certain fault frequencies will increase. When their energy level is higher than a certain threshold, a fault's occurrence and type can be detected based on the energies. This information is sent to the controller software to raise an alarm.

The controller also provides the following functionality which has been identified as desirable in a condition monitoring system:

- Make raw and analysed signals available for external viewing.
- Log data in response to a fault. Maintain a buffer of incoming data so that when fault occurs, both signals before and after the point of detection can be stored for a better understanding.
- Log data in response to instructions received by the remote control & viewing platform following an operators request.
- Provide an interface for updating settings and a way to view the systems state and data locally.
- Run independently from any remote supervision once configured.

The hardware used for the controller is a National Instruments single board RIO (Re-configurable Input and Output). The specifications of this platform which lead to its use are listed below along with a description of their use:

- Microprocessor running LabVIEW code with integrated MATLAB scripts for fault analysis and detection.
- FPGA used to implement an SPI (Serial Peripheral Interface) link to connect it to the DAQ hardware.
- Digital I/O suitable for controlling LED indicators and reset pins on the power management hardware.
- USB storage for settings and logged data.
- Ethernet port for connection to the remote control and viewing platform.

This platform was chosen over others with similar specifications because it was made available for free from National Instruments.

The controller addresses **points 2, 6 & 7** of the revised project specifications in **Introduction: section 1.2**.

3.2 Data Acquisition Hardware

For the system to identify faults from harmonic analysis of sensor signals, it must first be converted to the digital domain. This step is addressed by the data acquisition hardware which includes overvoltage and overcurrent protection, signal conditioning, multiplexing, analogue to digital conversion and digital signal processing.

Custom DAQ hardware was built to provide these functions with the following specifications:

- 5V overvoltage protection.
- 250mA overcurrent protection.
- 2 KHz analogue filters.
- Digital filtering with customisable cut-off frequencies and filter type settings.
- Up to 18 sensors that are individually selectable.
- 16 bit, 100 KHz Analogue to digital conversion.

The Custom Data Acquisition Hardware addresses **point 4** of the revised project specifications in **Introduction: section 1.2.**

3.3 Power Management Hardware

For the overall system to operate outside a lab environment it is important that it has its own power supply. This was identified at an early stage in the project and requires a very close interaction with the rest of the sub systems. This was dubbed “power management” system and through investigation, 5 voltage levels where identified as required these are: 5v, 1.3v, ±15v, 15v and 3.3v.

To update settings on the custom DAQ hardware, a connection from the controller to the power management board is used. This enables the controller to power cycle the custom DAQ board which upon restarting, requests new settings from the controller.

The power management hardware addresses **point 5** of the revised project specifications in **Introduction: section 1.2.**

3.4 Remote Control & Viewing Platform

The remote control and viewing platform offers the ability to remotely view the condition monitoring systems status including the identification of any detected faults. It can also show live and analysed data from the system and recall logged data. It has options to control the system which include making adjustments to its settings and

sending requests for it to log data for offline analysis. The single boards RIO's Ethernet port has been used for this link.

The Remote Control & Viewing platform addresses **point 8** of the revised project specifications in **Introduction: section 1.2.**

3.5 Component Interrelationship

Together, the four blocks identified in **Figure 2,** complete the design and functional requirements of the condition monitoring system proposed and addresses **point 3** of the revised project specifications in **Introduction: section 1.2**. **Figure 3** below shows a 3D model of the final system with colour coded boards to show how the previous components fit together.

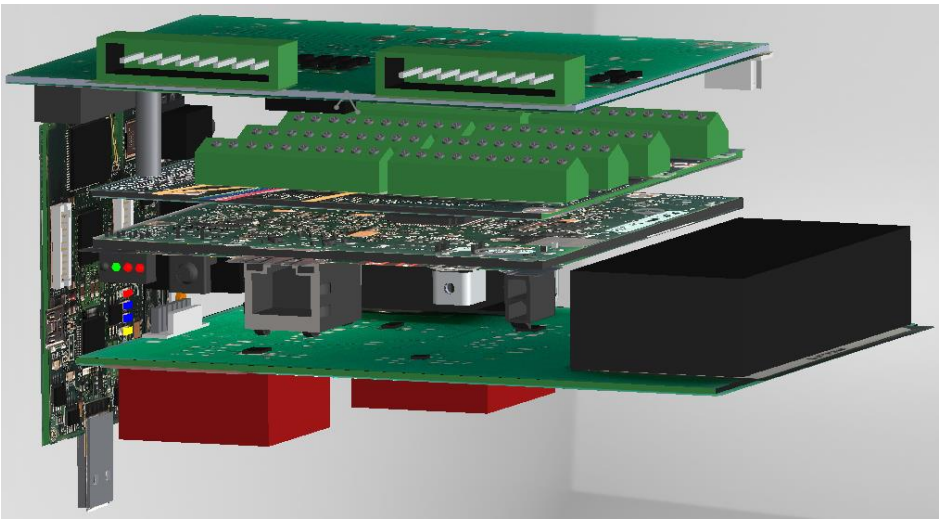


Figure 3: SolidWorks model of final PCB assembly

4. Summary of Remaining Chapters

The following chapters in the report present an introduction to the electrical and mechanical faults investigated and simulated in the project. A description of the short term fourier transform spectral analysis method used, as well as the fault detection algorithm and the selection of the condition monitoring system setting is included in the following chapter.

The report then follows with a discussion on the developed prototype system which is used to collect data and to test the fault algorithm before its final implementation on the embedded system. The report discusses the prototype's use of off-the-shelf data acquisition hardware and a test rig as a mean to simulate a squirrel cage wind turbine generator.

The embedded system design, justifications for it, problems, solutions and decisions taken to reach a final successful product are discussed with high level of detail. The embedded solution contains a customised DAQ hardware, power management board and a controller as well as the final implementation of the fault detection and GUI software.

Finally, the rest of the report includes the management chapter, which describes the overall progress of the team throughout semesters one and two, the technical achievement of the team and the different approaches taken to solve any team issues. Future work required for the system to obtain a wider set of functions, and discussion on the possible commercialisation of the developed product complete the final report.

Technical Objectives

1. Fault Analysis and Detection

Mechanical and electrical faults are the most of the frequent, damaging and expensive type of faults in wind turbine generators. Vibration analysis of the machine has been used in industry for many years as a way of identifying both types of fault; but now industrial companies and research organizations are starting to look and analyze the current output of the generator in the search for fault indicators. Spectral (Fourier) analysis along with fault detection algorithms have been used by the team to build a CM system that replicates the fault detection through vibration analysis and adds the current output analysis' new feature for electrical fault detection. The following section of this document details the signal analysis, explains the results and presents the conclusions of all the work undertaken by the fault detection and analysis team in this project.

1.1 Literature Review

A wind turbine generator is subject to many types of electrical faults in different parts of the machine. H. Henao et al [2] present a wide range of possible failures in wind turbines, the frequency of occurrence and their possible effects and downtime. Kevin Alewine's et al [3], however, takes a closer look at electrical faults, their causes, effects and present much more specific information on them. For instance, the broken rotor bars causing an increase in current density around the rest of the rotor.

The Short Term Fourier Transform for data processing and extraction of the spectrum was investigated in semester one and described in the interim report (**Appendix G2: Interim Report**). Further reading on the spectral analysis and windowing methods were practically inspected by the team for the collection of good frequency and time resolutions in the results.

Research done by the University of Manchester [4] (see **Acknowledgements**) provided the necessary fault characteristic equations for each electrical fault. The fault detection algorithms were coded based on the research carried out by Sandy Smith, Damian Vilchis-Rodriguez et al [5] which suggest the method of monitoring the spectral energy of the characteristic fault frequencies is an effective way of detecting faults on vibration and current signals.

Other fault detection techniques were found; such as pendulum oscillation [6] and investigating the fault frequencies present in the spectrum of the negative sequence component of the current output of the generator [7] are recommended for these types of faults in squirrel cage generators.

1.2 Electrical Faults

There are several causes for electrical faults such as winding short circuits, voltage sags, mechanical faults which could also lead to generator electrical faults and heavy stress in the machine [3], [8], [2].

1.2.1 Turn-to-turn Faults

This fault has been investigated in the project for Wire Round Induction Generators (WRIGs) and Squirrel Cage Induction Generators. The analytical fault frequency equations given below for WRIGs are the same that for Doubly Fed Induction Generators (DFIGs) and therefore the conclusions and analysis obtained during the project for those types of generators are equally valid both WRIGs and DFIGs.

Turn to turn faults could be present in the stator/rotor windings for DFIGs and WRIGs and in the stator windings for a squirrel cage generator. They are usually caused by the converter's high switching voltages and high

frequency harmonics leading to an increase in losses in the generator coil. This would produce an increase in the thermal stress in the machine's windings. This thermal stress ends up breaking the turn of the winding and creating a short circuit, which will usually result in the creation of an open circuit due to the winding quickly burning down [9], [8].

The short circuit in the winding has negligible resistance and, therefore, it results in high current passing through it and causing insulation degradation; ultimately leading to more turn to turn faults in the winding [10] (**Appendix G2: Interim Report**).

1.2.2 Broken Rotor Bars

Broken bars in the squirrel cage generator's rotor can be caused by several stresses, such as environmental, thermal, mechanical or magnetic stresses or a combination of these acting on the rotor. Broken bars or breakage of the end rings in a rotor result in overheating of the generator due to the resultant increased current in the rest of the squirrel cage rotor bars. Conductor damage will be induced in the laminations around the bar due to arcing across the crack and, therefore, the resulting healthy bars or the rotor will carry more current than expected and will be affected by a greater stress. Rotor faults are also quite common in AC induction machines; they represent around 5 to 10% of all the problems that appear in them [11]. This research has developed from work carried out in semester one, see **Appendix G2: Interim Report**.

A broken rotor bar would produce specific fault frequencies in the spectrum related to the rotor speed and the fundamental of the signal. These frequencies can be determined by **Eq 2** (**section 1.2.5**). Research papers studied outline the usefulness of analyzing radial vibration signals of the generator coming from the accelerometer rather than axial vibration signals [11].

1.2.3 Electrical Fault Simulation

To simulate breakage of rotor bars, the rotor was taken out of the team's generator (**Appendix A1: Bearing Study and WTCMS Generator Information**) and had a hole drilled perpendicular to the bar in the university's mechanical workshop (**Figure 4**). The depth and width of the hole was made sufficiently large to remove all the electrical contact when breaking the bar into two separate pieces. One broken bar was thought to be sufficient for the simulations, but if in the future, the team needs to further aggravate the fault, another broken bar could be done by drilling another hole and increase the stresses on the rotor.

Figure 4: Simulated rotor broken bar

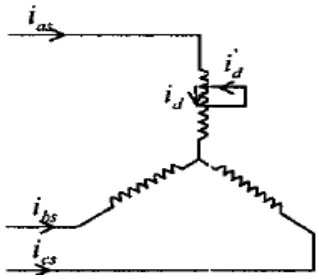


Figure 5: Short circuit winding

To simulate a stator short circuit in a squirrel cage generator (**Figure 5**) and ensure the safety of the procedure, the team used the MarelliMotori (**Table 2, section 1.6.1**) in the A1 laboratory of the University of Manchester. The MarelliMotori generator is significantly larger, with much larger ratings, than the team's squirrel cage generator. The windings of the generator were formed by several parallel paths that allowed easier manipulation and fault recreation conditions. Since both generators are of the same type, the calibration and conditions would be the same for both once the particular parameters for each generator were applied to the software. Details on the procedures and results can be seen in the **Technical Objectives: section 1.6.**

Unbalances in the stator windings of the machine can also be caused by a phase voltage unbalance. If a phase voltage unbalance occurs in the system, the electricity provider must ensure that this unbalance doesn't exceed 1% in England and Wales and 2% in Scotland of the phase voltage [12]. The generators would be connected to the electrical supply through a converter and therefore the unbalance should be dealt by the converter and not by the CM system. In this way, the problem of distinguishing between the phase voltage unbalance and a short circuit fault in one of the windings (which have very similar effects on the generator) would be solved.

A rotor/stator unbalance in one of the windings can produce severe damage to the machine if the fault goes unnoticed or the machine protection mechanisms (for example: overcurrent relays) don't operate fast enough (more details in the **Technical Objectives: section 1.11**). Due to faster operation of the protection relays that simply switch the generator on and off compared to the CM box that captures and processes data before choosing a solution, short circuit fault analysis is used more as a diagnosis tool for the maintenance engineer once that the fault has happened than as a protection mechanism against wind turbine damage (**Appendix G2: Interim Report**).

1.2.4 Fault Frequencies Characteristic Equations for Electrical Faults

Research at The University of Manchester [4] has defined a set of frequency characteristic equations (**Figure 6**) for each possible type of electrical fault in both Doubly-Fed Induction Generators (DFIGs) and Squirrel Cage Generators. These equations are the core of the team's fault detection software; they determine whether there is a fault in the system or not by looking at the spectral content of these characteristic frequencies.

DFIG/WRIG Characteristic Equations [4]**:**

WINDINGS		SUPPLY		INDUCED DFIG STATOR CURRENT FREQUENCIES
STATOR	ROTOR	STATOR	ROTOR	
BALANCED	BALANCED	BALANCED	BALANCED	$f_{ind}^{k} = \left\|6k(1-s) \pm 1\right\| f$
BALANCED	BALANCED	BALANCED	UNBALANCED	$f_{ind}^{k} = \left\|6k(1-s) \pm 1\right\| f$ $f_{ind}^{k} = \left\|6k(1-s) \pm (1-2s)\right\| f$
BALANCED	UNBALANCED	BALANCED	BALANCED	$f_{ind}^{k} = \left\|\frac{2k}{p}(1-s) \pm \left[s + \frac{1-s}{p}\right]\right\| f$
BALANCED	BALANCED	UNBALANCED	BALANCED	$f_{ind}^{k} = \left\|6k(1-s) \pm 1\right\| f$
UNBALANCED	BALANCED	BALANCED	BALANCED	$f_{ind}^{k} = \left\|\frac{2k}{p}(1-s) \pm 1\right\| f$

Figure 6: Fault Frequencies Characteristic Equations

k = 0 → Stator excitation frequency; k > 0 → Speed-dependent high frequency components
p= number of pole pairs; f= supply frequency; s =generator slip
(Synchronous speed assumed to be 1500rpm for the generators used in the project)

1.2.5 Squirrel Cage Generator Fault Characteristic Equations

Stator Unbalance [13]

$$f_{sqr}^{k} = \left| k \pm i * \frac{q}{p} * (1 - s) \right| * f \qquad \textbf{Eq 1}$$

If the machine is powered up from the mains directly, then the converter supply harmonic (k) would be equal to one and there won't be any other converter harmonics in the system. This is not the case since a converter is

being used to power the machine but, having said this, the change in the energy of the frequencies is most noticeable in the first harmonic of the converter and therefore a value of one will be used for testing.

Rotor Unbalance [13]

$$f_{sqr}^{k} = \left| k + \left[\pm i * \frac{q}{p} \pm \frac{2n}{p} \right] * (1 - s) \right| * f \qquad \textbf{Eq 2}$$

k =converter harmonic → k = 1,2,3… (1 is the fundamental); i= rotor slotting → i = 0,1,2,3…
q= number of rotor bars; p=Number of pole pairs; n= Rotor mmf harmonic → n = 0,1,2,3…
f= Supply frequency; s= generator slip
(Synchronous speed assumed to be 1500rpm for the generators used in the project)

1.3 Bearing Faults

Bearing faults are the most common faults in a generator. They can be categorised in single-point defects and distributed faults (**Appendix G2: Interim Report**). This report focuses on single-point defects.

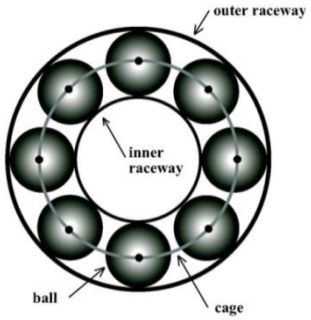


Figure 7: Diagram of bearing [14]

Single point defects are further categorised with the location of the fault. There are four essential parts of bearing where faults can occur (**Figure 7**):

1. Outer raceway defects
2. Inner raceway Defects
3. Ball Defects
4. Cage Defects

Each location has a characteristic frequency which is determined by equations.
The following are the vibration characteristic frequency equations:

Cage Fault

$$F_{cf} = \frac{1}{2} F_r \left[1 - \frac{B_d}{P_d} \cos \beta \right] \textbf{[15]} \qquad \textbf{Eq 3}$$

Outer Raceway Fault

$$F_{orf} = \frac{N_b}{2} F_r \left[1 - \frac{B_d}{P_d} \cos \beta \right] \textbf{[15]} \qquad \textbf{Eq 4}$$

Inner Raceway Fault

$$F_{irf} = \frac{N_b}{2} F_r \left[1 + \frac{B_d}{P_d} \cos \beta \right] \textbf{[15]} \qquad \textbf{Eq 5}$$

Ball Fault

$$F_{bf} = \frac{D_p}{2D_b} F_r \left[1 - \frac{{B_d}^2}{{P_d}^2} \cos^2 \beta \right] \textbf{[15]} \qquad \textbf{Eq 6}$$

F_r=speed of rotor, N_b=number of balls, B_d=diameter of ball, P_d=ball pitch diameter, ϐ=ball contact angle

During a fault the magnitude of the characteristic frequency increases. This can be seen in the vibration spectrum of the bearing.

The change in the vibration spectrum also translates on to the stator current spectrum. The vibrations cause air-gap eccentricities. That causes disturbances in the air-gap flux density which changes induction. This affects the stator current (**Appendix G2: Interim Report**). The following equation is used to determine the stator current characteristic frequency:

$$\boldsymbol{F_{bng} = \left| F_e \pm m F_v \right|} \quad \textbf{[16]} \qquad \textbf{Eq 7}$$

F_e=grid frequency, F_v=characteristic frequency

The equation above (**Eq 7**) shows that the stator current frequencies are dependent on the vibration characteristic frequencies.

1.4 Spectral Analysis

1.4.1 Short Term Fourier Transform (STFT)

Due to the stochastic nature of the wind, the rotor in any wind turbine generator will move at a constantly changing speed. The electrical grid supply is assumed fixed in most of the cases but, in reality, it is slightly varying to a value around 50 Hz which also affects the accuracy of the fault characteristic equations results (**Figure 6**).

The Fast Fourier method of spectral analysis assumes the periodicity of the processed signals and therefore it is not an appropriate method of signal processing through spectral analysis for non-periodic signals such as the current and vibration signals obtained from a generator whose output and condition changes with the wind and the electrical supply. To solve this problem the Short Term Fourier Transform has been implemented in the team's signal processing and fault detection software [17]. The STFT uses predefined periods of time, called windows, where the speed of the rotor and the supply frequencies are assumed constant and applies Fast Fourier analysis only for a short period of time (determined by the window length). Other methods of fault detection (see **Technical Objectives: section 1.11.2**) have been researched but Fourier analysis was the method that provided higher frequency resolution. The STFT relies on the assumption that during a single time window, the signals that it processes are periodic.

$$\boldsymbol{STFT\{x(t)\}(\tau,\omega) \equiv X(\tau,\omega) = \int_{-\infty}^{\infty} x(t)\omega(t-\tau)e^{-j\omega t}dt} \qquad \textbf{Eq 8}$$

w(t) = Window function; x(t) = Signal to be transformed; X(τ,ω) = Fourier Transform;
x(t)w(t-τ) = Function representing the magnitude and phase of the signal; ω = Frequency; τ = time

1.4.2 Windowing

The STFT truncates the given window with specific functions to obtain better resolutions when processing the signal. These functions are defined window types mathematical defined and used for many applications. Some examples are: Hann, Rectangular, Hamming, Blackman window types [18]. The software would allow the user to choose between several window types for signal processing to obtain a better result. Once the signal is truncated, Fourier analysis is applied to the resultant window signal for further processing. The length of the time window is the number of samples to which the window will apply Fourier analysis. Once Fourier analysis has been carried out in the first window, the process continues and Fourier analysis is performed subsequently until the generator stops (**Appendix G2: Interim Report**).

The outcome of the spectral analysis part of the software would be used by the fault detection algorithm coded both in MATLAB and LabVIEW. The test document for the algorithm providing the spectral analysis part of the project can be found in **Appendix A2: Testing of the STFT Algorithm**.

1.4.3 Spectral Leakage

If a 10 Hz sinusoidal signal is sampled at a determined frequency of 64 Hz and the number of samples taken is 64 data points, then the spectral resolution of the sampled signal would be 1 Hz. If instead the number of data samples taken was 32, then the spectral resolution of the signal would be 2 Hz [19].

$$Spectral\_resolution = \frac{Sampling\ frequency}{Nº\ of\ Samples} \qquad \textbf{Eq 9}$$

Poor spectral resolution usually results in the spectral energy of specific frequencies smearing out amongst discrete harmonics. See bottom left graph of **Figure 8**. Eg: If the spectral resolution of a 9 Hz sinusoidal signal is 2 Hz, then the spectrum of the signal will spread over the bandwidth between 8 Hz and 10 Hz. This is because the Discrete Fourier Transform cannot represent the frequencies that lie between multiples of the spectral resolution [19]. This phenomenon is called spectral leakage and can be reduced by using narrower time windows (increasing the time resolution). However, as explained before, Heisenberg's uncertainty principle will apply (**Appendix G2: Interim Report**).

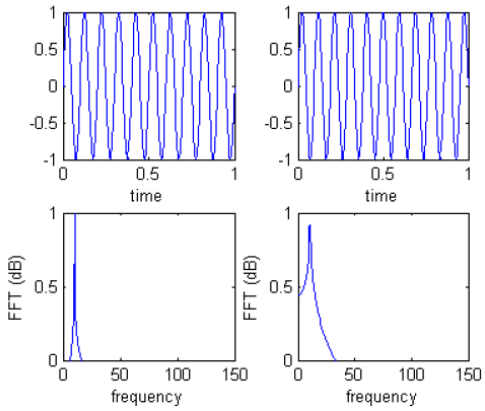


Figure 8: Comparison between periodic (left) and non-periodic (right) FFT signals (LDS_Group, 2003).

1.4.4 Frequency and Time resolutions

The length of the window should be short enough to make sure that the frequency components present in the signal are effectively tracked through time but it should also be wide enough to have the spectral content of the signal well defined within a narrow frequency band (**Appendix G2: Interim Report**). The sampling frequency needs to be taken into account since a time window with a large number of samples could not be sufficient in order to obtain good frequency resolution if the sampling frequency is too high.

The larger the time window, the more data samples there are to process and the better the spectral content of each frequency is known due to the large quantity of data defining it (**Figure 9 (A)**). On the other hand, when a smaller time window is used the opposite happens (**Figure 9 (B)**). Data for a smaller period of time is used, the spectrum gets updated more frequently and the change through time of the characteristic frequencies is monitored at a better resolution.

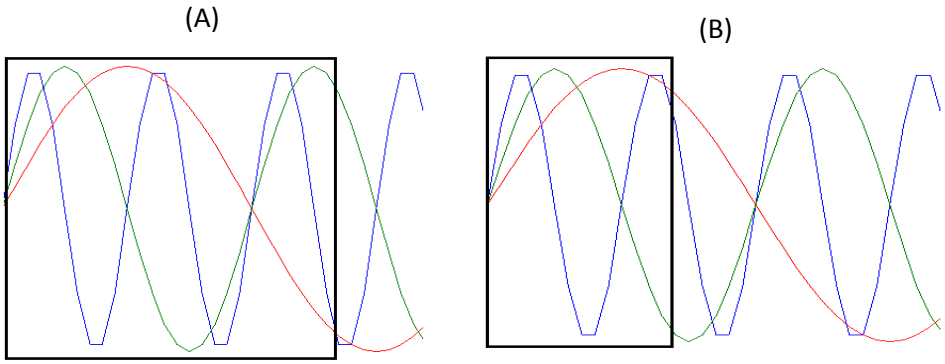


Figure 9: (A) Large time window; (B) Small time window

Example

A non-periodic signal with several frequency components has been recorded for a determined period of time. According to the sampling rate and the period of time, 8192 samples have been taken. If a large window length of 1024 samples is considered, then high spectral leakage would be present in the results if the frequencies in the spectrum change considerably with time. In terms of time and frequency resolution:

- Due to the large size of the window, a significant portion of the non-periodic signal has been taken at an instant (each window is regarded as an instant where the signal is considered periodic), thus the spectral content of many frequency components should be more defined. This means good frequency resolution of the spectrum. However, this would only be the case if the signal is periodic. Otherwise high spectral leakage would be present.
- 8192/1024 = 4 windows have been taken (ignoring the overlap in this illustrative example). This means that we can only see the frequencies of the signal 4 times as time passes and that there will only be 4 records that define the change in the magnitude of the frequencies in the spectrum through time. This means that the spectrum has poor time resolution. The change is not well monitored and the energy content of the frequencies spreads across the spectrum. High spectral leakage if the fault frequencies change considerably with time.

The following figures represent the frequency spectrum of a chirp function with different window lengths and illustrate the previous example. **Figure 10(A)** shows very poor time resolution were the change in the frequencies in the spectrum is monitored approximately every 0.1 seconds which is a large period of time when looking at the rate of change of the signal (see the time bands in the picture clearly seen in blue color). **Figure 10(B)** shows the same data but processed using a smaller time window. The time bands are now much smaller than before and the spectral content in them is not very well defined due to a significant decrease in the frequency resolution of the signal. Due to the changing nature of the frequencies of the chirp function, **Figure 10(A)** presents high spectral leakage that offsets the good frequency resolution results and that is why the frequencies in the spectrum do not appear with a high definition.

(A) (B)

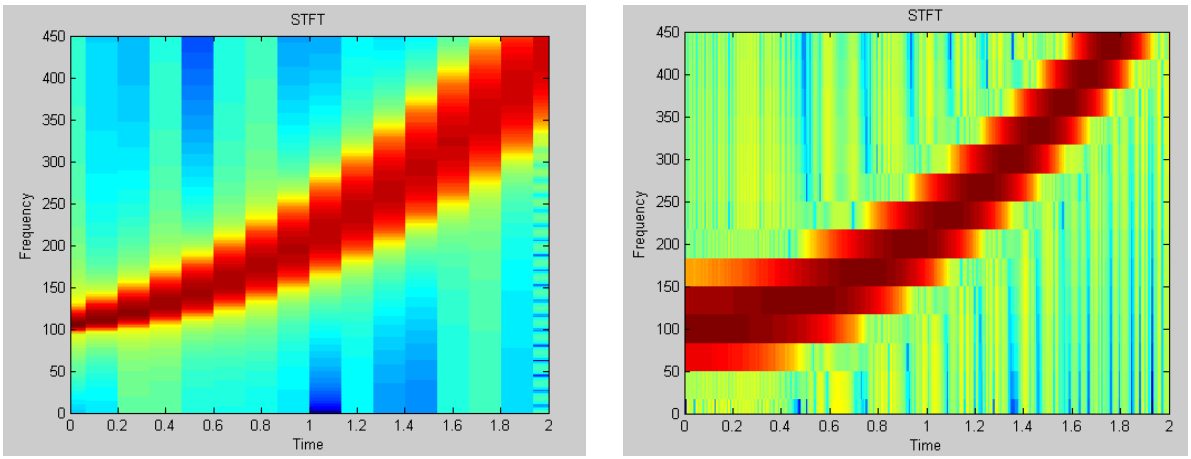


Figure 10: (A) Good frequency resolution, high spectral leakage; (B) Good time resolution, low frequency resolution

To select the window length:

The number of samples taken per window cannot be low (when compared to the sampling frequency) if high frequency resolution is desired. Otherwise the results would have very poor frequency resolution. Eg. If the number of samples per window is 2048 and the system's sampling frequency is 25 kS/sec (therefore the system's Nyquist frequency would be 12.5 kS/sec according to the Nyquist theorem), then our frequency resolution would be:

- 12500/2048 = 6.1035 Hz – Poor Frequency resolution.
- 25 kS/sec = 25kHz sampling frequency = 1 sample every 0.00004 seconds. Therefore the change of the frequencies through time will be monitored every 0.08192 (2048*0.00004) seconds - Good Time resolution.

If instead, a number such as 2^{15} = 32768 data samples per window was selected; the frequency resolution of the results would improve to 0.3815 Hz but the time resolution would decrease.

Heisenberg's uncertainty principle applies here; in this case the more it is known about the frequency resolution of a signal (wide window, assuming zero spectral leakage which is explained below), the less precisely we will know about the time resolution of the signal (narrow window). A balance is necessary in the system for the good quality results. Different windows would also result in different signal to noise ratios (SNR) or tradeoffs between the main lobe of the signal in the frequency spectrum and the low side lobes attenuation depending on the measured signal. The challenge is to identify the best possible window type and window length for the spectral analysis of wind turbine data at a determined sampling frequency (**Appendix G2: Interim Report**).

1.4.5 Signal Overlap

It is desirable to have a high overlap percentage in the spectral analysis part of the software. This means that a high percentage of the samples in the time window being processed by the software are repeated from the previous window. This is necessary to avoid a result with high spectral leakage and inconsistent frequency measurements through time in the spectrum, which is a key point in a fault detection method that examines fault frequencies. A high overlap increases the time resolution of the entire system, solving problems related to the speed of operation of the CM system. Further explanation of these problems can be found in the **Technical Objectives: section 1.11**.

1.4.6 Envelope Analysis - Hilbert Transform [20]

In vibration spectral analysis, envelope analysis acts like a filter. It eliminates the signal originated from the initial vibration of the machine, facilitating the fault detection after the spectral analysis has taken place. This means that the processed signal is highly "concentrated" with the vibration caused by the fault and not the initial vibration from the machine running at 50Hz in a healthy condition. The signal is cleaner and the effect of faults is also clearer to see. Envelope analysis also clears the signal from the reflection of the power of the negative frequencies in the negative side of the spectrum. This is why envelope analysis is mainly used in vibration analysis rather than the current output analysis for fault detection.

The Hilbert transform *H* of a function *x* can be described by the following equation:

$$\boldsymbol{H(t) = \frac{1}{\pi}\int_{-\infty}^{\infty} \frac{h(x)}{t-x}\,dx} \qquad \textbf{Eq 10}$$

H(t)=Hilbert Transform; h(x)=input function; t=time

Using the Cauchy principle, the following equation can be written as the convolution:

$$\boldsymbol{H(t) = \frac{1}{\pi t} * h(t)} \qquad \textbf{Eq 11}$$

Using now the Fourier transforms convolution theorem this can be obtained by evaluating the product of the transform of h(x) with –i*sgn(x), where I is an imaginary number and sgn is the signum function.

$$\boldsymbol{sgn(x) = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases}} \qquad \textbf{Eq 12}$$

The Hilbert transform becomes then a filter that shifts by –π/2 radians the phases of all the frequency components of its input. To obtain the envelope an analytical equation is formed (**Eq 13**), called Y(t) which is described as follows:

$$\boldsymbol{Y(t) = y(t) + jh(t)} \qquad \textbf{Eq 13}$$

y(t)=input signal; h(t)=Hilbert Transform of y(t); j=complex number

The envelope of the signal y(t) is defined as the absolute value of the analytical equation Y(t).

$$\boldsymbol{Envelope = |Y(t)| = |y(t) + jH(t)|} \qquad \textbf{Eq 14}$$

The MATLAB function "hilbert()" gives as an output the analytical equation and not just the imaginary component of it. Therefore for an input x, the envelope analysis will be performed by using the following command line:

"*Envelope=abs(hilbert(x))*;"

1.5 Fault Detection Algorithm

1.5.1 Introduction

In the event of an electrical or mechanical fault, certain frequencies called fault frequencies in the current and vibration spectrum of the machine increase their energy content. These fault frequencies can be calculated using the fault characteristic equations (**Figure 6**). The energy increase in the calculated fault frequencies of the machine and its harmonics can be monitored through time and, depending on the frequency where this energy increase is produced, the fault can be located in the bearings or the rotor/stator of the generator [5].

The fault detection algorithm is coded in a MATLAB script of 600 lines of length and undergoes three different stages:

- Spectral Analysis
- Capturing the energy of the fault frequencies
- Comparison with pre-defined thresholds

Appendix A3: Fault Detection Software Flowchart presents a flowchart of the fault detection algorithm for a detailed breakdown of the different stages and functions in the fault detection algorithm coded in MATLAB. The MATLAB script can be found in **Appendix S1:** MATLAB **Fault Detection Algorithm.**

Spectral Analysis: The vibration and current signals coming from the generator undergo spectral analysis by means of the Short Term Fourier Transform (**see 1.2 Spectral Analysis**). This spectral analysis, together with the detection algorithm, is performed on a single window of the signal at a time in order to increase the speed of the data processing and to obtain a precise reading for the rotor speed for a given period of time. The characteristic fault frequencies equations for each type of fault are used within this algorithm coded by the fault detection team members.

1.5.2 Energy of Fault Frequencies:

Each of these fault frequencies' energy and their first 8 harmonics are checked in the software for an energy increase through time. The changing speed of the wind directly affects the speed at which the rotor of the generator is moving. Due to the dependency of the fault frequencies on the rotor speed and the supply frequency of the machine, the fault frequencies calculated are constantly changing with time. To solve inaccuracies that could arise from assuming a fixed rotor speed, an encoder measuring at 1024 ppr (pulses per revolution) is used to give and accurate measure of speed for the fault detection algorithm. The grid frequency is also measured with the algorithm developed in semester 1 for more accurate results.

Any measurement with the sensors or with the software produced by the team, includes some error. This error will be present in the fault characteristic equations and, therefore, in the fault detection algorithm. To solve problems related to these inaccuracies, the fault detection algorithm doesn't only monitor the energy in the calculated fault frequency, but also in a frequency band of the fault frequency ± 2Hz. All data can be found in **Appendix S2: Prototype Test Rig Data**. **Figure 11** shows an example of the frequency bands of the calculated fault frequency and its harmonics. The color in the graph represents the magnitude of the different frequencies in the spectrum.

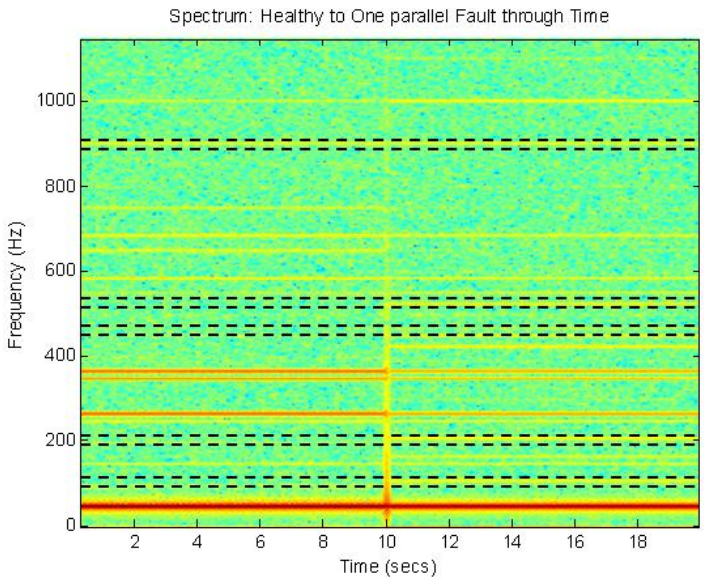


Figure 11: Spectral bandwidths where energy is measured

Bandwidth Calculation:
If sampling frequency Fs= 10 kHz, and the number of samples in one window is window_length/2, eg: 4096. Then the bandwidth where the energy is monitored would be:

$$fs = \frac{10{,}000}{2*4096} = 1.2207\ Hz$$ (but this is only at one side of the frequency)

Thus, the total band bandwidth = 2*1.2207=2.4414 Hz.

The energy in the fault characteristic bandwidths is measured correctly and several tests (see **Technical Objectives: section 1.6**) have been successfully taken to prove its performance. **Figure 12(A)** shows the spectrum through time of a rig running at a varying speed; the previously mentioned changes in the fault frequencies with the rotor speed and supply frequency are visible and the fault frequencies vary through time in contrast with **Figure 11** above where the fault frequencies were constant due to the constant speed of the generator and a fixed value of the supply frequency.

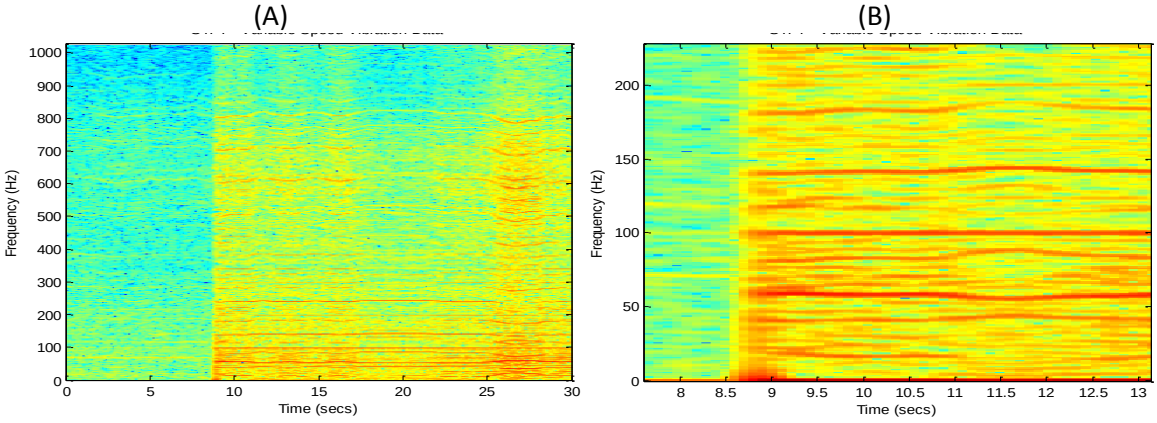


Figure 12: (A) Frequency spectrum of a time varying signal; (B) Zoom in image

By zooming in (**Figure 12(B)**), it can be seen how the fault frequencies vary with time and are not fixed (straight lines) as they are when the speed of the generator and the supply frequency are constant. The frequency band

where the energy is captured will also vary with time effectively capturing all the spectral energy content of these frequencies independently of how large is the change [5].

Energy Thresholds:

A fault in the system will inevitably increase the energy of the fault frequencies given by the characteristic equations by a value between 7 to 10 times the energy of the selected frequencies in the healthy case. This value would depend on the severity and type of fault. When such a change in energy occurs, the fault detection algorithm has been set to give notice of the fault to the operator, including the location of the fault (which would have been identified by looking at the frequency that has increased its energy and the fault characteristic equation that it has been originated from) and the severity of the fault.

The fault detection algorithm has been developed and tested for both bearing and electrical faults successfully showing the expected results (see **Technical Objectives: section 1.6 and 1.9**). The results on the current spectral analysis performed on the experimental squirrel cage generator in the A1 laboratory at the University of Manchester were particularly successful. The tests identified fault frequencies in a squirrel cage generator for stator short circuits that researchers at the University of Manchester had not seen before (since the university work with fault frequencies in the current spectrum of squirrel cage generator signals is at an early stage now) and they are probably one of the greatest novelties achieved in the project. If the system was aimed to be commercialised, it could be said that the inclusion of previously electrical fault detection discovery and its correspondent fault detection algorithm would be one of the team's unique selling point.

It is also important to note that of the frequencies and harmonics produced by the fault characteristic equations, only a few are fault frequencies (increase their energy when a fault occurs). These fault frequencies always appear in the same harmonic for its characteristic fault type; independently of the rotor speed. The CM system has been calibrated to monitor the energy in these particular frequencies. The different tests on the data collected from the team's equipment where these particular frequencies have been identified are described in the following sections and in the appendices.

1.5.3 Interpretation of Results

The calculated set of frequencies obtained from the analytical equations above (**Figure 6, Eq 1, and Eq 2**) is not only formed by fault frequencies but also by healthy frequencies. Using laboratory data and considering 8 harmonics on several types of faults, it was seen in the working fault detection algorithm that only some frequencies in all the inspected harmonics increased its level of energy when the fault happened.

The previous point was checked and proved by visual inspection of the 3D spectrum (eg: **Figure 11**).The addition of the energy of all of the calculated bandwidths in the spectrum distorted the expected increase in the energy of the machine due to the presence of the healthy frequencies' energy (which did not increase but had a high energy content) (**Figure 13(A)**).

To solve this problem, each individual frequency was inspected (without adding them all together) and then the harmonic indicative of the fault was identified (**Figure 13(B)**). After several tests running at different speeds it was seen that, independent of the generator speed, a particular harmonic (different with each fault type) was indicative of a fault type and it was always this harmonic or this set of harmonics that showed the expected energy increase. It was then decided to monitor only the harmonics which indicated a fault and to calibrate the equipment according to these researched parameters. The overall results can be seen in **Appendix A4: Measured Bearing Fault Threshold Energy Matrices**.

For further justification see **Technical Objectives: section 1.7**.

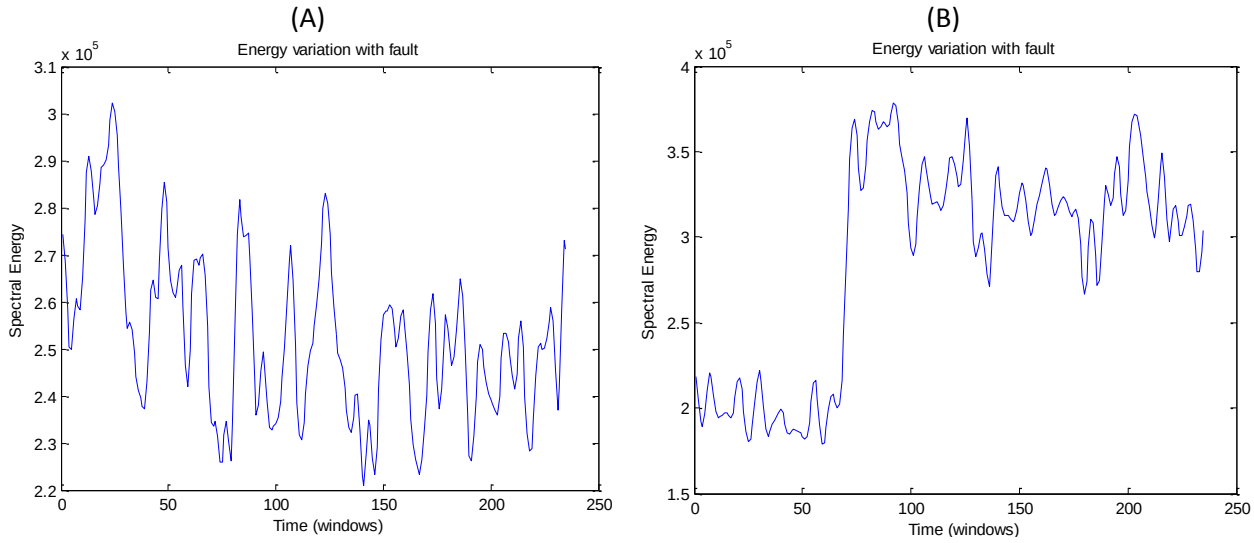


Figure 13: (A) Total energy of all bandwidths; (B) Energy of a single bandwidth

For testing purposes, most of the tests were performed at a constant rotor speed. Otherwise faults could go undetected during development if an actual fault frequency was just outside a calculated bandwidth due to inaccurate rotor speed measurements. The final MATLAB script implemented in the final product would have an input, which comes directly from the sensors, stating the rotor speed for each time window. Tests were also conducted for a large number of time windows (relatively large time. e.g. 20 seconds) while in the final product each time window (small period of time, e.g. 0.5ms) will be processed at a time; allowing quicker processing and faster fault detection (e.g. 0.5ms rather than 20 seconds). Running the generator at constant speed has allowed the team to use the same software in the testing and implementation stages of the project being the final product's software as effective as the testing software.

The following case studies, test results and conclusions from the **0 Technical Objectives: section 1.6** to the end of this technical chapter helped determine the harmonics that are indicative of a fault. Once these were identified the energy measuring bandwidths could be set and the energy thresholds could be measured and defined for the team's squirrel cage generator.

1.6 Electrical Fault Detection by Current Spectral Analysis

1.6.1 Squirrel Cage Generator Case Study

Machine Model:	MarelliMotori EN60034-1 IEC 34-1 A4C 180 L4	**Frequency:**	50 Hz
Phase Voltage:	400 V RMS	**Connection:**	Delta
Phase Current:	24.2 A RMS at 1465rpm	**Nº of Poles:**	4
Rated Power:	22 kW	**Nº of Rotor Bars:**	40

Table 2: A1 Squirrel Cage Generator Info

Stator Short Circuit Fault - Test at Generator Speed: 1530 RPM

Current data was recorded in the A1 laboratory for the previous squirrel cage generator (**Figure 15(B)**) running first in a healthy condition, and then with a short circuit fault in one of the stator windings switched on after 6

seconds (**Figure 14(B)**). The fault through the created short circuit was restrained using large current limiting resistors (**Figure 15(A)**).

Based on **Eq 1**, the software calculated a set of possible fault frequencies. As discussed in **Technical Objectives: section 1.5.3**, all the calculated frequencies were inspected individually to identify the healthy and the faulty ones. In this particular case with the generator type described in **Table 2** above and the generator running at a constant speed of 1530 rpm; the frequency indicative of the fault which recorded the expected energy increase was identified as 1070Hz.

Figure 14(B) was obtained using a Blackman window with a 4096 samples window length (good frequency resolution) to help visual inspection. The 1070 Hz calculated frequency shows a clear increase in energy when the fault happens at approximately 6 seconds.

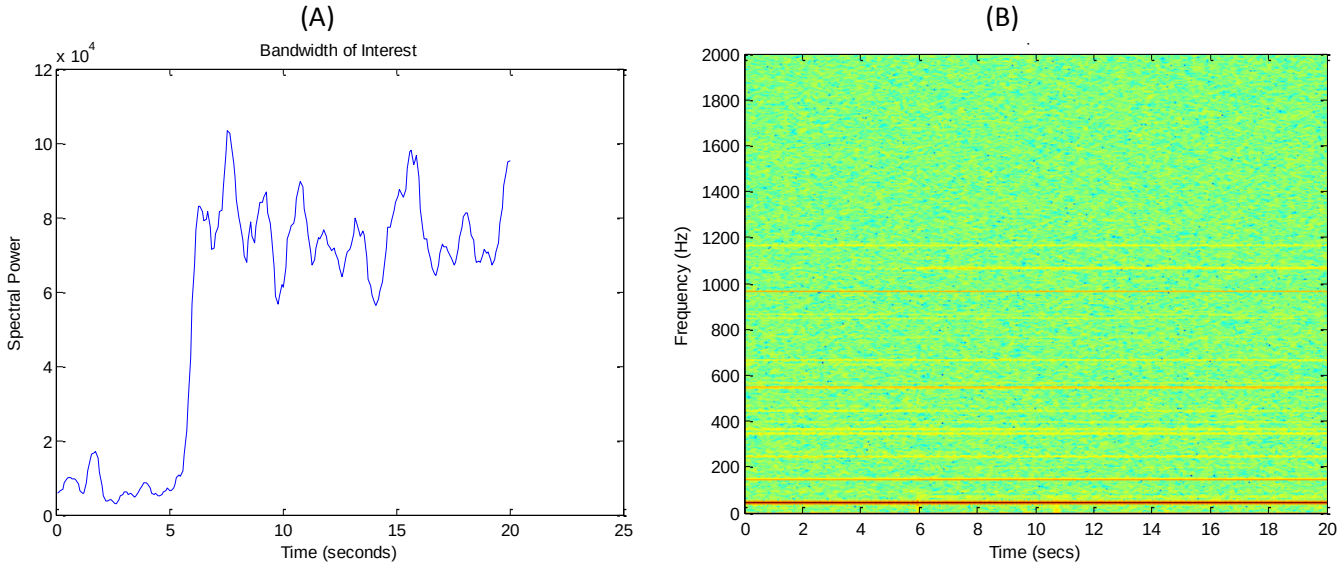


Figure 14: (A) Energy contained in bandwidth at 1070Hz; (B) Current spectrum of generator with stator short circuit fault

After hand calculations and further verification by looking at the calculated variables produced by the MATLAB software, it was determined that the 1070Hz fault frequency is originally from the first time harmonic (i=1) in the addition case, see **Eq 1** for further reference. Converter harmonic in this equation is assumed to be the fundamental (k=1) due to the clarity of the fault effects in the frequencies in the spectrum.

The energy of the identified fault frequency is seen to increase visually in **Figure 14(B)** above but to obtain further mathematical verification of this energy increase the energy of the bandwidth around the 1070Hz component was obtained and plotted through time. The results of **Figure 14(A)** (energy of individual frequency above) are very clear and confirm the energy increase in the mentioned harmonic during a short circuit fault. To confirm these results, further tests were undertaken at 1510rpm and 1550rpm.

(A) (B)

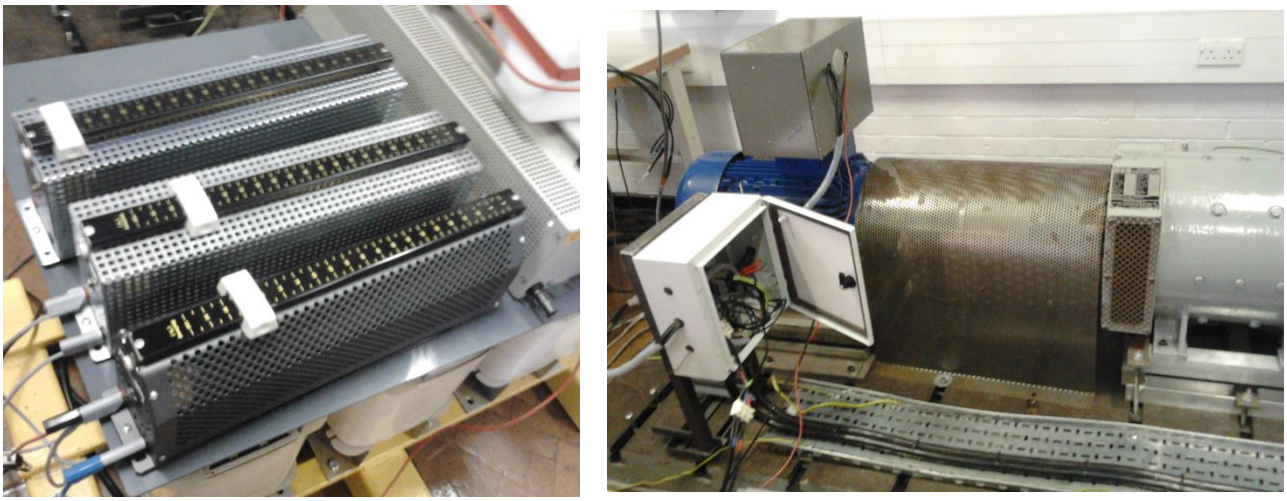


Figure 15: (A) Current limiting resistors; (B) MarelliMotori Generator (Table 1)

Stator Short Circuit Fault - Test at Generator Speed: 1510 RPM and 1550 RPM

Using the previous parameters and calculating the fault frequency by using the first converter harmonic, the first time harmonic in the addition case in **Eq 1** $\boldsymbol{f_{sqr}^{k} = \left| k \pm i * \frac{q}{p} * (1 - s) \right| * f}$; the fault frequencies calculated were:

$f = 1056.7\ Hz$ for the generator running at 1510 RPM

$f = 1083.3\ Hz$ for the generator running at 1550 RPM

The frequency-time-magnitude diagrams for the generator running at 1510 rpm are as follows:

(A) (B)

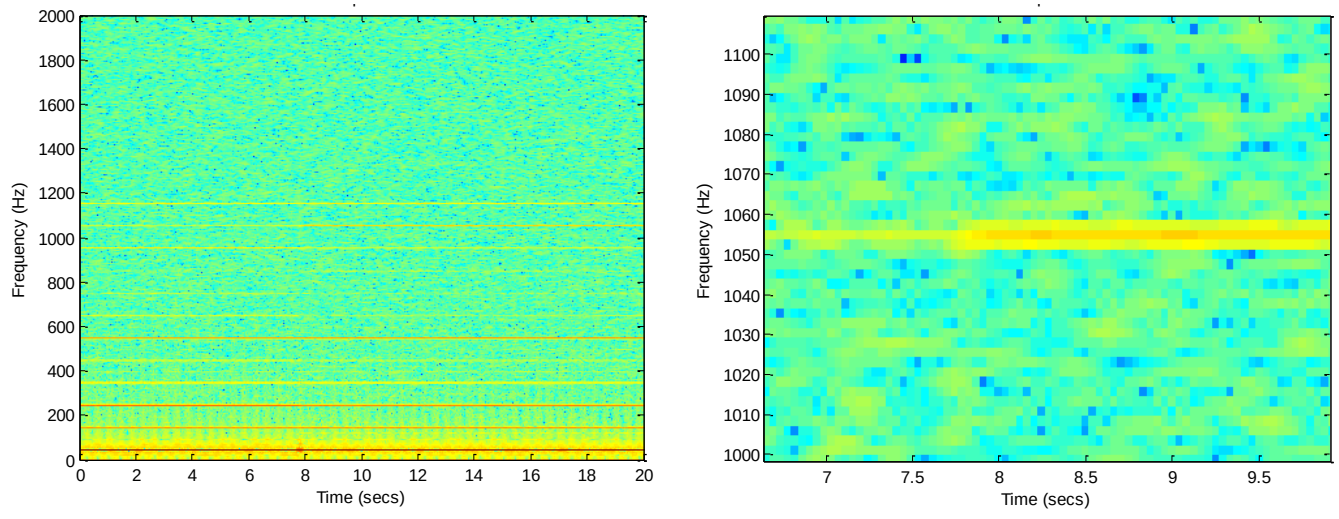


Figure 16: (A) MarelliMotori current spectrum at 1510rpm; (B) Zoom in image of fault frequency

The same window parameters as in the 1530rpm case were used. With the machine running at lower speed than in the previous case, the magnitudes of the fault frequencies in the event of a fault decrease their value. Therefore it is harder to see the change in energy in **Figure 16(A)** but if zooming on 1056.7Hz (**Figure 16(B)**), the change becomes clear. The individual bandwidth energy diagram is similar to the one depicted in **Figure 14(A)** but

the magnitude of the energy present is lower since the generator is running at a lower speed and carries less energy.

Similarly, by inspecting the 1083.3Hz frequency calculated using the first time harmonic of the signal at 1550 rpm (using the first converter harmonic as well), it is possible to see the expected increase in the spectral content of that particular frequency in the spectrum (**Figure 17(A)**). The expected energy increase when the fault happens is fairly clear as well in **Figure 17(B)**. Notice that the speed of the generator is higher and, therefore, the energy of the signal in **Figure 17(B)** is higher than in **Figure 14(A)**.

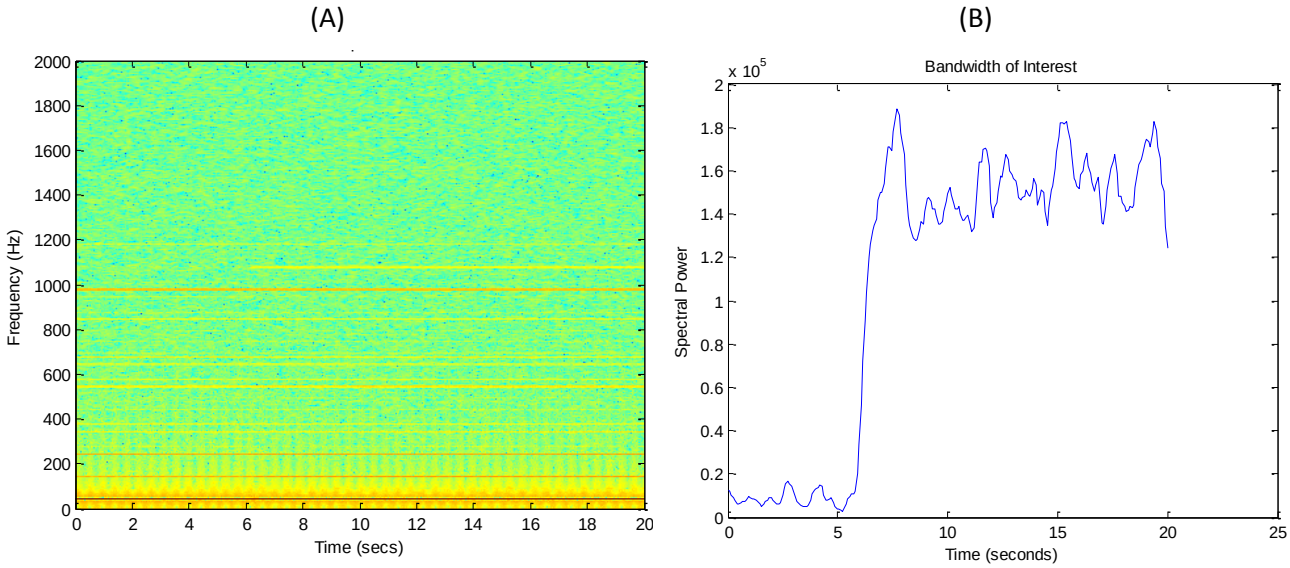


Figure 17: (A) Current spectrum of generator at 1550rpm; (B) Energy variation of 1083.3Hz

For a more conventional representation of the spectrum, both the healthy and faulty spectrums have been plotted against each other. A 2D representation is possible because the generator is running at constant speed and, therefore, no spectral leakage will be produced. **Figure 18** zooms into the 1083.3±2Hz fault frequencies in the selected bandwidth rising with the fault frequency to see this magnitude increase more closely.

In summary, no stator unbalance fault detection on squirrel cage generators has been investigated before at the University of Manchester using spectral analysis of the current output of the generator. The results have clearly shown the success and novelty of this method of fault detection in squirrel cage generators and, when talking about the commercialization of the product, this type of detection along with the rotor unbalance fault detection using current analysis could very easily become the unique selling point of the WTCMS at the end of the year. A second case study for stator unbalance using fault detection on DFIG and WRIG types of generators has been included in **Appendix A5: WRIGs Case Study**. The team has provided software allocations so that the fault detection algorithm would not only be applicable in squirrel cage generators, but also to DFIG and WRIG types of generators if required. The user would be able to select the generator type in which he/she will install the CM product which greatly improves the flexibility of the final system as well as extends its market current market place.

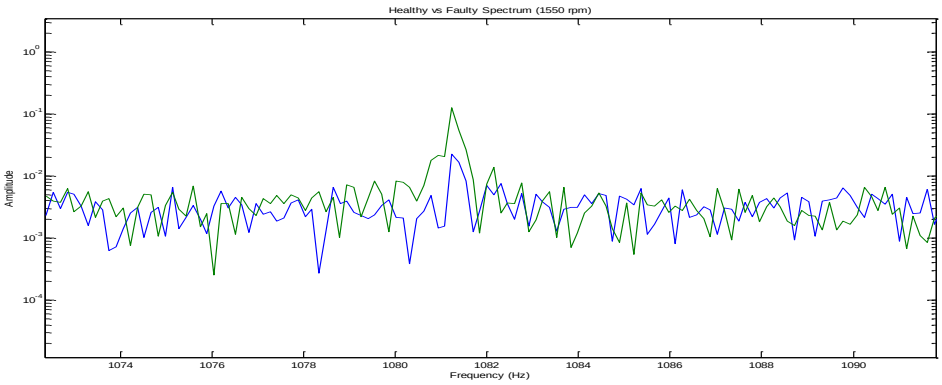


Figure 18 - Zoom into fault bandwidth

1.7 Energy of the Machine – Observations

When a fault occurs, the generator increases its vibration level to an abnormal condition. The vibration of the machine in the faulty condition will be much larger than the vibration of the machine when running in a healthy condition. Therefore there will be a large energy difference between the fault frequencies in the healthy and faulty conditions when looking at the frequency spectrum of vibration data. However, when analyzing the output current of the squirrel cage generator data this will not always be the case.

Sometimes, the energy in the machine during normal operation is larger than the energy of the machine during a fault (e.g. when the fault reduced the voltage of the system). However although this might be the case in the overall energy of the spectrum, there will always be specific frequencies that see an energy increase with a fault. These fault frequencies are the main focus of the project but there are also other frequencies calculated with the analytical equations with different behavior, for example: the fundamental frequency.

1.7.1 Explanation

When a fault such as a short circuit fault in one of the paths of the stator windings occurs, the generator will need more current to maintain its speed and keep supplying the load. During tests, a winding path was short circuited which meant less turns in the winding. As a result, the generator demanded higher current from the grid to keep its phases balanced and this lead to a supply voltage drop. Therefore, the energy carried by the fundamental frequency dropped. This is a typical fault scenario when a fault occurs in a power network or a single generator.

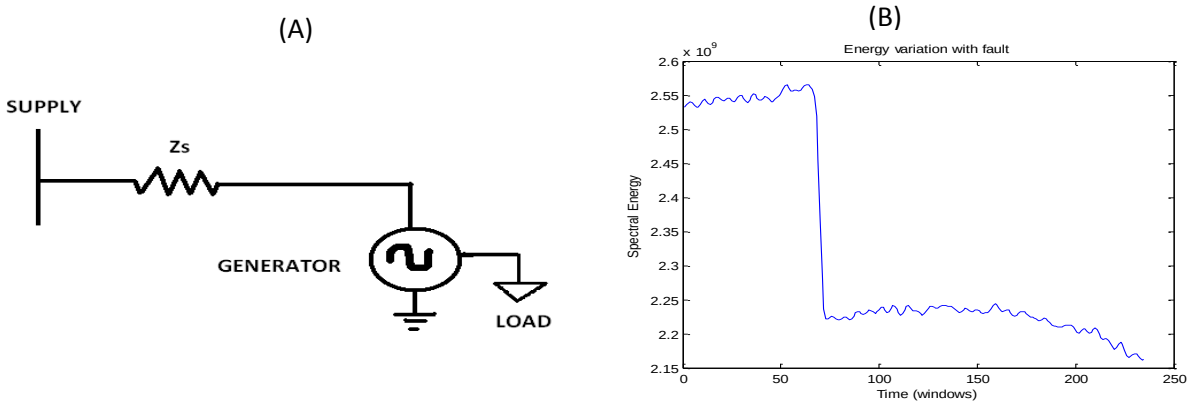


Figure 19: (A) Impedance diagram; (B) Energy of fundamental frequency

Higher current demand means increased losses in the impedance between the power supply and the generator (Zs, **Figure 19(A)**) and therefore there will be a voltage drop at the generator terminals when a fault occurs. In

summary, the interaction between the supply impedance and the demanding power at the terminals of the generator causes a voltage drop which is reflected in the energy of the 50Hz fundamental frequency (**Figure 19(B)**). This voltage drop in the machine terminals during a fault are the reason that the energy of certain frequencies in the output current spectrum of the generator is higher when the machine is healthy than when the machine is faulty.

The fundamental is the frequency component that carries higher energy in the whole spectrum. Looking at the analytical fault frequency equations for a squirrel cage generator (**Eq 1**) in a short circuit fault it is possible to spot that the fundamental frequency is one of the frequencies given by this equation. When k=1 (for 50Hz supply) and i=0 the frequency calculated from the analytical equation is equal to 50Hz.This means that the energy of the fundamental (which is by far the largest in the signal) is being summed up in our energy summation algorithm of fault frequency bandwidths (this is characteristic of squirrel cage generators such as the one the team is working with). Therefore the loss in energy, due to the previously mentioned voltage drop, will heavily affect the energy measurement of the signal as a whole. This is another reason to the one given in **Technical Objectives: section 1.5** why the software has been set to monitor the energy of specific harmonics and not the sum of the energy of all the frequencies given by the analytical equations.

Another approach to solve this problem was normalizing all of the frequency magnitudes in the spectrum with respect to the fundamental frequency (divide all frequency magnitudes by the magnitude of the fundamental), therefore eliminating the effect of the fundamental in the summation of spectral energies of the calculated fault frequencies. However, normalization relies in the assumption that the frequencies in the system are proportional to the fundamental frequency which may not always be the case.

1.7.2 Other causes of voltage drop

There are other possible causes for a decrease in the energy of the fundamental frequency. Voltage sag is one of them and, therefore, there must be protection mechanisms against it. This is a real problem for wind turbine owners since a change in voltage would affect the generator's torque, increasing the rotor acceleration and mechanical instability. A voltage stabiliser can be used to solve this problem. This is basically a large capacitor bank that will supply the required voltage for a small period of time (**Figure 20**). However, this is a very expensive solution and it only works for a short time. In theory, the electricity supplier needs to ensure the perfect operation of the line. This type of fault and its effects also reassures the choice of identifying the harmonics indicative of a fault rather than using the summation of all frequencies given by the frequency equations (which include the fundamental frequency and will yield erroneous results).

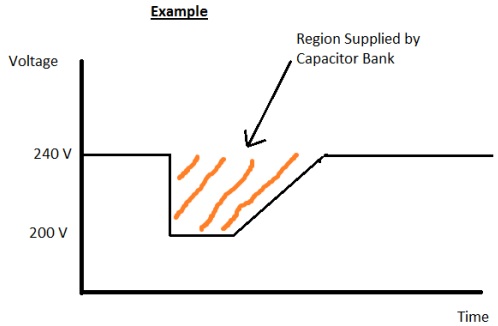


Figure 20: Voltage sag representation

1.8 Bearing Fault Detection by Vibration Spectral Analysis

The principle of operation of the fault detection software for bearing fault detection is the same as the one described in the **Technical Objectives: section 1.5** and the one described in the **Technical Objectives: section 1.6**. The vibration levels of the machine will increase with any type of mechanical fault and therefore the energy of the characteristic fault frequencies will irremediably be seen to increase. The energy of the fault frequencies calculated by the bearing single point defect fault characteristic equations effectively do so when such a bearing fault occurs.

Problems such as the ones described in the **Technical Objectives: section 1.7** will no longer be present due to the different nature of the signals obtained from the accelerometers. The energy increase will be detected by the software and through the comparison between the energy levels in the healthy and faulty conditions the bearing fault would be identified.

The following example shows a 4mm inner race bearing fault simulated in the team's squirrel cage generator purchased by the team (**Appendix A1: Bearings Study and WTCMS Generator Information**). For testing purposes the analyzed signals are a combination of the vibration signals in a healthy and faulty condition coming from the accelerometers mounted in the team's generator and analyzed by the team's software. The generator is running at a constant speed of 1484 rpm. The energy increase in the whole spectrum is clearly visible in **Figure 21** below; where **Figure 21(B)** shows the energy summation of all the calculated fault frequencies in the first 8 time harmonics. This case differentiates itself from the electrical faults because looking at the 8 checked individual harmonics is no longer necessary. The energy increase is present in all of them.

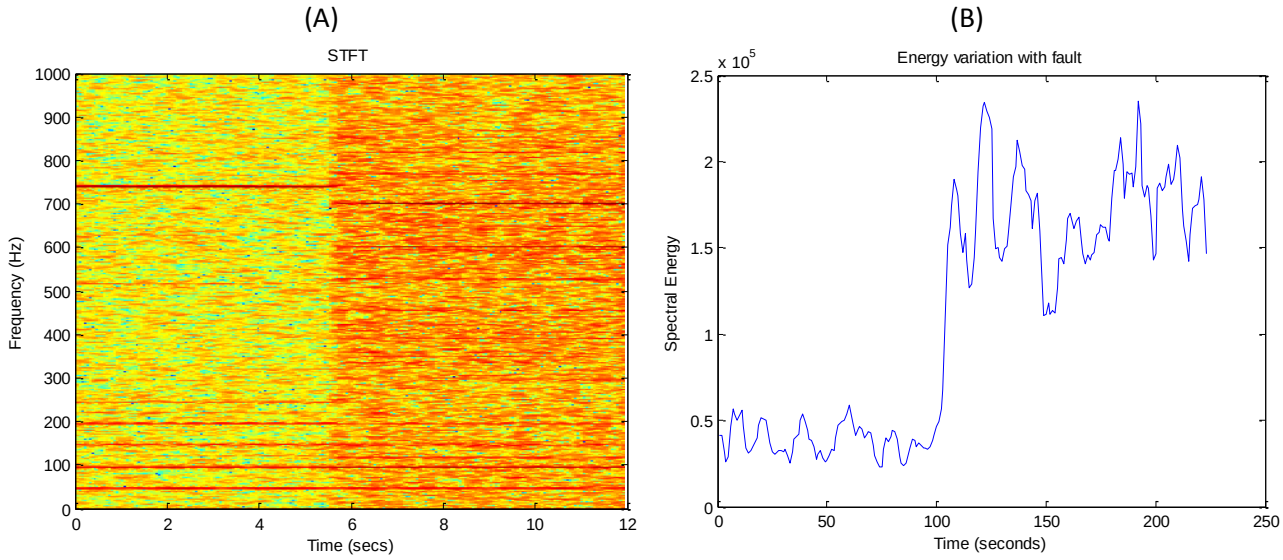


Figure 21: (A) Vibration spectrum with 4mm inner race bearing fault; (B) Energy variation in calculated fault frequencies

The rest of bearing faults are detected using the same approach with the vibration data and using their respective analytical equation. The bearing parameters used in the team's fault detection software are available in **Appendix A1: Bearings Study and WTCMS Generator Information.**

Spectral analysis on the current output of the generator for bearing fault detection was also investigated but the results from the vibration analysis were much clearer and convincing that the results from the current spectral analysis (**Appendix A1: Bearings Study and WTCMS Generator Information**). This fact reinforces the conclusion

that vibration analysis is a much more reliable way of fault detection for mechanical faults than current analysis. Current analysis outputs better results for stator/rotor unbalances.

1.9 Results: Vibration Thresholds

The energy of the thresholds for Inner Race, Outer Race and Cage Fault defect were measured for demonstration purposes in the aim to towards a working prototype and embedded system. Ball defect faults and cage faults present an additional difficulty when establishing these thresholds; their characteristic fault frequencies only raise their energy when the bearing ball rolls over the defect, but this rolling is not continuous. Measuring the energy of these fault frequencies becomes, therefore, a difficult and time consuming task since in most of the tests these fault frequencies will not appear or will do so for a few tens of a second which is not long enough to take accurate readings.

The team is still working on the setting of threshold energy levels for the detection of broken rotor bars in the squirrel cage generator towards the demonstration. **Appendix A4: Measured Bearing Fault Threshold Energy Matrices** shows the experimental tables that formed the energy threshold matrices for bearing faults in the fault detection algorithm. The units are spectral power units. The sampling frequency was selected as 2kHz, thus the resolution comments when the window size is large. When working with different sampling frequencies the ratio of the sampling frequency to the window length needs to be considered. For example, for 2kHz sampling wl= 1024 would give an appropriate result since the changes in the signal would be monitored every half a second when using the STFT. For 10kHz, a larger window of 4096 samples of length would be needed to achieve similar time resolution.

1.10 Relationship with the rest of the project

The Fault analysis and detection software developed addresses point 2 of the revised project specifications in **Introduction: section 1.2.**

The fault detection algorithm makes use of all the processed signals obtained from the hardware (sensors and signal conditioning hardware) to evaluate the condition of the machine. The algorithm is run in a single processor contained in the SBRIO. Vibration analysis is very common in industry and presents very few challenges in terms of fault detection and investigation. Current analysis is more challenging (see **Technical Objectives: section 1.11**) and directly affects the design and requirements of the conditioning monitoring system; particularly the increasing need for fast data acquisition and processing.

The fault detection algorithm has been developed so that is can be embedded into the prototype and embedded system LabVIEW software. All the data acquisition is done in LabVIEW while data processing is performed by the fault detection algorithm coded in MATLAB. Data from the IV transducers and accelerometers are filtered with the DSP board and transmitted using the SPI communications protocol. The data arrives to the SBRIO where it will be processed in data packages by the LabVIEW software which then passes them on to the fault detection algorithm.

The CDAQ prototype system was key in defining the energy threshold matrices for each of the mechanical and electrical faults. Measurements of the machine's energy levels during healthy and faulty operation at different rotor speeds were possible during the embedded system design and construction thanks to the CM prototype and rig built by the team.

The results of the fault detection algorithm are outputted so they can be shown in the LabVIEW GUI developed by the team. These results contain the state of the machine and if a fault is present, the location of the fault would be displayed in the GUI. The fault detection algorithm would also send signals to light up LEDs in the final embedded design in the event of a fault.

1.11 Generation protection- Relay problems affecting the CM system design

Wind turbine generators have many parallel paths that form its windings. These windings carry high currents and voltages. For reliability and flexibility of operation (variable number of turns), they are formed of many parallel paths with a determined number of turns. When an electrical fault occurs; such as a turn-to-turn fault, one of the paths of the winding gets shorted, causing an unbalance in the stator or rotor. This unbalance in one of the paths can cause asymmetries that over a period of time can develop into a more severe fault where the whole winding can be destroyed.

To prevent severe damage, wind turbines have overcurrent relays attached to the generator. If a fault happens these relays are designed to operate in one generator's cycle. This is an advantage for protection of the generator but has problems that arise when trying to locate the fault. The relay's fast operation (milliseconds) conflicts with the system's data acquisition and processing time. There are ways to solve this problem but not much research has been done in this area, for more information see **Overall Summary: section 2**.

1.11.1 Justification for fast operation of the CM System - Example

The overlap in the frequency spectrum analysis (see **Technical Objectives: section 1.4** giving reasons why to use overlap) becomes really important with electrical faults. Assume the machine will start up in a healthy operation. The first time window of 1024 samples length (0.5 seconds is sampling at 2kHz approximately) is acquired from the sensors and then the next window with an overlap of 90% is obtained. Therefore (1-0.9)*1024 = 102 new samples would be obtained with each new window and the rest (90% of the previous time window) would come from the system's buffer. Sampling at 2 kHz (1/2000 = 1 sample every 0.5ms) the system would be obtaining new data every 61ms (102*0.5 = 61) and therefore data would be processed in 61ms, plus the fault detection and spectral analysis processing time. This is the closest that a CM system can get to "real time operation". This overall processing time would be reduced if the overlap is increased as much less new data samples would be acquired each time. The number of windows necessary to sample one second of data (for example), will increase but the objective is sampling the coming window as quickly as possible and not reducing the number of windows used. At this point, the fault detection algorithms in MATLAB interact with the LabVIEW software which provides the required inputs (conditioned signals coming from the generator).

If, for example, the processing time of the software produced is 20ms; then the overall time to output a result and take action will be 61+20=81 ms. When a fault happens (and the generator still running because the fault hasn't developed enough to trip the relay), the CM system will be able to locate the fault (provided 61 ms have passed and the CM system has data to process). If the fault happens and develops before 61 ms, the system won't work since there will be not enough data to process. This is the main limitation when spectral analysis in the generator's current output for fault detection.

However, these types of catastrophic faults are not the only types of electrical faults there are, the results of these faults are very obvious; burns can easily be seen in the windings by technicians using off-line fault detection equipment. The relay cannot be attached to all the parallel paths individually. If a small fault happens in a path of one of the windings and the relay doesn't operate since such a change in current (small unbalance) is at first unnoticed, the generator keeps operating with the fault. The fault will eventually erode the parallel path winding and cause a catastrophic fault. These are situations where the designed software could come into effect to detect and locate the electrical fault. A small electrical fault can also create pulsating torque in the machine and, in time, this can lead to machine failure. This is another reason to aim for very fast operation; which has set strong design requirements and equipment selection for the rest of the parts of project. Bearing faults, develop slowly over a large period of time and therefore they do not need such a fast operation. The algorithm has remained the same for bearing fault detection since fast operation is still an advantage.

1.11.2 Other methods of fault detection

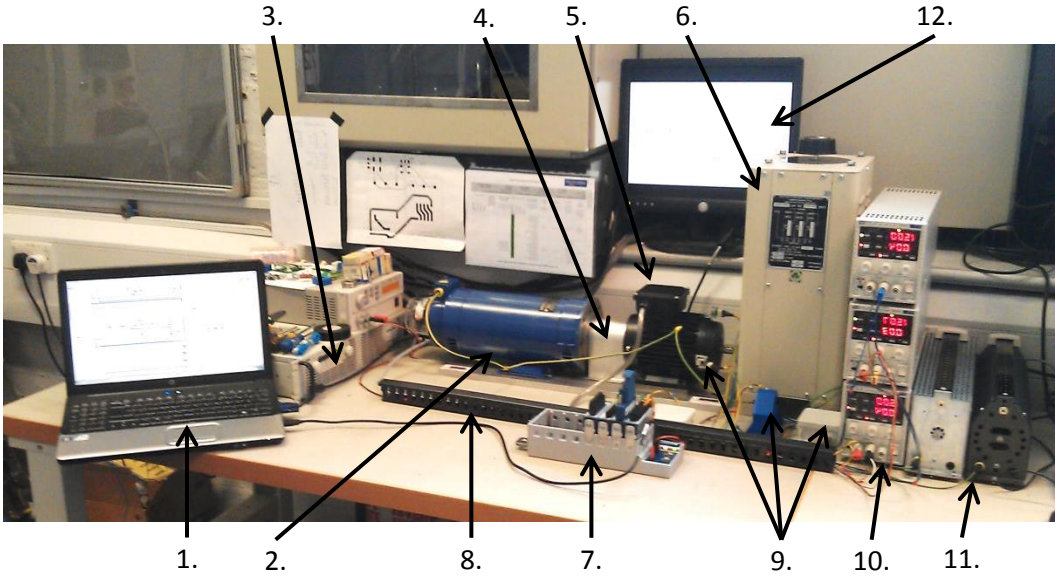
The previously mentioned problem related to the lack of generator data due to very fast (almost instantaneous) relay operation has led the fault detection sub-team to think about using EMD (Empirical Mode Decomposition) and Wavelets as fault detection techniques. The reason these methods have shorter processing times than the Spectral Analysis method using the Fourier transform. The advantage of the Fourier transform is that it gives the system defined frequencies that can be compared to the ones obtained from the analytical equations and, therefore, the fault detection algorithm is easier to implement and to check its accuracy.

Example: EMD resolves the signal into sub-signals with a very simple mathematical algorithm that is quick to process unlike the Fourier transforms. The data can be processed as soon as it comes through the system and doesn't have to wait for the new 102 samples (see **Technical Objectives: section 1.11.1** above, this value is just to illustrate the point) and form a window. Also the mathematics behind EMD is not as complex as for Fourier and this reduces the processing time for EMD [21] [22]. These sub-signals are a mixture of several frequencies and are used in fault detection algorithms but previous research at the University of Manchester has not given positive results and many researchers are very skeptical about the reliability of the results of EMD [23] Therefore, it is known that the processing time is shorter for EMD but the accuracy and reliability of the results from the sub-signals analysis are yet to prove.

1.12 Summary

The fault detection algorithm has been designed to detect stator/rotor unbalances (electrical faults) and mechanical faults such as inner, outer race, ball and cage bearing faults. This algorithm calculates the vibration and current spectrum of the data recorded by the sensors using the Short Term Fourier Transform which uses specific window types and lengths which can be selected by the user within a wide range of options. The fault detection algorithm has been tested using several current and vibration signals in all of the specified signals giving the expected results. The algorithm has been designed to perform a continuous monitoring of the spectral energy content of the fault frequencies calculated by the fault characteristic equations and to detect a drastic increase in their energy when a fault occurs. The team has innovated through the inclusion of the electrical fault detection software which has not been implemented before in any industrial system. The latest laboratory tests have successfully proven the validity of this CM method.

2. Prototype System



1. Laptop running LabVIEW prototype application.
2. DC permanent magnet motor (drive).
3. 160v programmable power supply.
4. Protective coupling guard.
5. Induction machine (generator).
6. Variable 3 phase transformer (connection to grid).
7. NI Compact DAQ (CDAQ) and modules.
8. Sensor power & signal cable loom.
9. Current, Voltage, vibration and temperature sensors.
10. Power supplies for sensors.
11. Power resister in series with one phase
12. 2nd screen for application GUI

Figure 22: Prototype system in operation

To aid the implementation of this project's main aim (**Technical Objectives: section 3)** a prototype system was developed to allow the testing of hardware, software and fault detection techniques. **Figure 22** shows this system with labels for its main parts. The system can be simplified as shown in **Figure 23** as four main blocks connected together in series.

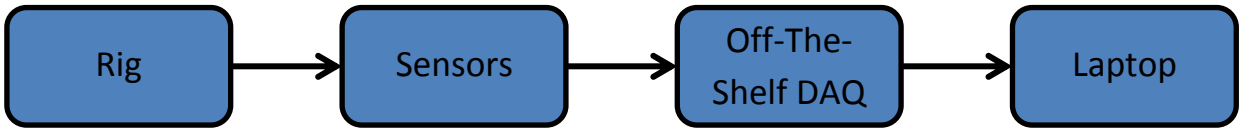


Figure 23: Four block representation of the Prototype System

The rig was developed as part of the prototype. It offers a platform for faults to be introduced to by purposefully introducing defects to its components and for sensors to be attached to for evaluation. As a cheap and scaled down alternative to a real wind turbine generator, a 0.55KW 3-phase squirrel cage induction motor was selected for use in the rig. While there are some limitations that come with it being different to the more complex generators used in most commercial wind turbines, it has allowed a set of similar faults to be introduced to produce data that has aided the development of fault detection techniques. In testing, the induction machine can run below synchronous speed (motoring) with its connection to the grid through a three phase transformer (variac), or it can be turned above synchronous speed (generating) by the DC permanent magnet motor it is coupled with. The DC motor is powered by a power supply which can be programmed to produce a varying speed drive for the induction machine, allowing simulation of varying wind speeds with turbulence.

To get signals from the rig, a number of sensors have been evaluated during development of the project; some are permanently fixed to the rig and some can be swapped in and out so that the quality of their measured data can be assessed. An encoder and two temperature sensors are permanently fixed in position to the rig. For the other measurands of interest, the variac's removable front panel has provided a safe and convenient means for connecting current and voltage transducers for power measurements. For vibration data, four mounting points on the induction machine allow accelerometers to be attached

To interpret the output of the sensors mentioned above, off-the-shelf hardware was selected with customisable signal conditioning and ADCs to allow quick and simple measurements from the sensors. In the final design, this is replaced by the teams own custom DAQ hardware.

The program developed for the prototype running on a PC connected to the data acquisition hardware provides the following functionality:

- Acquire data from sensors making live and buffered versions available for logging and analysis.
- Provide all settings and machine specifications to analysis functions which will include a MATLAB fault detection script.
- Configuration of connected sensors, data acquisition settings, fault detection parameters and data logging.
- A means to set rig specifications making it cross platform.
- Viewing of any raw and processed data simultaneously.
- Automatic logging of data and reports in response to a fault being detected.
- Manual logging of data for offline processing.

This list of abilities serves well as the justification for spending time developing this prototype platform. As both, the prototype and embedded platforms will use the same programming language, much of the code developed for the first, will be ported directly onto the second with much of the specified functionality already implemented. The prototype software has been made with a lean and efficient architecture, as is required for less powerful embedded platforms.

Using off-the-shelf hardware designed for use with the chosen programming language, reduces the complexity of getting data into the computer and allowed all focus to be upon developing the techniques and functionality required for the final system.

2.1 Relationship with the Rest of the Project

The prototype system partially addresses points 6, 7 & 8 and completes points 1 & 9 of the revised project specifications in **Introduction: section 1.2.** While the prototype contains almost all of the code required for these two points, it only runs it on a laptop and not the Single Board RIO.

The prototype system was developed with collaboration from all sub-teams as a stepping stone to the final system. While it has addressed (fully or partially) some of the objectives, its main use has been in helping other sub-teams complete theirs. The points below highlight where each sub-team has had input in the development of the prototype system and where each has used it during testing.

2.1.1 Hardware team

The hardware team aided the design and development of the prototype system through the following points:

- Selected and sourced all power components including motors, cables and power supplies.

- Selected and sourced all sensors for evaluation on the rig including; a range of current transducers, voltage transducers, an encoder, RTDs and a range of accelerometers.
- Design of the necessary interfacing circuitry to connect sensors to the data acquisition hardware.
- Supervised and provided designs for mechanical work completed on the rig including; coupling of encoder, attachment of a suitable guard, internal attachment of resistance temperature detectors (RTDs) to bearing mounts, making accelerometer mounting points, drilling out rotor bars and introducing a variety of faults to bearings.
- Wiring of the high power AC and DC components and a sensor power and signal loom.
- Writing a risk assessment for the rig and ensuring it was mechanically sound to operate on and that all high power AC and DC wiring was completed in accordance so that the rig could be signed off before testing began.

2.1.2 Software

The software team has aided the design and development of the prototype by writing the application running on the PC connected to the rig via the data acquisition hardware. The software team was also responsible for writing the application that creates a variable speed profile with the programmable DC power supply to get more realistic data from the test rig.

2.1.3 Fault Analysis

The fault analysis team aided the development of the rig in two ways. The first was in helping the hardware team with the designs for the various faults that were to be introduced to the rig's mechanical components. The second was in helping the software team define requirements for their application that would enable integration of the fault analysis team's MATLAB script in the main application.

2.2 Literature Review

User guides from National Instruments describe a number of architectures for various applications scales. The state machine architecture is described as an ideal structure for the implementation of programs with a user interface or with many distinguishable states**Invalid source specified.**. For a user interface the structure can help order multiple processes that are required after a user interaction. This will definitely be a requirement of the prototype system program

For more complex programs a second user guide from National Instruments describes the queued message handler architecture. This architecture provides a structure for separating tasks while still allowing communications between them**Invalid source specified.**. It goes on to say that the communications channel also provide part of the structure needed to make each of the tasks into its own state machine. This architecture should be ideal for the prototype and embedded system application.

2.3 Design & Implementation

The following sections discuss the design and the steps involved in each of the four parts identified in **Figure 23** that make up the prototype system.

2.3.1 Test Rig

Figure 24 shows a block diagram identifying all test rig components and where sensors are attached.

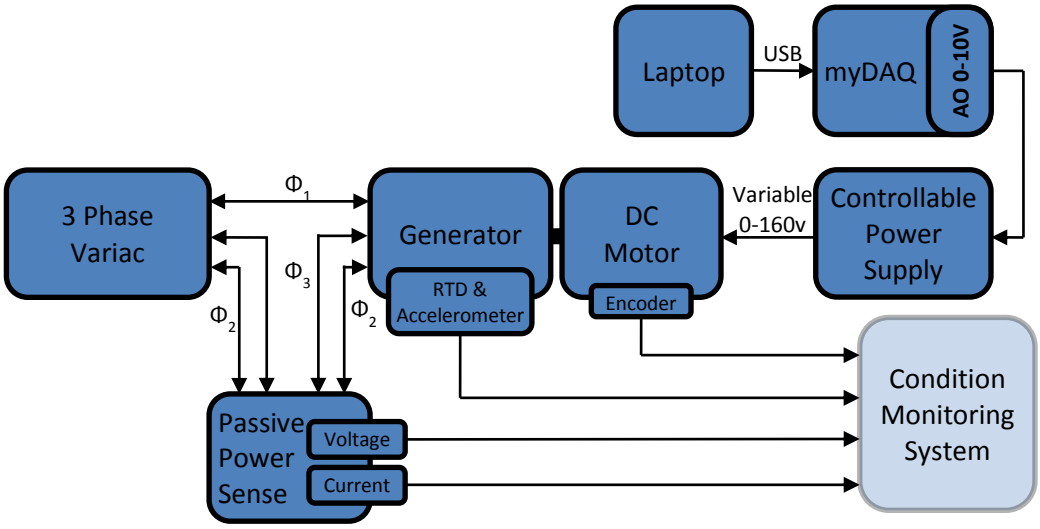


Figure 24: Test rig block diagram (Appendix G2: Interim Report)

Electrical Machines

Following advice from the project supervisors and research into the scope for fault replication, two 3-phase, 4-pole squirrel cage induction machines are used as generators in the rig (one to remain unchanged and 'healthy, the other to have faults introduced to it). An Oldham style coupling allows for small parallel misalignments between the induction machine and the DC motor it is coupled to on a heavy cast mounting plate. Supplied with the mounting plate was a clear Perspex guard for the coupling. However, while appealing for demonstrations, this first guard made the process of swapping bearing timely. A second aluminium guard was designed and made to reduce bearing switch time from around 10 minutes to 2 minutes. This proved useful when running a series of tests.

Sensors

The sensors that are attached directly to the rig include RTDs, accelerometers and an encoder. **Figure 25** shows the mounting of these three sensors to the rig.

(A) (B) (C)

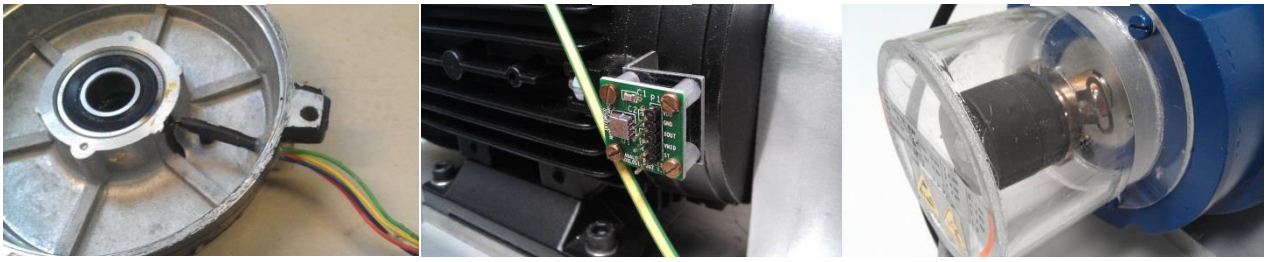


Figure 25: Pictures of sensor mountings (A) RTD; (B) Accelerometer (C) Encoder

Figure 25(A) shows the RTDs glued to the bearing mount inside the motors hub (one at each end) through an access hole drilled on its underside. The first choice for an adhesive was silver-based thermally conductive cement. However, after two failed attempts of trying to get a firm mounting without the glue touching the pins of the RTDs and shorting them, this was discarded. A second option of a wide temperature range epoxy provided a satisfactory mounting. Though not as thermally conductive, it gave a secure connection with a small gap between sensor and the bearing mount.

Figure 25(B) shows the mounts designed and built to hold the chosen accelerometers in the correct orientation in either the X or Y axis of the bearings (four mounts, two at each end). During initial tests it was discovered that when attached to the rig the accelerometer output was dominated by a noisy 50Hz sinusoid. This was diagnosed to be a ground loop issue caused by the brass screws used to attach the accelerometer through its grounded PCB mounting holes. To fix this, nylon screws were used to isolate sensors from rig solving the problem.

Figure 25(C) shows how the encoder is attached behind a Perspex guard at the rear of the DC motor. There was much better access to the drive shaft of the DC motor than with the induction machine where access was required to change rotors and bearing.

All sensors where chosen during the first semester. Justification on the choice of sensors was made through consideration of their resolution, signal to noise ratio, connectivity and other specifications. The research and justification can be found in **Appendix G2: Interim Report.** Data sheets for all sensors can be found in **Appendix B1: Sensor Datasheets**

Wiring

The high power AC and DC sides of the rig are wired through a variac and DC PSU respectively. Shielded cabling was used in accordance with the rig's risk assessment. Wires to power sensors and to carry their output signals to the data acquisition hardware are all routed from their location on the rig to their destination through a wire loom for neatness.

Appendix B2: Rig Wiring Diagram shows an electrical and electronic wiring diagram of the rig. **Appendix B3: Rig Risk Assessment** shows the risk assessment for the rigs operation in Sackville Street Building A2 lab.

DC Motor Speed Control

National Instruments myDAQ has been used to interface and control the speed of the DC motor. The DC motor is connected through a programmable supply then through the myDAQ to a laptop running LabVIEW code.

One of the analogue outputs of the myDAQ is connected to the DC power supplies voltage sense input. The appropriate configuration switches are set on the back of the supply to put it into remote mode.

The commented LabVIEW code in **Appendix B4: Commented block diagram code for the Speed controller** describe the operation of the program. The specifications for the program state that it had to show the speed of the motor in revolutions per minute and that various wind-speed profiles could be set as options to select between. **Figure 26** shows an initial draft of the user interface to this program with a means to select different profiles and view live and historical speeds:

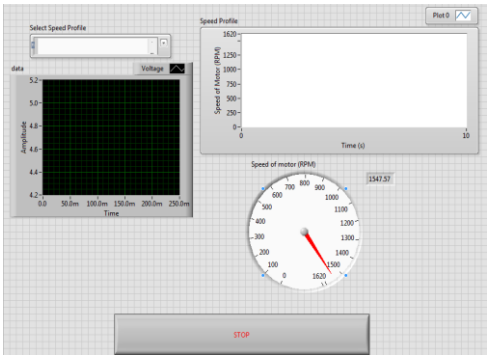


Figure 26: Front Panel of initial speed controller program

Upon revision of this design; the specification for the program was changed. There was no need for the user to be able to see the historical data on a chart. All that needed to be included was the speed gauge with current and voltage limits for the programmable speed controller. The voltage and current limits are a safety precaution set in order to make sure DC motor does not draw more power than its specifications say it can handle. **Figure 27** shows the revised model which was used for testing:

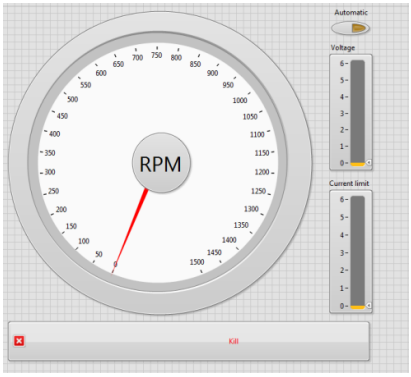


Figure 27: Front panel revised LabVIEW speed controller

During testing it was discovered that with or without load, the DC motor would spin at a near constant speed drawing more current as load was increased. Because of this, no feedback control was needed to spin the rotor at the desired variable speed. This was a welcome discovery as the speed readout of the program which uses the myDAQ's single counters would not have been stable enough for control. During testing the speed read out of the program varied between ±20 rpm of the actual value. **Technical Objectives: section 2.4.4** describes how the same issue was encountered in the main prototype application and the workaround to fix it.

2.3.2 CompactDAQ & Modules

For the prototypes systems data acquisition hardware, a National Instruments CompactDAQ (CDAQ) was selected. The CDAQ platform consists of a metal backplane with USB connectivity, acquisition control, timing and triggering hardware**Invalid source specified.**. Up to eight data acquisition modules can be inserted into the chassis. **Figure 28** below shows the setup for the prototype system.

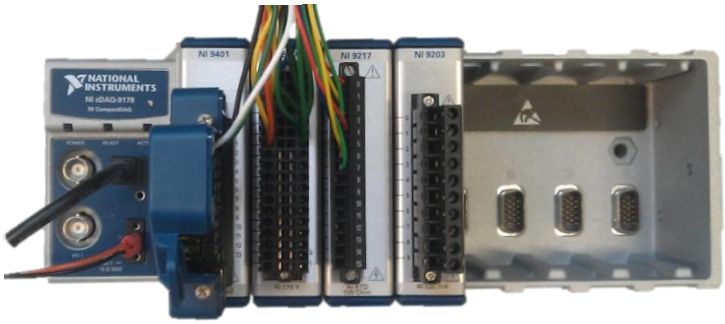


Figure 28: CDAQ chassis with four modules

The four modules in the prototype systems cDAQ chassis and there uses are stated below:

1. **NI 9401 100ns TTL Digital I/O module:** High speed digital input for the one of the encoder's digital outputs. Has access to the CDAQ backplanes on board counters for reliable, hardware timed, speed measurements.
2. **NI 9205 Analogue input module**: ±10V analogue input for accelerometers, current and voltage transducers.
3. **NI 9217 100Ω RTD module:** Provides current excitation and differential measurement (4 wire measurements) for the two RTD.
4. **NI 9203 ±20mA current input module:** Not used in final design but useful when testing output of current and voltage transducers before there interfacing PCBs where completed.

The decision to use LabVIEW was made early on in the project due to its suitability for the type of system proposed, and due to the expertise that already existed in the team. The National Instruments single board RIO platform was selected for the final embedded designs controller as a system that could run LabVIEW programs with all the processing power and connectivity needed. Since LabVIEW was to be used for the final design, it made sense to choose a prototype platform like the CDAQ that was also compatible.

2.3.4 Prototype Software Application

The following sections describe the design process and implementation of the prototype software architecture and a selection of the code modules used within it. Descriptions for many of the smaller modules are not included but are fully documented in **Appendix B5: Prototype and Embedded System UI Manual and Code Documentation**. This 34 page document includes information that will be useful for an operator wanting to use the system, and to a developer wanting to understand the code and make future modifications. This was requested by project supervisors so that future projects and extensions to this project could benefit from the functions already built.

Software Architecture

At the start of the software design process, thought was given to an architecture that the code may make use of. With the proposed functionality it was identified that the solution could be doing up to five different tasks simultaneously. These tasks are:

1. Responding to user input
2. Acquiring data
3. Analysing data
4. Logging data
5. Presenting data

With this break down in mind the first design considered was a state machine. In this architecture a task is split up into smaller sections called states. Each state has a section of code that performs a function, followed by a piece that decides what state to enter next. The five tasks identified above could fit into this model, each having its own state and flowing in a sequential order. Depending on user settings this could change at run time; for example, if log setting is off, then the logging state will be skipped. **Figure 29** shows a simplified state machine where the state code would change depending on the state entered. In this single state machine architecture all states run in the same process and can pass data onto the next state. Each of the tasks listed above could be broken down further in this structure, each having several states of their own; e.g. “initialisation”, “dormant”, “running” and “shut down”.

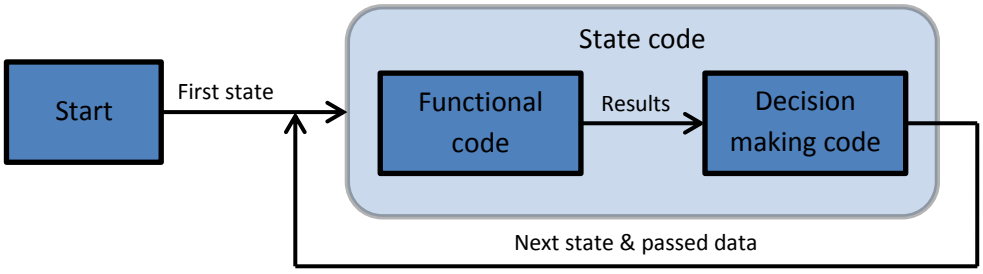


Figure 29: Simplified state machine implementation

The main problem with this design is that although the order the states run in is dynamic at run-time, once one is entered, the next cannot start until it is complete. This is the case even when a state does not require the one before it to run first. If one state has a delay in it to control its operation, that delay will also affect following states. Considering this issue, it was concluded that if tasks that can run disconnected from one another, they should be separated. By putting each task into its own while loop in LabVIEW, it is automatically assigned to its own process thread.

Closer to a good understanding of the system, some standard implementations were looked at and the queued message handler architecture was chosen. This architecture acknowledges that tasks can cause delays if combined and that separating them removes this problem. If separated though, passing instructions between tasks which they must do becomes difficult (e.g. analysis task must be able to tell logging task when to start logging and send it that data). The queued message handler allows communication between tasks by allocating a first in, first out (FIFO) message queue for each task, and giving them all access to the others’ queues. When one of these queues is created in LabVIEW, an area in memory is reserved for the multiple data elements that may be added to it. In the architecture used, the data is a cluster containing two data types. One is a string which must be the name of one of the states in a task’s state machine. The other is a variant which can hold any data type and be used to send specific data required by the receiving task.

At the start of each iteration, a loop (containing the functions of one task) will look at its queue and take the next message available from it. It will enter the state instructed by the message string and make available any data sent with it. Using these queues, each task is able to operate on its own as a state machine by adding messages to it its own queue. Since other loops can also add to it, it can be prompted into another state remotely. The multi element queue means that requests from other loops are never ignored and are processed in order. **Figure 30** shows how this works.

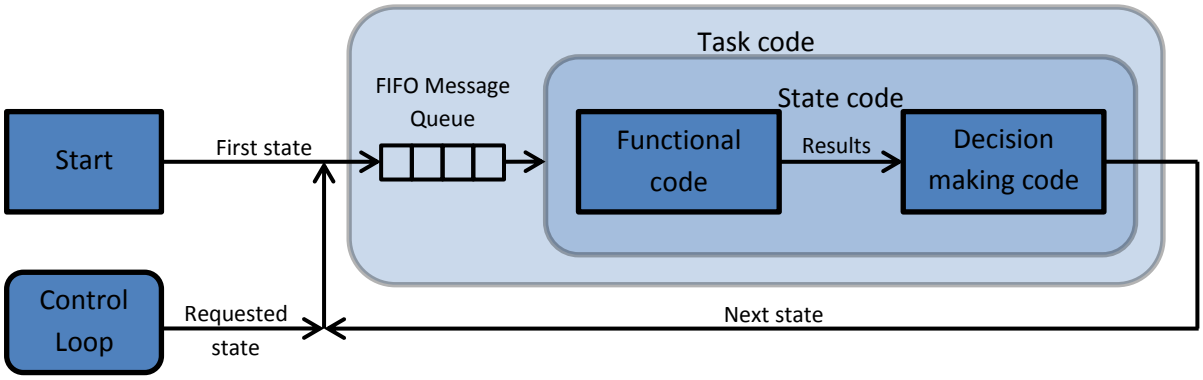


Figure 30: Queue based state machine

With the choice to use a queued message handler architecture finalised, the previously listed tasks were reviewed. It was settled that each of the tasks should run in separate loops but after taking time to understand what each loop would be doing, it was decided that not everyone needed to use the same architecture. The three main functional loops (as they will be referred to from this point) are; acquisition, logging and analysis. When the system starts, these have on-going operations with start-up and shut-down procedures (e.g connect to hardware, open file to log, release hardware, close file). Therefore it made sense that they make use of the state machine architecture each with a message queue to communicate with each other and be controlled. The main control loop also has its own message queue as it could be receiving messages to enter states from any three of the functional loops or the loop responding to user inputs. The last two loops however do not have enough functionality to require a state architecture nor does either of them need to be sent messages to alter their operation. This list below describes each of the loops with an overview of the tasks they fulfil:

1. **Event Handling Loop:** Uses a LabVIEW event structure to respond to any input on the user interface and send a message to the main control loop. Nothing needs to be sent to it so it does not have its own message queue. To interact with the rest of the system it uses the main control loop's queue to pass messages to other loops.
2. **Control Loop:** Controls a start-up routine to initialise loops 3-5 and acts as a middle man between the user interface and them; sending them messages (start, update settings, stop) in response to user input. To avoid complexity that could come from all loops being able to send messages to any other loop, only the control loop has access to all loop queues. If one of the functional loops needs to message another, it goes through the control loop
3. **Acquisition Loop:** Initialises acquisition channels and begins sampling the specified sensors following initialisation message from the control Loop. Responds to messages from the control Loop to change its mode of acquisition, receive settings or stop acquisitions completely and release control of hardware.
4. **Logging Loop:** Initialisation commands from the control loop puts this loop in a dormant state where it holds a configured duration of data in memory ready to log as pre-triggered samples in the event of a fault. It should also respond to commands from the control loop that it should log data without a fault having happened but instead on user request.
5. **Analysis Loop:** After initialisation from the control loop, the analysis loop continuously receives data from the acquisition loop and holds a window of past data which it feeds to integrated fault detection algorithms on each iteration. The control loop sends settings to this loop stating which data should be displayed on the user interface so that it can be sent. The analysis loop was chosen to do this as it has access to all raw and processed data.
6. **Data Display Loop:** Like the first loop this does not need its own message queue. This loop simply updates the user interface indicators with raw and analysed data sent to it from the analysis loop.

One requirement discussed in the above list but not achievable using the queued message handler architecture is the transfer of large amounts of data between the separate loops. For this separate data queues are made to transport large blocks of sampled data between loops.

- **Log Data** – Used by the acquisition loop to stream data to the logging loop. Sends 256 sample chunks of data from any number of configured inputs sensors along with a timestamp of the first sample and a single reading of the rotor speed.
- **Raw Data** – Used by the acquisition loop to send data to the analysis loop. Sends data in the same format as the log data described above.
- **Display Data** – Used by the analysis loop to send any combination of user requested raw and analysed waveform data, measured grid frequency and torque to the display loop.
- **Live Data** – Used by the acquisition loop to send data that is not needed for processing straight to the data display Loop rather than through analysis Loop. This includes three double data types for rotor speed, front hub and rear hub temperature.

Memory is allocated for each of these data queues with a reference that is passed sending and receiving sides. Of the four data spaces needed, there are two different types of operation required; non-lossy and lossy. For log data and raw data, it was important that no data samples are lost as it would cause discontinuity problems in the analysis and post processing of recorded data. Therefore these two data spaces use FIFO queues, which act as a buffer and allow for the connected loops to run unsynchronised. For display data and live data, it is not critical that all data be sent and only the most recent data is required. The data spaces for these use single element queues called notifiers in LabVIEW. If an element is added to a notifier where the previous one had not been read, the old data is replaced.

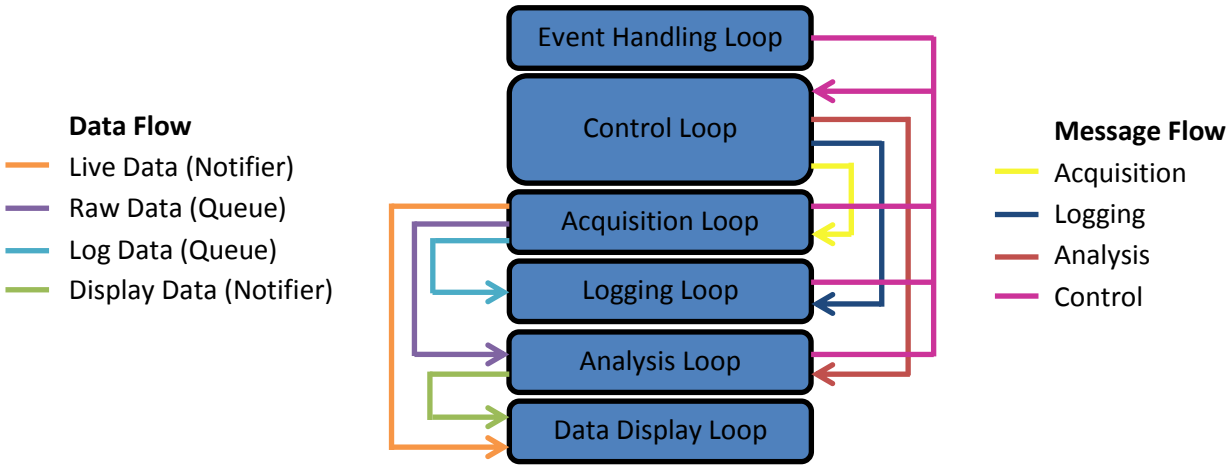


Figure 31: Shows the flow of instruction messages and data within the prototype system software.

Figure 31 shows a diagram that summarises the final structure of the prototype system software. Shared data spaces and message queues enable different tasks to be split into loops that execute in separate process threads and therefore do not delay each other's execution.

Along with the performance benefits that have come from designing a well-structured architecture, another benefit noticed during development has been the ease with which new functionality can be added. During the implementation of the functional code that populates each of the loops above, sub teams have made additional requests for functionality that should be included. Implementing these base and additional functions has been made easier by the built in routes for passing data and instructions. Adding new functions is achieved by adding a

new state in the control loop to handle an interaction between loops and then another in the loop which should respond to a new instruction.

As already described, the loops which perform the most functions or hold the most responsibility are the three functional loops and the control loop. The other two loops perform significantly simpler functions and are not explained further in this chapter. The following sections describe the design and implementation of some of the functional code that fills the many states of the state machines. This is done in full for the control loop but for the other functional loops only descriptions of the functionality they provide is given. Full documentation of all code can be found in **Appendix B5: Prototype and Embedded System UI Manual and Code Documentation**.

Functional Code Implementation

A strategy of designing and adding a few small code modules at a time that could be tested before adding more was adopted while adding functions to the application architecture. Initially, attempts were made to populate a whole loop with the functional code required. This wasn't the best approach as it made debugging problems more difficult. Another reason trying to populate a whole loop did not work was that it was impossible at the start to anticipate all of the modules that were required.

Control Loop

Unlike the functional loops which have on-going processing to do, the control loop spends the majority of its time waiting for messages from the UI or functional loops. Most of its states do not have any decision making code to decide what state to enter next leaving the loop left in a waiting state for the next message. Listed below are the all the states in the control loop along with a description of their function. (Note: A VI in LabVIEW is analogous to a function in other programming languages).

- "Initialise" – Loads the previous system settings from an XML file within the applications root folder and stores them in a local register. It adds two messages to its queue:
    1. "Update Status" – With the text "Initialising"
    2. "Launch Settings Dialog"
- "Launch Settings Dialog" – Calls "Settings Dialogue [UI].vi", the only other VI with a user interface. It passes the settings it recalled from the XML file in the previous state and then receives a new or unchanged version from the VI. If changes have been made it updates the local register and overwrites the old XML file so that settings do not have to be re-entered next time the program is run. A separate VI "Update UI Channel Select.vi" also runs to create a list of available sensors that is to be shown in two drop down boxes in the UI. A message is now sent to the Analysis loop, "Change Displayed Data", with details of the list as it is the Analysis Loops responsibility to send requested data to the memory locations used by the Data Display Loop. If changes were made, a new message, "Broadcast Settings", is added to the Control Loop message queue.
- "Broadcast Settings" – Sends an "Update Settings" messages to each of the functional loops the along with the new settings taken from the local register.
- "Change Displayed Data" – In response to new sensors being added, analysis being turned on or the user changing the desired display channel, this state is entered. It adds a "Change Displayed Data" message onto the analysis loops queue along with the new requested data channels.
- "Update Status" – This state receive messages with message data that give instructions to add or remove lines from the status display on the user interface. The state code runs the VI "Message to Status.vi" which interprets the message and outputs the new status.

- "Start Analysis" – This state is entered in response to the user turning the Analysis button on. First it runs “Update UI Channel Select.vi” and routes its output to the message data of a “Change Displayed Data” message it send to the Analysis message queue. It then adds a “Start” message to the Analysis queue.
- "Stop Analysis" – Exactly the same procedure as the “Start Analysis” state and in the same order accept for the final step where it adds a “Stop message instead of “Start”.
- "Start Logging" – Takes the message sent with the “Start Logging” instruction from the UI Loop and sends it as new data in a message to the Logging Loop with the instruction “Start”.
- "Stop Logging" – Sends a “Stop” Message to the Logging Loop with no data.
- "Acquisition Mode" – This state runs in response to the UI Loop’s messages in response to the user pressing either the Record or Live button. The message data passed to the “Acquisition Mode” state is forwarded to the Acquisition loop with message instruction "Acquire”.
- "Exit" – This state handles the smooth closure of the prototype system application and can be triggered by the UI loop in response to the user pressing the stop button or by any of the functional loop if they should encounter an error. The states code releases control of all memory created for the shared data spaces and sends “Exit” messages to each of the functional loops so they may close down with their own specific procedures. If an error had occurred in any of the functional loops, it is written to the user interface in the status panel.

Acquisition loop

The acquisition loop is responsible for the following functions:

- Loading all sensor data into shared data spaces in 256 sample chunks in the required format for logging and analysis functions.
- Receive settings defining sensor information and acquisition parameters and configure attached hardware appropriately. If this is received during acquisition, all hardware references are released and reconfigured to begin again
- In live data mode it continuously acquires data and adds it to the shared data spaces.
- In playback mode, acquisition is paused and recorded data is instead sent to the shared data spaces. This enables recorded healthy data to be played back and then switched seamlessly to live data from a faulty system. The results of this feature are reported in section **Technical Objectives: 2.4.3**.
- Record all inputs on request so they may later be used in playback.
- Releasing control of all configured hardware on exit.
- Report status of acquisition to control loop to update the user.

The acquisition loop operates as a state machine just as the logging and analysis loops do. The strategy for implementing the required functionality in the acquisitions loop was split into 4 steps:

1. Design a sequential program (but not dynamic like a state machine) separate to the main application that reads in settings, configures the data acquisition hardware and displays reading. It waits for instruction to stop which it does by closing down the configured hardware.
2. Add steps to that program to pause the acquisition of data update settings on the configured data acquisition tasks and re-start.
3. With these first two steps combined into one, divide each of the functions into their own simple state machine architecture and add decision making code so that the process can run on its own and be fed updated settings.
4. Add this into the prototype system architecture so that it may be controlled externally.

After these base steps were brought into the main architecture, adjustments were made so that data was added into data queues rather than being immediately displayed.

Logging Loop

The Logging Loop is responsible for the following functions:

- Hold in memory a window of past data of a length in seconds specified by the user. When settings are sent to this loop it uses the requested value of pre-trigger data log duration to create a buffer which will hold the window of data.
- In the event that a fault is detected (the trigger), this loop logs the pre-trigger window of past data held in memory and continues to log live data for a time specified in the settings.
- On request from the user, manually log data from all connected sensors for a specified period of time or until sent instruction to stop.
- Produce logs with all raw sensor data and with all information required for post processing.

Analysis Loop

The Analysis Loop is responsible for the following functions:

- Sending any requested raw or analysed data to the shared data space used by the data display loop.
- Receive incoming data from the acquisition loop and provide a moving window of data for analysis routines.
- Pass the moving window data and all required settings to integrated fault detection algorithms.
- Pass messages back to the control loop when faults are detected by the fault detection algorithms.
- Route the user specified sensors to auxiliary analysis routines to detect the grid frequency and make estimation for torque.

The main state in the analysis loop “Analysis”, operates by performing the following five actions in sequential order:

- Provides settings and access to the raw data queue for “Moving Window.vi”.
- Sends the current window of data and settings to “Auxiliary Analysis.vi”.
- Passes the grid frequency output of the “Auxiliary Analysis.vi” and the current moving window to “Analysis.vi”.
- If a fault was detected, send a “Start Logging” message to the control loop with the fault description.
- Pass the “Analysis.vi” FFTs output and the current moving window to the “Send to Display [Analysis].vi”

Grid Frequency

The grid frequency is needed by the fault detection team to complete their algorithm of electrical fault detection. The method obtained for the grid frequency measurement is based on the research which has been discussed in detail in the interim report (**Appendix G2: Interim Report**). The measurement for the grid frequency is done by collecting samples of the current signal which originated from the grid and then placing these collected samples into an array. The array is then passed into the LabVIEW code where the frequency of the current signal is determined using the zero crossing detection technique. **Figure 32** below shows the process happening in the program using the prototype system as the mean to collect data.

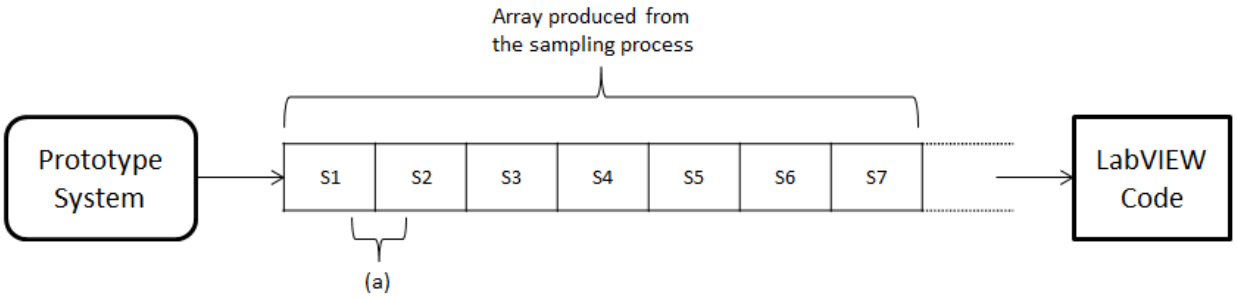


Figure 32: The transfer of array from the prototype system to sbRIO with (a) showing the time between each sample obtained from sampling frequency

Zero crossing detection technique uses a method where it detects the time when a signal reaches a certain reference point. This reference point is usually set at zero. When a signal reaches two reference points, it means that the program has detected half of the time period of the signal's cycle. The program is then expanded to include the third reference point. After the third reference point, the program returns the value of the whole time period of the signal in one cycle. By obtaining the time period of the signal, the frequency of the signal can be obtained using the equation:

The program for the grid frequency measurement is written in LabVIEW. It requires the value of the sampling frequency used to collect the samples of the current signal from the grid. The program works by counting the number of samples in one period using the zero crossing detection technique and then multiplying the obtained number of samples with the period of the sampling frequency. This multiplication process gives the measured time for one period. The equation used is as followed.

$$t_m = \frac{N}{f_s} \qquad \textbf{Eq 15}$$

t_m=measured time in one period; f_s=sampling frequency; N=number of samples

However, because the wind turbine condition monitoring box will be placed inside a noisy environment, there is a high chance that a high frequency noise will be introduced to the current signal measured from the grid. The noise affecting this current signal will also affects the accuracy of the measured grid frequency. To overcome this problem, the program is made so that it is able to increase the length of the measured time, t_m by imeasuring over more periods. The obtained measurement time is then divided with the amount of the specified period to obtain an average value of the signal's period. By doing this, the noise which caused the inaccuracy in the measured time of a period can be cancelled out. The averaged signal period is then inverted to gain a more accurate result of the grid frequency. The equation used for the whole process is as followed.

$$f_{av} = \frac{nf_s}{N} \qquad \textbf{Eq 16}$$

f_{av}=average frequency; n=number of period; N=Number of samples; f_s=sampling frequency

The front panel of the completed program is as shown in the **Figure 33** below. The sampling data is the input array from the prototype system and the frequency panel shows the calculated frequency based on the input array and the given sampling frequency. The input for the sampling frequency is added in the sampling frequency panel. The number of period can be specified in the no. period panel as shown in **Figure 33** below.

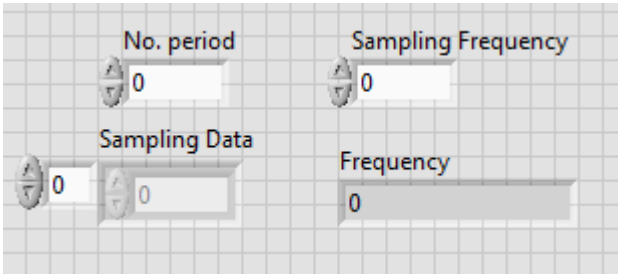


Figure 33: Front panel of the frequency measurement program

Torque

Another addition to the fault detection technique is measuring the torque produced by the generator. This auxiliary measurement is needed in case the fault detection algorithm fails to detect a disturbance in a component inside the turbine. Although the torque measurements do not contain the information required to specify the location or type of fault, they can be an indicator that one has occurred.

An extensive research of the most suitable technique for the measurement of torque has been done. Based on the research, discussed in more detail in interim report (**Appendix G2: Interim Report**), the technique chosen to measure the torque uses the output voltage and the output current produced by the generator. The equation used to calculate torque based on these measured outputs given below:

$$T_e = \frac{\sqrt{3}P}{6}\left[(i_a - i_b)\int V_{ca}dt - (i_c - i_a)\int V_{ab}dt\right] \qquad \textbf{Eq 17}$$

T_e=Torque; P=Number of Poles; i_a,i_b,i_c=Phase currents; V_{ab}=Line to Line Voltage

The inputs signals that need to be measured from the generator are the current and voltage of two phases. The third phase current mentioned in the equation can be calculated using the measured two phase currents. The equation used for this is given below.

$$i_c = -i_a - i_b \qquad \textbf{Eq 18}$$

In addition, because the measured voltages are line voltages, further calculation is needed to convert these line voltages into line to line voltages to correspond with **Eq 17**. The equations used in this calculation are,

$$V_{ca} = \sqrt{3}V_c \qquad \textbf{Eq 19}$$

$$V_{ab} = \sqrt{3}V_a \qquad \textbf{Eq 20}$$

To solve **Eq 20**, the time between the first samples of the output measurement with the next one is requires. To get the value of time, the program requires the sampling frequency used to sample the output signals of the

generator. The period of the sampling frequency is multiplied each time a new sample enters the equation so that the program obtains the present time value to be used for this new sample.

A more accurate equation to calculate the torque is given below:

$$\boldsymbol{T_e = \frac{\sqrt{3}P}{6}\{(i_a - i_b)\int[(V_{ca} - r_s(i_c - i_a)]dt - (i_c - i_a)\int[V_{ab} - r_s(i_a - i_b)]dt\}} \quad \textbf{Eq 21}$$

r_s=Stator resistance

However, this equation requires the value of stator resistance. If the stator resistance is unknown, **Eq 17** is used instead. The program is written in a way that it allows the user to switch between the two equations (**Eq 17 and Eq 21**) based on the user preferred options.

The two equations for torque are written using MathScript node in LabVIEW. MathScript node allows the use of syntax to write the equations so it becomes easier to debug, in case a fault occurs during testing. The block diagram for this program is given in **Appendix B6: Block diagram for torque measurement**.

All code developed for the prototype system can be found in **Appendix S3: Prototype Source Files**.

2.4 Testing

The following sections describe some of the core tests and notable function tests carried out during and towards the end of development of the prototype application code.

2.4.1 Software Architecture

To test the architecture of the prototype software before functional code was designed, a number of states were added to each loop. At this point the states were unpopulated and just a guess at what the final ones would be. To run each test, the user interface was created with buttons to trigger each of systems functions. Each of the events that respond to these in the UI loop was coded so that the relevant message gets sent to the control loop when its button was pressed. To make sure everything functioned as expected each button was pressed in turn with the LabVIEW option to step through code turned on. Using this, the flow of the message could be followed, for example:

1. User presses Analysis button.
2. UI loop message: “Start Analysis”, to control loop.
3. Control loop message:
    a. “Change Displayed Data”, to control loop.
    b. “Start”, to analysis loop.
4. Control loop enters “Change Displayed Data” state where is will update the content of channel select list boxes.
5. Analysis loop enters “Start” state where it clears any remaining “Dormant” messages and then sends message “Analysis” to its own loop

This process was carried out for every type of user interface interaction and also for the messages that originate from each of the functional loops. An example of one of these would be a message from the analysis to initiate logging in response to a fault. For this, a message is passed through the control loop to the logging loop. Testing carried out at this stage ensured that while implementing the functional code, no errors would arise from problems with the architecture.

2.4.2 Test Features of the Control Loop and Functional Testing

As functional code was added to the prototypes state machines, a feature was built in to the control loop to aid troubleshooting when code inevitably did not perform as expected. Code was written in each of the functional loops so that if an error occurred, they would close and send a message with the error description to the control loop. The status indicator on the front panel was then used by the control loop to display the description. As long as functional code was added in sections such that no more than one error was likely to arise, errors could be fixed individually. If more than one error was reported to the Control loop, only the most recent would be displayed.

Most errors during functional testing were found while developing the acquisition and logging loop. Because these loops regularly open and closes references to memory and hardware, if the order in which this is done is not correct, errors can occur. The most common issue occurred when the program shutdown unexpectedly following an error and did not clear open references to resources. This then caused an immediate crash on the following start up due to conflicts over resources. To safe guard against this, code was added to the start of each location where these problems could occur to clean any stale references.

2.4.3 Testing of Record and Playback Functions

The functionality to record all data coming into the system so it can be played back later, was added to aid development of fault detection algorithms. The first version of the MATLAB software written to detect faults worked by making comparisons between energy levels over a small duration of time. Because of this it would not detect a fault if started in an already faulty system. The idea behind adding this additional functionality was that it could be set to playback healthy data in a machine with a fault then while the machine is running, switch from recorded healthy to live faulty data.

When this function was tested it did work as expected but with one unforeseen problem. Because of inevitable difference in the phases of line currents when the switch occurs, there are discontinuities in the merged signals. When these are put through an FFT the effect is a large increase in the energy for all frequencies. Because of this the record and playback feature was decided as unsuitable for its original purpose to aid the development of the fault analysis and detection algorithms. **Technical Objectives: section 3.3.4** describes how the playback feature was still used while testing the final embedded systems.

2.4.4 Rotor Speed Measurements

To measure the rotor speed of the rig, one of the encoder's 1024 pulse per revolution outputs is connected to a digital input on the CDAQs 9401 module. This is configured to increment the value of a counter on each pulse without interrupting the connected PC. The first version of code written to calculate the rotor speed used sent requests at regular intervals that would prompt the counter to send its current value. This used the PC's millisecond clock to synchronise when it sent requests so that it could calculate speed from the difference in pulses over a known time. In theory this should have produced a resolution of around 0.5RPM at 1500RPM but what was observed over a number of readings for a fixed speed is shown in **Figure 34**.

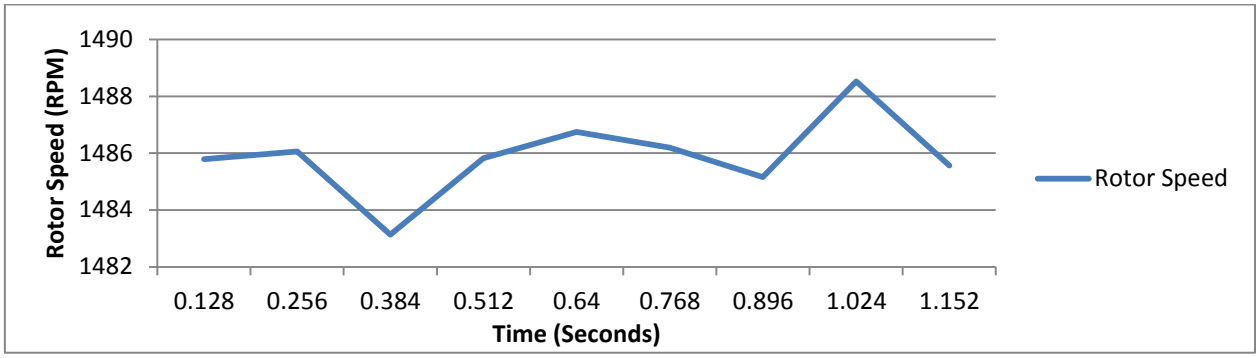


Figure 34: Plot showing software timed rotor speed measurements while motoring at constant voltage

This shows more variation than there should be for a constant voltage. While troubleshooting, the magnitude of the variation was linked to the amount of load on the PC running the test code. While running processor intensive applications alongside the prototype code, the variation saw a steep increase. The conclusion drawn from this was that despite the reliability of the PC's millisecond clock being high there must be other unpredictable timing problems that resulted in a varying non-periodicity between the actual time the counter would return its current value. These could include variations in the execution speed of the program and perhaps any jitter in the USB driver used to communicate with the CDAQ. To solve unreliability caused by software timing issues, the second version of the code used hardware timing. In this code, when the hardware counter is set up as an input to count pulses, another is set up to output a fixed frequency output. The frequency configured is dependent on the sampling frequency used but for an example, with 2kS/s it is 16. This is used as a sample clock for the first counter which now instead of it waiting for a request from software, outputs its count at on the rising edge of the second counter. Results from this second version of code produced an output with a significantly smoother measured output; variation = resolution = 0.47RPM at around 1500RPM

2.4.5 Testing of System Performance

When the prototype system was complete and ready for testing, the MATLAB code was still under development. Still wanting to test the systems performance with similar loads, FFTs where implemented with LabVIEW in the place of the MATLAB script under development. **Table 3** (below) shows tests carried out with the prototype code running on a relatively low spec dual core 2.1GHz 64-bit machine.

Sensors	Analysis	Logging	Processor Load	Observations
18 @ 2kHz	Off	Off	19%	Number of elements in raw queue remains at 0 at all times indicating no delay in processing is caused by multiple elements of data waiting to be processed.
	Off	Continuous	21%	
	256Wl, 90% OVL	Continuous	34%	
	2048Wl, 99% OVL	Off	45%	
		Continuous	65%	
18 @ 4kHz	2048Wl, 99% OVL	Off	83%	Number of elements in the raw queue hovers around 0 and 1.
		Continuous	100%	Number elements in the raw queue builds at a rate of 374 per minute.

Table 3: Processor load testing with 18 simulated sensors (WL = window length, OVL = overlap)

These tests gave some measure of the code efficiency before moving to the embedded platform. Observations from the tests show that; when using the required sampling frequency of 2kHz, with the most demanding analysis settings and with logging turned on, the system performed well with 18 sensors. The processor load was low enough that the queue that feeds the analysis loop with data did not fill past 1 element. As soon as an element was added, the same element was taken for processing without any delays from queuing. Tests run for the most severe case; 18 sensors at 4KHz, most demanding analysis settings and logging did show to be too much for the system. After a minute of running under these settings, the raw queue feeding the analysis loop contained 374 elements equivalent to 23.9 seconds of data. This would obviously be unacceptable for a condition monitoring system but the settings that caused this would never actually be required.

The relevance of these tests when considering the move to the Single Board RIO is important. Though they have been carried out on a system with a much higher spec, they do show that processor load scales with the demands of the application. When testing performance on the RIO, settings can be tuned for a balance between signal processing latency and fault detection accuracy. More detailed tests were planned at this stage for the RIO including some for latency. These are detailed in **Technical Objectives: section 3.3.4.**

2.4.6 Using the System to Extract Data

The prototype system has not yet been used to test the MATLAB algorithms under development by integrating them in the code. However data taken using logging function has been used. Multiple tests have been run under a variety of conditions:

- Above and below the synchronous speed of the motor. Speeds between 1460 and 1540 RPM.
- In a healthy state or with electrical or bearing faults.

Appendix B5: Prototype and Embedded System UI Manual and Code Documentation shows the excel formatted output of the log files produced by the prototype system code. **Appendix S2: Prototype Test Rig Data** contains all data gathered during development and testing.

2.4.7 Grid Frequency Testing

The grid frequency program was first tested by simulating a sinusoidal signal using a function provided in LabVIEW. The test was carried out with an array of simulated data so the frequency was known. The number of

samples in this array was set at 1000 with a sampling frequency set at 2 kHz, matching the specification of the data acquisition hardware. A number of simulated signals with frequencies around 50 were put through the program which detected each correctly.

A further test was done when the prototype system is ready to measure the signal from the generator. The prototype provided an excel file of the measured current signal from the grid. A new test VI was needed to convert this excel file into a suitable format where the samples can be read by the grid frequency program. This test VI is shown in block diagram in the **Appendix B7: Test VI for the grid frequency measurement**.

The sampling frequency used by the prototype system to sample the data was 2 kHz. The number of period specified for this test was set at 100. The frequency measured by the grid frequency program based on the given input was 50.0525 Hz. This was validated by measuring the period of the signal with a scope and current probe.

2.5 Summary

Developing the prototype system has provided a useful tool to aid development. It has provided data for various faults that can be introduced to the rig and by using off-the-shelf data acquisition hardware this was done quickly before the custom DAQ board was ready to acquire data. Data has been used evaluate the rig, fault detection techniques and sensors. As well as aiding the development of other sub teams, equal worth that has come from its development is in the prototype software that has much of the functionality required in the final embedded design.

3. Final Embedded Design

This section presents the final embedded systems design, implementation and testing

The development of this system has involved all team members in multiple tasks. The project objectives addressed by this design work presented in this section are:

- Develop condition monitoring system layout – power supply, interface input A/D circuits, microprocessor, GUI, A/D outputs etc.
- Design and build power supply.
- Design and build A/D interface circuits.

These tasks can be seen in **Appendix G2: Original Project Description.**

Following the prototype used for development, the final embedded system is the final implementation of this project. It provides a portable platform for installation in wind turbines that can run the fault analysis and detection algorithms discussed in **section 1**. This platform is a standalone system that has power circuits, signal conditioning, data acquisition and data processing electronics. It is required that it is flexible and customizable all within a tight budget.

3.1 Relationship with the Rest of the Project

The embedded design completes points 6, 7 & 8 partially covered by the prototype system and also covers points 3, 4 & 5 of the revised project specifications in **Introduction: section 1.2**.

The embedded system was designed to have two separate processing units. The two main components in the embedded system are the DSP, controlling the custom DAQ board and the single board RIO (sbRIO) which functions as the systems main controller. Separate teams have been involved in the development of software for these two units. The code for the sbRIO was inherited mostly from the prototype development and handles all data processing logging and analysis using built in MATLAB scripts. The DSP code was written by the hardware team with an understanding of the control needed by the DSP over components like the multiplexers and ADC.

The communications link between the DSP and sbRIO was an essential part of the project to get right as it is the point of entry of all data for the sbRIO. Both teams in charge of selecting these processing units worked together in early stages to make sure the specifications of these units would allow them to communicate with each other and with enough bandwidth for the desired amount of data. This speed is provided by a serial communication protocol called SPI.

The custom DAQ board is controlled by the DSP but before signals get to that, they go through several other circuits for protection conditioning and digitising. It was the work of the hardware team to put these together.

3.2 Literature Review of the Technical Topic

In a Condition Monitoring system designed by the University of Strathclyde [24] LabVIEW was used with and two National Instruments DAQ cards. One was a USB connected DAQ card with maximum transfer speed 480Mbs [25]. The other card was a PCI card with much higher transfer rate. The USB 2 data rate will be more than sufficient as was shown in the prototype system.

Research into DAQ systems showed that many data acquisition systems have a standard hardware layout:

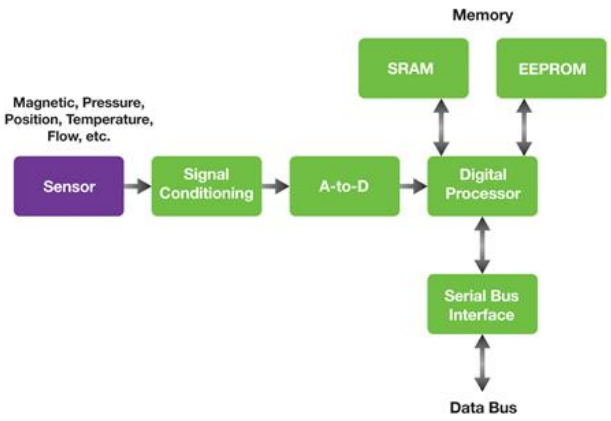


Figure 35: Common DAQ system layout [26]

Like standard layout in **Figure 35,** the digital receiver DAQ by Texas Instruments also uses a DSP as a final stage similar to [27]. The ICSSA 2009 International conference on signals, systems and automation states the same layout although they use a PIC microprocessor [28] in the final stage. From review it is decided that a DSP and a microprocessor will be reviewed. A review of these components can be seen in **section 3.3.3**.

The design of the protection circuit is based on the book written by Ronald B. Standler [29]. From this book, three design choices have been considered and the one chosen suitable for the developed system is the combination of Zener diodes and rectifier diodes. A detailed discussion on the circuit can be seen in **Appendix G2: Interim Report**.

For information on FIR theory a book by Edward R. Cunningham was looked at [30]. Practical knowledge of FIR code was researched from the internet. The website DSPGuru [31] had practical comparisons of different types of filter.

3.3 Analysis, Design, Implementation Testing, Experiments

3.3.1 Communications

Two communication protocol where identified as possibilities for the required communications link between the ADC, DSP and sbRIO. There are numerous options but the most common are:

- I^2C – Inter-Integrated Circuit [32].
- SPI - Serial Peripheral Interface [32].

I^2C is a two wire interface with wires for clock and data. It has three speed configurations; slow (up to 100Kbps), fast (400Kbps) and high speed (3.4Mbps). Typically, most low cost devices are compatible with slow and fast speeds. More expensive devices are capable of high speed [32].

SPI communications protocol with four wires. These are MISO (Master Input Slave Output), MOSI (Master Output Slave Input), clock and CS (chip select). Each device on the SPI bus is connected to the master device by its own CS line allowing the controller to communicate with individual components on an SPI bus. The speed of the SPI protocol isn't defined in the standard and is only limited by the speed of the lowest clock frequency device on the bus [32].

The main advantages and disadvantages between the two protocols are shown in **Table 4** below:

I^2C		**SPI**	
Advantage	**Disadvantage**	**Advantage**	**Disadvantage**
Easy to implement	Possibly limited to 400Kbps	Faster Speeds	Many wires required 4+
Only two wires required	Slave address take up data.	Widely implemented.	Only one master.
Widely implemented.	Master and slave share the same data line and further limits transfer speeds.	Simultaneous send and receive is possible.	Limited on the number of slave devices.
Multiple masters.	Usually limited to onboard communications.		No defined speed categories. This adds ambiguity during device selection.

Table 4: I^2C and SPI advantages

As ADCs are typically compatible with SPI and not I^2C, an SPI communications bus between the ADC and processor is required to increase the possible choices for this device. The choice between I^2C and SPI is unclear for the processor to sbRIO communication because initially both seemed to provide enough bandwidth. Two slaves on the same SPI bus increases program complexity requiring more time.

A low cost microcontroller and a DSP evaluation kit was purchased, as early stage design required a platform for testing code. This enabled code development to run alongside hardware development. It was discovered during testing that the 400Kbps I^2C protocol wasn't going to be possible on the DSP selected. A data rate of at least 600Kbps was calculated as the minimum required for the 18 sensors and 2kS/s rate defined in the specification for the embedded system. Following this discovery the decision was made to use SPI for both the communications between the ADC, DSP and sbRIO, with the DSP set to master and the ADC and sbRIO to slaves. This allowed a communication speed of up to 6Mbps. This decision increased the available bandwidth for a possible further expansion of sensors or additional devices in the future. Manufacturer's example code for SPI on the DSP and sbRIO was another benefit of this design decision.

3.3.3 Custom DAQ Hardware

The custom DAQ hardware designed during this project is required for the interface between sensors and the sbRIO. The decision to develop a custom DAQ as opposed to off the shelf gave improved flexibility with no limitations. Each component was carefully selected by the team based on the requirements found from research and the original project outline. The Custom DAQ block Diagram can be seen in **Figure 36** below:

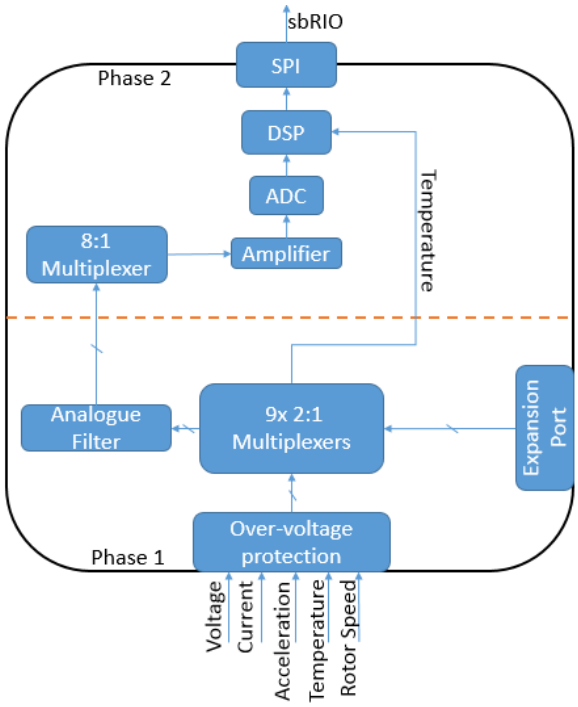


Figure 36: Custom DAQ hardware diagram

The system consists of over-voltage protection to protect the PCB from human error such as connecting the wrong sensor into the wrong port. From here the signals enter a multiplexer that combines the expansion port signals with the onboard signals. The output of this is passed through an analogue filter with the exception of temperature, which is connected directly to the DSP. The analogue filter was selected to limit the signal's frequency to help EMI rejection and aliasing. This signal then enters an 8:1 multiplexer that allows the selection of the sensors required for sampling by the ADC. In-between the ADC and multiplexer is an amplifier. This scales the signal into a range that makes full use of the ADC's resolution. The analogue signal is then converted into a digital signal that is then received the DSP by the SPI link. Firmware is implemented on the DSP to apply a digital filter to these signals. The data is then transmitted to the sbRIO via SPI.

The following sections describe in more detail the design and implementation of the hardware outlined above. Included in this is a breakdown of the code on the DSP explaining its functions and how they were implemented.

Protection Circuit

A protection circuit is needed to protect the custom DAQ hardware from an overvoltage and overcurrent condition. These two conditions can be caused by a human error such as accidentally plugging the DAQ into a high voltage supply line exceeding the voltage tolerance of the DAQ.

Following research in this area, the team decided to use a design involving a combination of two Zener diodes and two rectifier diodes to mitigate the effect of possible human error. **Figure 37** shows the chosen circuit design.

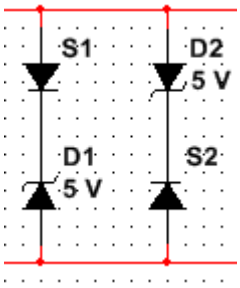


Figure 37: Protection circuit using zener diodes

When a surge voltage higher than the maximum operating point occurs, one of the two zener diodes conducts depending on the polarity of the surge voltage. This conduction creates a short circuit which in turn diverts the current away from the main component in the DAQ board. By doing so, major damage towards the DAQ can be avoided. The components used in this design can protect the DAQ from an input voltage higher than ±5V. It also needs an additional component to protect the DAQ board from a high surge current. For protection against overcurrent condition, a fuse was added in series with the overvoltage protection circuit.

A simulation of the protection circuit using Multisim was carried out to evaluate the performance of the chosen design. The result of the simulation is shown in **Appendix C1: Simulated result of the protection circuit**. A completed schematic diagram of the circuit can be found in **Appendix C2: Schematic diagram of the protection circuit**.

Analogue Low Pass Filter

Measurement of the signals entering the DAQ board from the sensors can be distorted by high frequency noise produced from the interference of different components of the board itself. This can cause inaccurate readings when the signal is being processed. A filter is needed to provide a solution for this problem. Although a digital filter implementation already exists on the DSP the analogue filter was still included as a back-up.

The design of the low pass filter consists of only two components; a capacitor and a resistor. The reason for this design choice is because it is simple to be built as it required only two component which can be easily replace if a malfunction occurred. The values chosen for the capacitor and the resistor are calculated from the equation below.

$$f_c = \frac{1}{2\pi RC} \qquad \textbf{Eq 22}$$

f_c=Cut off Frequency; R=Resistance; C=Capacitance

From this equation, the value of the resistor and the capacitor determine the value of the cut off frequency of the signal allowed to enter the DAQ board. As defined by the fault detection team there are no frequencies of interest above 1kHz. To allow these to be captured sampling at 2kHz minimum would be needed so a would be that. The chosen value for the capacitor is 1nF and for the resistor is 82kOhm.

A simulation of this low pass filter has been performed by using Multisim to see the value of the filter cutoff frequency and to confirm that the filter is operational. The result of the simulation is shown in **Appendix C3: Simulated result of the low pass filter**.

Expansion PCB

One of the key features required for the presented condition monitoring system is flexibility. To strive towards this goal the ability to switch between 9 additional expansion port sensors was added.

The design allows additional expansion PCBs to be connected onto the main DAQ PCB. These additional boards could have extra electronics to make the system compatible with future sensors such as piezoelectric accelerometers. As future research identifies further techniques to detect faults the sensors required for these techniques can easily be connected without a redesign of the main DAQ board. The DAQ PCB and placement of the Expansion sensor PCB can be seen in **Figure 38**:

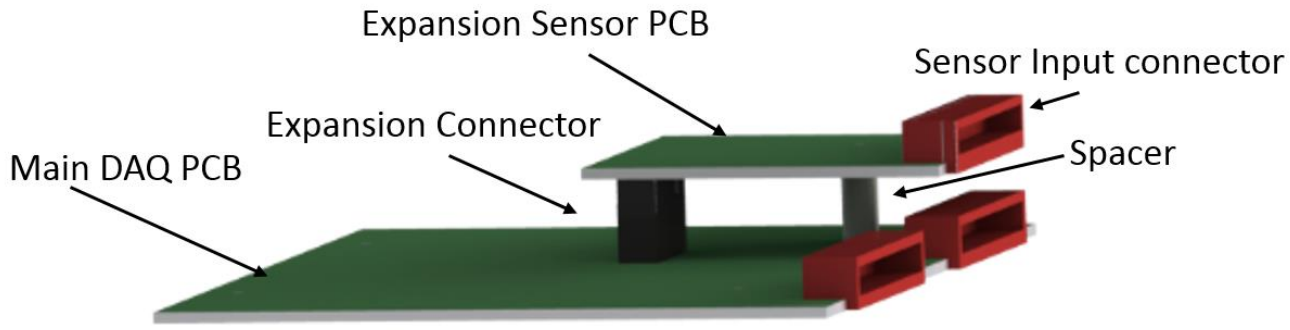


Figure 38: PCB Expansion board SolidWorks drawing

Nine sensor inputs are multiplexed by 9 2:1 multiplexers while the other 9 inputs come from an expansion connector. Therefore additional sensor interface circuits can be placed on a separate PCB and connected easily. This speeds up design for a new sensor and allows a suitable amount of flexibility. The DAQ's processor then uses control techniques to switch the required multiplexers on or off. The settings for the activation of the sensors are expected in the form of a string of 18 boolean values from the sbRIO where each value corresponds to an input and either activates or deactivates. It is the shared knowledge of these settings that will allow the sbRIO to decode the data it receives into sensor readings

The multiplexer selected to allow switching on one ADC is the Analogue Devices ADG1233YRUZ. This was selected as it contained 3 2:1 analogue SPDT (single poll double throw) multiplexers which reduce the amount of ICs needed. It was also selected because it is very competitive on price, at £4.43 per unit and could cope with the maximum input voltage swing of ± 5v. Further information can be found about this device in **Appendix C4: ADG1233YRUZ Datasheet.**

Signal conditioning

The purpose of the op amp circuitry is to scale signals so they can be processed by the ADC. The signals need to be converted from their input value of ±5V (Max) into the range of 0-3V. A non-inverting amplifier calculation gave negative resistance results, since the signal is attenuated. Therefore an inverting amplifier was considered, followed by an inverter. First a value of 10KΩ was chosen based availability of the input resistance. The equation that was used to calculate the feedback resistance for a gain of 0.3 is shown below:

$$V_{out} = -\frac{R_f}{R_i} V_{in}$$ **[33]** **Eq 23**

Vout=output voltage; Vin=input voltage; R_f=feedback resistance; R_i=input resistance

The inverter was designed using **Eq 23** with a gain of 1. Both the resistors were chosen to be 10KΩ based availability, particularly as the same value is used in the inverting amplifier.

Finally to convert the signal to positive, 5V DC had to be added to the signal using a summing amplifier. The equation for a summing amplifier for two signals is shown below:

$$V_{out} = -R_f\left(\frac{V_1}{R_{i1}} + \frac{V_2}{R_{i2}}\right) \quad \textbf{[33]} \qquad \textbf{Eq 24}$$

Vout=output voltage; V_1,V_2=input voltages; R_f=feedback resistance; R_{i1},R_{i2}=input resistances

The summation of signals is accomplished by setting the two input resistances the same. Using the same values chosen before, both input resistances are set to 10KΩ and the feedback resistor is set to 3KΩ.

The circuit diagram made in Multisim, shown in **Figure 39** was used for simulation and confirmed the desired scaling was accomplished by the design.

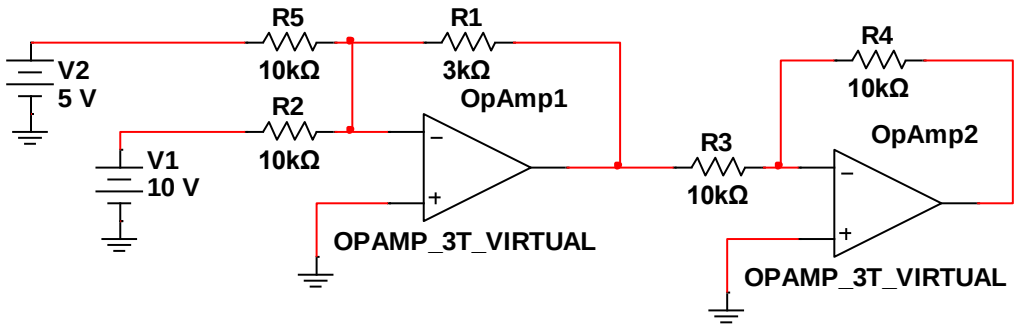


Figure 39: Amplifier circuit

Sensor Multiplexer

As there are multiple sensor inputs it is a common solution to multiplex them before they enter the ADC. This is to reduce the amount of ADCs required because typically they are more expensive than multiplexers. This also helps reduce the overall circuit complexity that usually leads to errors such as corrupted data. The multiplexer (**Figure 40**) has 8 sensor signals multiplexed and selected by the signals; MUX1, MUX2 and MUX3. The output can be seen at D.

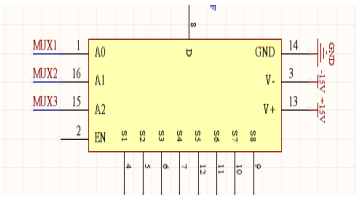


Figure 40: Final stage multiplexer

The multiplexer selected for this function is Vishay Siliconix's DG408DY. It was selected as it met the requirements seen in **Table 5** below:

Requirements	**DG408DY**
Switching Frequency of 100Khz(ADC max sample frequency)	Absolute max switching frequency of 6.25MHz
Input of 0-10V	Input signal range of ±15V
Selection signals should be compatible with the processor.	TTL compatible logic
Reasonably priced and stocked	£1.48 per unit price and over 300 in stock

Table 5: Final stage multiplexer requirements

Further information about this multiplexer can be found in **Appendix C5: DG408DY Datasheet**.

ADC

One of the main stages identified from the literature review was the conversion of analogue signals into digital signals. There are many types of ADC that could accomplish this. Some specialise in sampling speed such as a flash ADC where others specialise in accuracy. For our application a successive approximation ADC is suitable as it provides the accuracy and speed required. Successive approximation ADCs can be purchased as ICs and incorporated into PCBs easily.

The requirements for the ADC were received at an early stage from the fault analysis and detection team during their research. These requirements can be seen below in **Table 6** and the original research can be seen in **Technical Objectives: section 1.1**.

Requirements	**AD7680ARMZ**
Sample frequency per sensor: 2KSPS	100kSPS which equates to 6.25KSPS per sensor
Resolution 16BIT	Resolution 16BIT
Communications SPI or I^2C	SPI

Table 6: ADC requirements

The device selected was the Analogue Devices AD7680ARMZ further information can be found in **Appendix C6: AD7680ARMZ Datasheet**.

DSP Device Selection

Initially it was clear that for the project to meet the requirements there had to be some digital signal processing. There are typically three devices used for this application; a microcontroller, a digital signal processor (DSP) and a field programmable gate array (FPGA). Each one of these solutions have their own disadvantages and advantages but the way to understand which one is needed for this application further information should be presented. The application requires:

- The ability to apply a digital filter within minimal latency due to real-time fault detection requirement.
- To operate as a standalone system if the control board fails.
- Operate the multiplexers with at least 12 digital IO.
- To have an evaluation board so software development can continue alongside hardware development.

FPGAs and DSPs are optimised for digital filtering. Although a microcontroller can achieve this, it usually takes up large resources on such devices. One desirable feature was USB connectivity. This would enable the device to operate as a standalone system without the attached sbRIO by attaching it to a computer. This would allow data from the custom DAQ to be accessed without the sbRIO in instances where it was for some reason not working. All three devices can support or be interfaced with USB so this did not narrow down the decision. There are also many devices with more than 12 digital IO so this was considered as a non-limitation factor. Most manufacturers provide evaluation boards for their devices although these range in cost and ability.

In the selection of which device to use some non-functional specifications were taken into consideration. These non-functional requirements are:

- Should be low cost
- Have in team expertise to minimise learning curve time
- Free to use IDE
- Have many examples to facilitate programming

For a microcontroller the 8051 series (**Figure 41**) seemed sufficient, cheap and came with an evaluation board and an in circuit debugger.

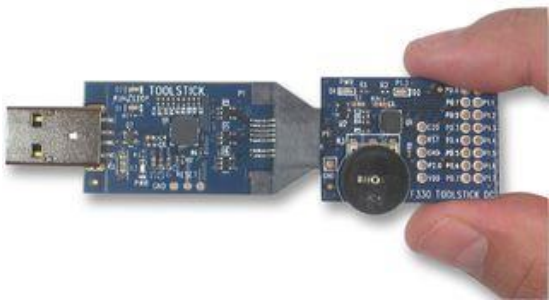


Figure 41: MCU ToolStick Invalid source specified.

FPGAs easily met all requirements although usually USB programming for these devices is sometimes difficult. It was decided as a team that FPGAs where to expensive and overkill for this application. The sbRIO has a FPGA so the project can still utilise the functions given by this device without needing another purchase. During DSP selection a meeting was taken with the University's digital signal processing specialist Professor Patrick Gaydecki. The meeting ended with the result of using an EZDSP USB stick shown in **Figure 42** below.

Figure 42: EZDSP stick

This choice was based on a low price, easily accessible IO pins for PCB mounting and meeting of all project requirements. Both the DSP and 8051 MCU where ordered for evaluation.

The initial solution was to use the microcontroller for basic digital IO manipulation and communications with the sbRIO whilst the DSP would focus on digital signal processing functions such as filtering. Further iterations of the design resulted in the conclusion that the TMS320C5515 (the specific DSP) was easily capable of doing all the functions that the microcontroller was able to-do with low resource use. Therefore the decision was made to use the TMS320C5515 in the project and to use the microcontroller with its large example collection as a hardware debugging tool.

DSP program

The DSP is used to control everything on the custom DAQ PCB. It is programmed in C code. The program is complex and required two team members. The program was dissected and split up into numerous simpler subprograms. These subprograms were required to:

- Send control signals to the multiplexers.
- Read and write data from the ADC and sbRIO.
- Calculate the speed of the encoder.
- Read temperature measurements.
- Apply a digital filter to the sensor data.

The subprograms were then combined to create the final program. The final program flowchart can be seen in **Figure 43**. This figure shows the complete operation of the DSP and its two main parts; the interrupt (executed every point a timer reaches 0) and the main program. The timer is set with the required sample frequency that is received as a constant from the sbRIO on startup. The main program waits in a loop until data is ready to be sent. The DSP then applies a finite impulse response (FIR) filter and then sends the filtered sensor signal data to the sbRIO.

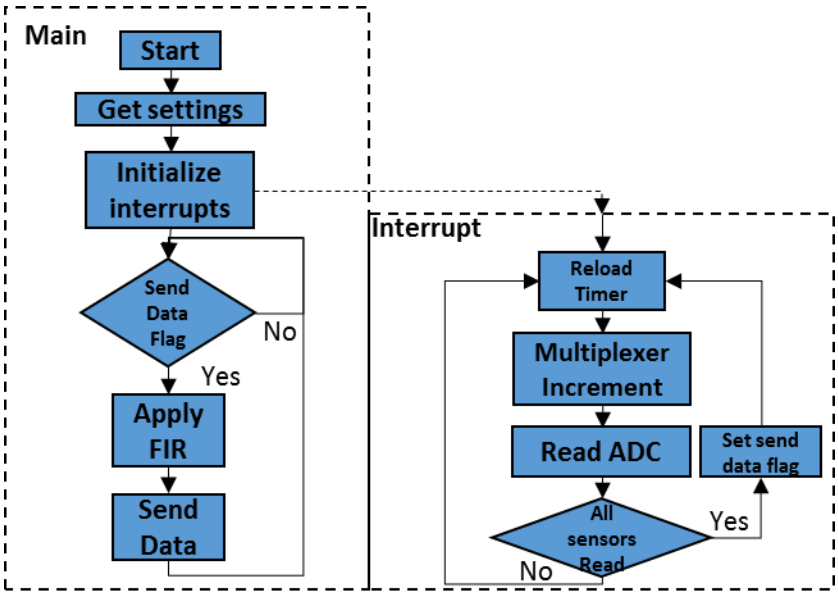


Figure 43: Combined program flowchart

The source code for this program can be found in **Appendix S4: DSP source code.**

Splitting this program up it allowed the engineers to debug each subprogram individually and increased the likeliness of a simple integration at the end. It allowed the engineer to focus on specific tasks, and complete them efficiently to a good standard. The following sections will explain each of these subprograms with the required formulas and algorithms needed.

First a start-up LED test as suggested by the instruction manual was completed. This used a program stored in the IDE examples. If the DSP worked properly, the LEDs blinked in a pattern. To continue working, further familiarization to the DSP IDE program was required. This process required extensive research, considering the complex nature of the DSP and IDE program. This research involved close inspection of manufacturer documents (**Appendix C7: ezDSP Technical Reference, Appendix C8: DSP User Guide**), testing, understanding examples, and reading through tutorials.

The DSP board came with a default program stored in the NOR flash memory. Whenever the DSP is restarted, this program overrides any program stored in the DSP before it is restarted. This program is called the boot-image program.

To create a boot-image, a hex conversion utility program was used. This came with the DSP software in the form of ‘hex55.exe’. The ‘hex55.exe’ was copied to the folder where the boot image was to be made. Then a DOS command was entered which creates the boot image file. The programming of the boot image was also done through a utility program downloaded from the manufacturer’s wiki. This program, when loaded has well defined steps to burn the boot image into the NOR memory of the DSP board.

When the DSP is programmed, it remembers some of the settings from the boot image program. This is why initially the DSP board was burned with a blank program. The purpose of this was to clear the settings in the default program. Since the final system will not have the default program, these settings will have to be coded in the final program.

SPI

The code to implement a master SPI controller on the DSP was programmed from scratch. The manufacturer's document shown in **Appendix C7: ezDSP Technical Reference, Appendix C8: DSP User Guide** and **Appendix C9: SPI Application Notes** were referred to for the programming of SPI

The first part in SPI is to set the SPI clock frequency. For this system, the SPI input clock is used. This is already enabled in the default settings of the DSP. The SPI clock frequency is set through a programmable clock divider. The SPI input clock frequency, which in this DSP is 1.2MHz, divided by the clock divider gives the SPI interface output clock (SPI_CLK). In this system the SPI_CLK is a constant sent from the sbRIO, set by the user. Therefore the clock divider is calculated to give the specific SPI_CLK that is required. This is calculated by the following equation:

$$CLKDV = \frac{SPI\ input\ clock\ frequency}{SPI\_CLK frequency} - 1 \qquad \textbf{Eq 25}$$

The higher the SPI_CLK frequency, the better the result will be. The highest frequency that may be required for this system is calculated below:

$$\underbrace{100KHz}_{maximum\ sampling\ frequency\ of\ ADC} * \underbrace{16Hz}_{resolution\ of\ ADC} * \underbrace{2}_{simultaneous\ use\ by\ ADC\ and\ sbRIO} = 3200KHz$$

The SPI clock is disabled before the frequency is set and enabled after that to avoid errors in settings.

The clock phase is set to transmit at rising edge of the clock signal. The clock polarity is set so that the SPI_CLK pin is high when data is not being transferred. The polarity of the chip select pin is set to active low.

The units of data in the SPI module are characters and frames. The character is the smallest unit of data transferred and frame comprises of a number of characters transferred continuously. The character length is 1 more than the value set (CLEN+1).In this case it is set to 15 which means the character length is 16 bits. This is set to 16 because each signal is represented by 16 bits in the complete system. The frame length is also 1 more than the value set (FLEN+1. This will be set to 16 because each frame represents a sensor and the DSP will be connected to a maximum of 16 sensors. This means each frame consists of 16 characters, or 256 bits. The following diagram in **Figure 44** illustrates this.

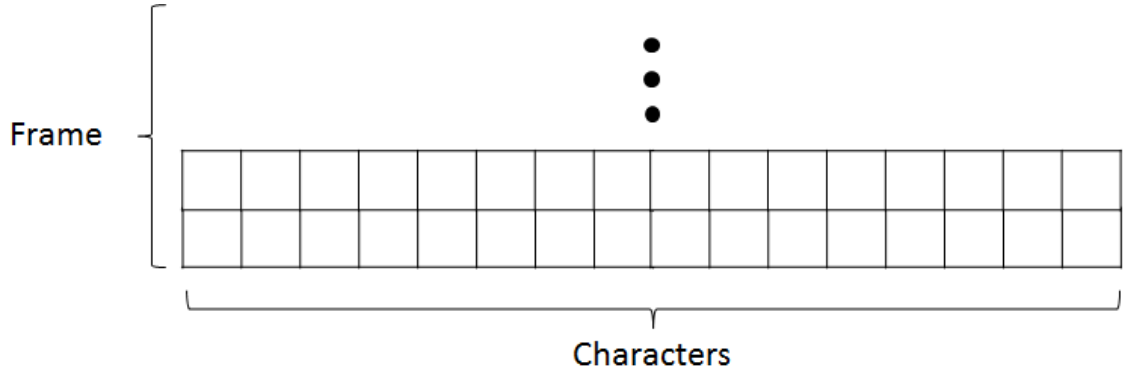


Figure 44: SPI Units of Data

The chip select function is used to select the chip out of the four chips available. It acts as the trigger for the sbRIO to start the data communication on the selected chip. This is discussed in more detail in **Technical Objectives: section 3.3.5.**

The final setting is the SYS_EXBUSSEL setting. This allows 6 different modes for different configurations of the 21 external pins. This is set to mode 1 which has 7 signals of the SPI module, 6 GPIO signals, 4 signals of the UART module and 4 signals of the I2S2 module. This mode is adequate for the system because it needs 4 SPI modules and 1 GPIO module.

The data transfer is only started when the SPI module is idle. There are two registers (SPIDAT1 and SPIDAT2) of 16 bits each which act like a 32 bit shift register. The data enters into SPIDAT1 during a read and shifts into SPIDAT2 if it is more than 16 bits. During a write the data exits from SPIDAT2 while the data in SPIDAT1 shifts into SPIDAT2. This is illustrated in **Figure 45** below:

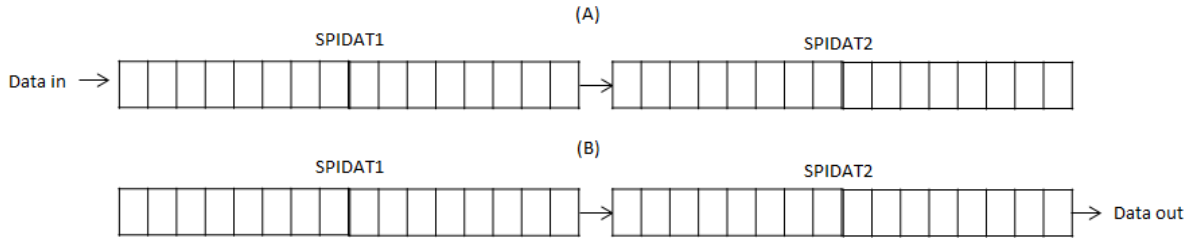


Figure 45: (A) Read; (B) Write

The SPIDAT1 and SPIDAT2 registers need to be cleared before a read command to avoid corruption of data. Before a write, the data that is going to be transferred needs to be stored in SPIDAT2 register.

The source code for SPI can be found in **Appendix S4: DSP source code.**

The SPI has been tested with the sbRIO. This is shown in the test document (**Appendix C10: Test document**). This test was done on a single packet. Further testing on multiple packets will be done in the near future before the SPI is integrated into the final system.

Temperature/ADC

The temperature sensor is the only sensor directly connected to the DSP via the internal ADC module. This is because the temperature changes slowly, therefor a high resolution, fast sampling ADC is not required. The program was developed to use channel 3 analogue input. This input is the only analogue input that doesn't have a potential divider as this had already been implemented in hardware during phase 1. To implement the ADC on the DSP, channels need to be selected along with analogue signal pins, which are distinct from digital pins. A read from the ADC is requested in a loop. After this the program waits until the sample complete flag is given and the loop is executed again.

When this sub-program was merged it was implemented in the final block of the combined program flow chart (Send data) **Figure 45**. There it is placed with similarly sampled data from the off chip ADC and then sent to the sbRIO. This implementation of using the onboard ADC for slower changing signals is suitable and allows an increase in utilization of the DSPs resources instead of using additional hardware, which would increase the cost and size.

The flowchart for this sub-program is can be seen below:

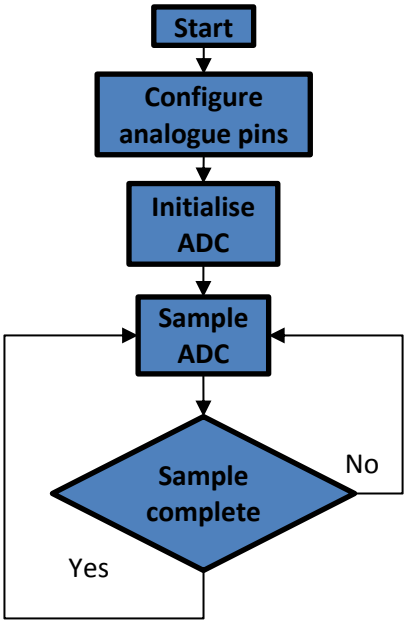


Figure 46: Temperature ADC flowchart

The source code for this can be seen in **Appendix S5: Temperature Sensor source code.**

This code was programmed with use of the manufacturer's application notes and user guides provided for the TMS320C5515 Digital signal processor.

FIR filter

The main advantage of the digital filter over the analogue filter in this system is that the user can set the cut-off frequency through the GUI. It also allows the user to easily set different types of filter with superior signal to noise ratio. This is relatively difficult to do in hardware and would require the user to reassemble the system with significant changes in the circuit designs.

There are two main ways to design digital filters; finite impulse response (FIR) filters and infinite impulse response (IIR) filters. Unlike IIR filters, FIR filters are never unstable and they give linear phase output, which means the phase of the output is not distorted by the filter [30]. Therefore, for this reason and others, they perform better than IIR filters and are used more commonly. The disadvantage of FIR filter over IIR filter is that it uses more memory [31]. This is not a problem in this system because the DSP has much more memory than is needed for the system. Therefore, FIR design method is chosen for the filter.

An FIR filter filters a set of samples of data at a time. This means that he the filter is applied to a finite part of the signal, ignoring the signal before and after the set of samples.. This is called truncation. Each sample is multiplied with a coefficient and the result is stored as the cumulative sum. That is, the signal is convoluted with the coefficients. This gives the filtered output. This is shown in the equation below:

$$\boldsymbol{y[k]} = \sum_{\boldsymbol{n=0}}^{\boldsymbol{N-1}} \boldsymbol{h[n]} * \boldsymbol{x[k-n]} \quad \textbf{[34]} \qquad \textbf{Eq 26}$$

y[k]=filtered output; h[n]=impulse response; N=number of taps or number of samples

The main task involved in designing an FIR filter was the generation of the coefficients. There are several methods that can do this. There were two that were considered. The first choice was the Parks-McClellan method. Research showed that this method allowed more customizability to the user than other popular methods and therefore is

widely used [35]. Therefore a C code example to implement this method was found. Since this method is more customizable, the settings that it asked for from the user were extensive. After some analysis of the code, it was found to be overcomplicated and consumed more time than expected to use in the system. Additionally, this system requires a simple low pass filter to remove high frequency noise, making the customizability redundant.

The second method found was the window method. In this method, a window function is multiplied with the coefficients. Multiplying the coefficients with the samples without a window function effectively means that a rectangular window is used. The abrupt truncation of the signal causes large ripples in the response. This leads to errors in the output. To reduce this effect, different kinds of windows are used to provide gradual truncation [30].

The example code for the window method was found in a presentation by Dr. Naim Dahnoun, Bristol University [34], and modified (**Appendix S6:** MATLAB **FIR code**). It uses the Hamming window. This example is in MATLAB code so it had to be converted to C for the DSP. One option to convert the code was to use the MATLAB coder. However, this was not available. The next best option to convert the code was to code it manually. The algorithm involved array calculations which, unlike MATLAB could not be done directly in C. Therefore, the calculations were done in for-loops. The for-loops were minimised to make the program more efficient and faster. After the code was complete, it was tested in MATLAB and C. A sine wave of two frequencies, 500Hz and 2500Hz was generated in MATLAB and input into the filter with cut-off frequency set at 2000Hz. The frequency spectrum of the sine wave is shown in **Figure 47(A)** and the frequency spectrum of the filter with the cut-off frequency is shown in **Figure 47(B)**.

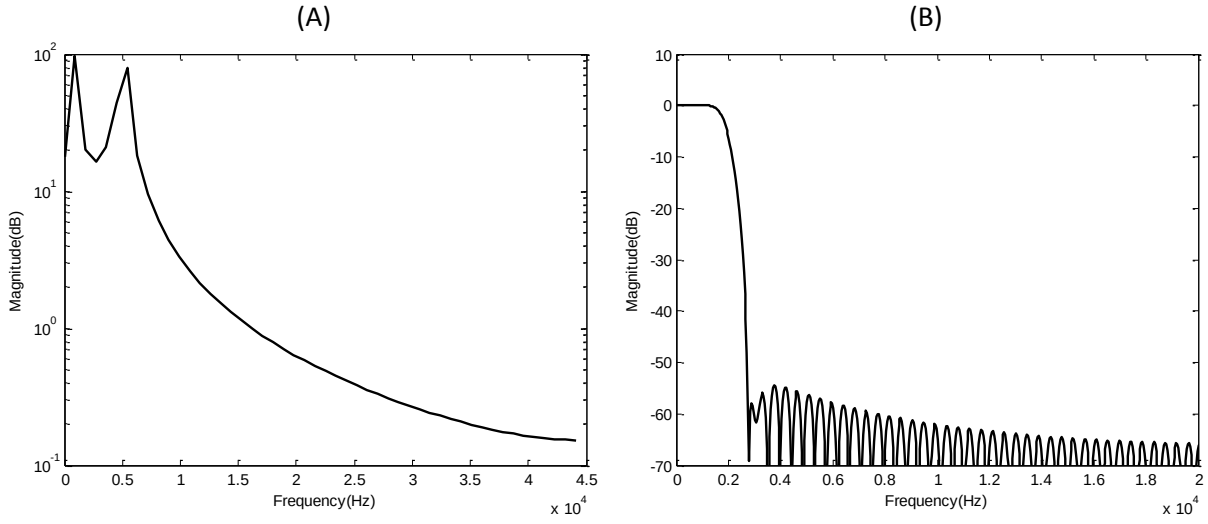


Figure 47: (A) Frequency spectrum of input sine-wave; (B) Frequency spectrum of filter cut-off 2KHz

The results were analysed extensively in time domain. However, in the time domain the filtration was difficult to detect. Therefore, the results were analysed in the frequency domain, which required extra coding in MATLAB. After extensive testing and debugging, the MATLAB code showed positive results. This meant that the MATLAB code did not have any problems, and any problems found in C code would have been in the translation process. After testing and debugging the C code until the results of the MATLAB code and the C code matched. They are show in **Figure 48**. The C code result is graphed in MATLAB by converting the result into an array.

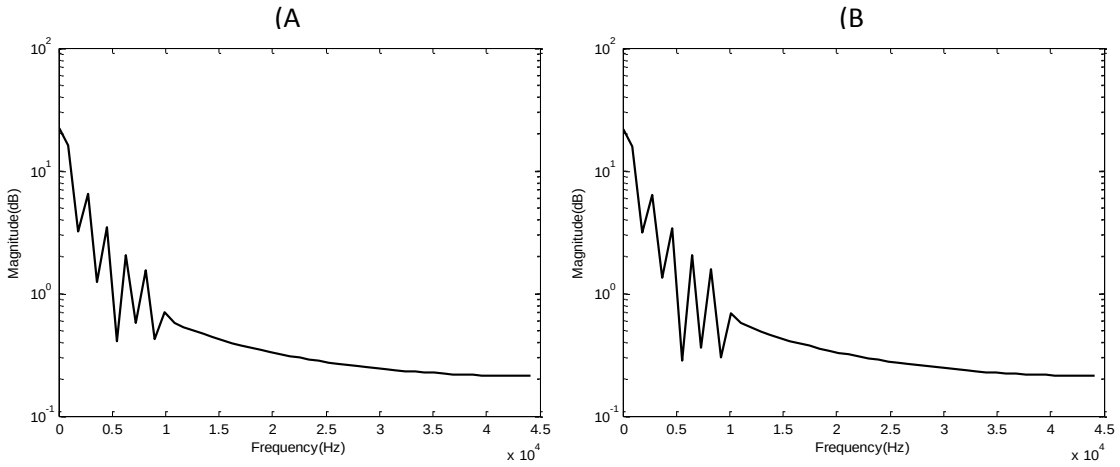


Figure 48: (A) MATLAB **output; (B) C code output**

The final structure of the FIR filter code is shown in the flowchart in **Figure 49** below:

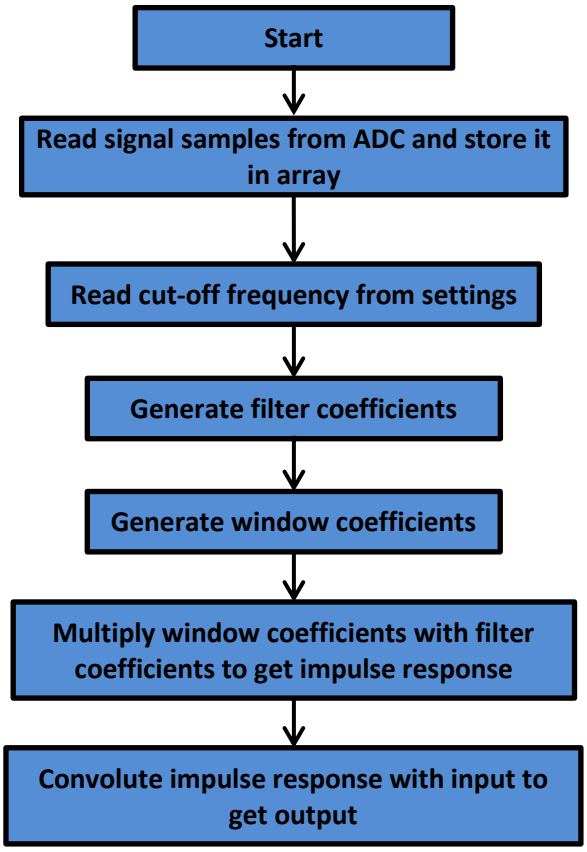


Figure 49: FIR program flowchart

The source code of FIR filter can be found in **Appendix S4: DSP source code.**

Multiplexer

The multiplexer program operates within the timer interrupt. It is needed to latch the multiplexers at the required position when required. There are two multiplexers that are used within this hardware design and they both need to be reconfigured when a new sensor is sampled. The multiplexer configuration needs to take the enabled sensor

settings that are received from the sbRIO into consideration as this affects the maximum allowable sample frequency. The multiplexer sub-program can be seen in **Figure 50**.

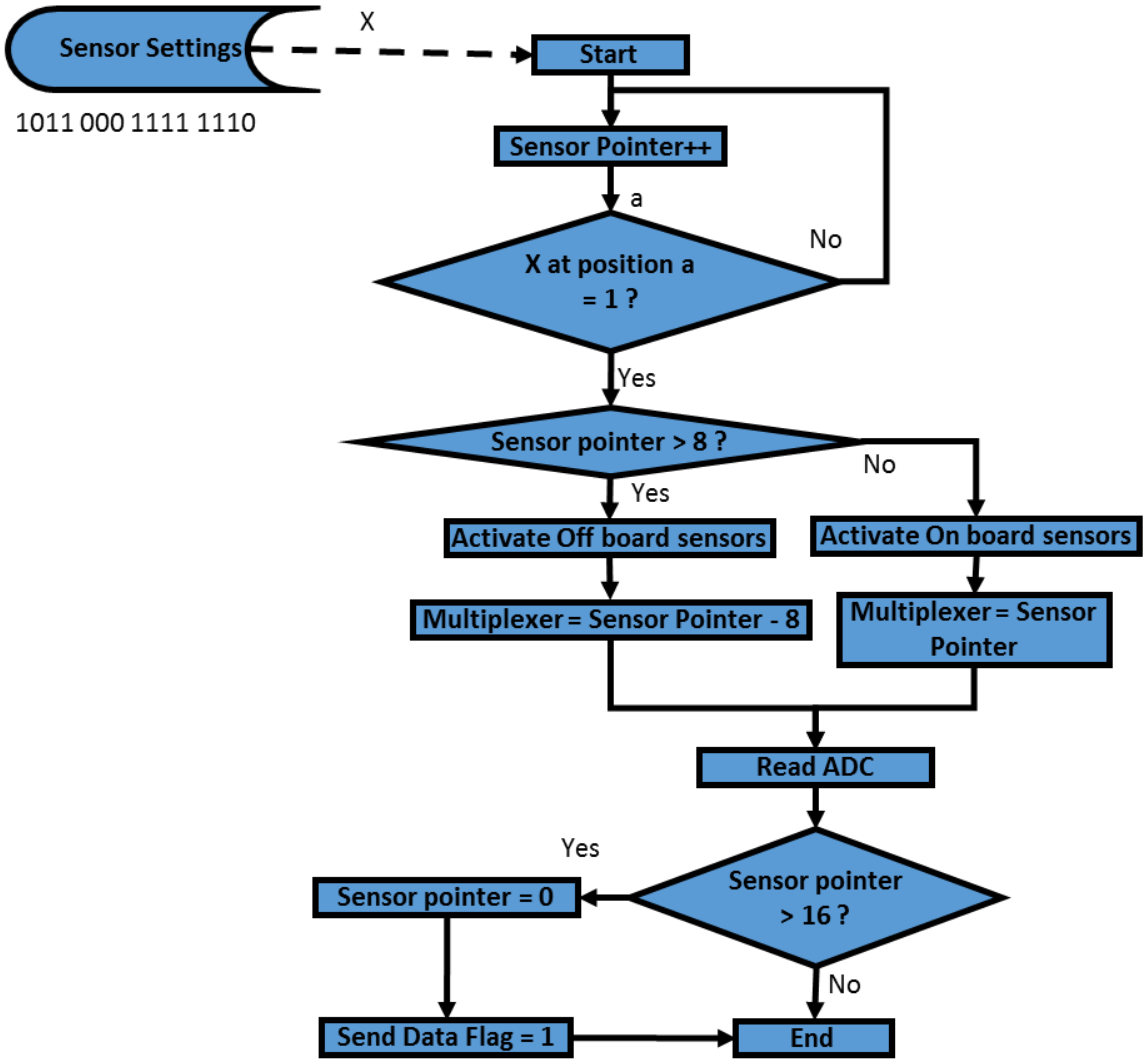


Figure 50: Multiplexer flowchart

This code is executed at every timer interrupt which is set depending on the sample frequency. For instance, if a sample frequency of 2kHz is required for each sensor then the timer is loaded with a value that takes 0.5ms for the timer to hit 0. In the program the timer is set with no pre-scaler, therefore the timers clock is 120MHz. The formula required to identify the reload value of the timer can be seen below:

$$\boldsymbol{Timer\ reload\ value} = \frac{\boldsymbol{Clock\ Frequency}}{\boldsymbol{Sample\ Frequency}} \qquad \textbf{Eq 27}$$

In this case the timer reload value would be 60,000.

When the ADC is read, data is stored in an array at a position defined by the sensor pointer. The sensors that were deactivated according to the sensor settings are then removed from the array ready for the data to be sent by the main program.

Encoder

The encoder program was designed around the notion of using a timer to measure the rising edge of the pulse received from the encoded. To get a more accurate measurement several pulses where averaged. This decreases the update rate. The flowchart for this program can be seen in **Figure 51** below.

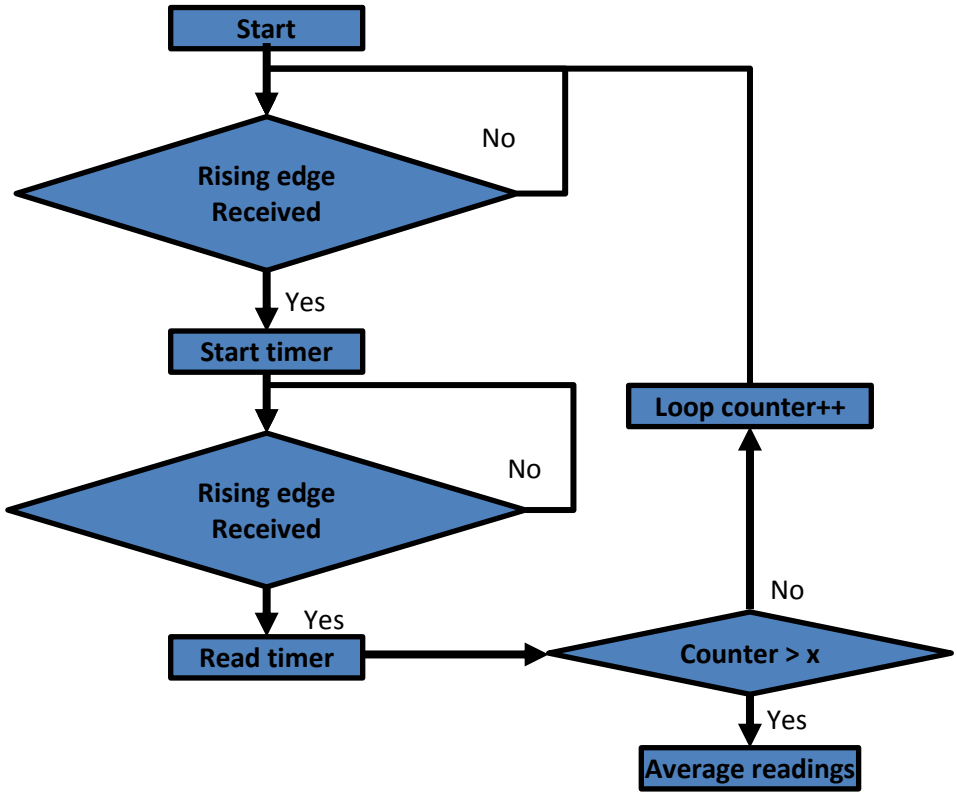


Figure 51: Encoder flowchart

The program has been designed to execute when the measurement is required. This is at the end of the collection of sensor data prior to sending the data to the RIO. The code polls a pin for a rising edge from the rotational encoder. Once the rising edge is detected the program starts a timer which counts up. The program then polls for the next rising edge and then the timer is stopped. This value is then stored until X amount of timer values are stored and an average is taken.

This encoder firmware development was set to low priority as the sbRIO already had a module for encoder interfacing. This code was envisioned to complement the rest of the firmware and to provide an interface to the final required sensor. Although it seems redundant it actually gives contingency allowing for a “plan b” scenario.

Full sensor interfacing decreases the reliance on the completion of the sbRIO. If the sbRIO was left uncompleted impeding the function of the final system, an alternative approach can be taken fairly quickly. This approach would be to use the USB functionality of the DSP to communicate the sampled digital filtered data to a computer. From this a MATLAB script can interpret the signals and pass them to the script discussed in **Technical Objectives: section 1**.

Although “plan b” decreases the flexibility of the system and requires a computer it still allows the demonstration of 2/3 of the project’s progress.

Testing

Every aspect of the custom DAQ had to be tested step by step until the final system is integrated. This is seen as the bottom up approach and allows the detection of design errors at an early stage. Once each component is tested there is a final system integration testing procedure that tests all of the components interactions. Phase 1 was tested separately to phase 2 to encourage integration at an early stage. Phase 1 concentrates on analogue circuitry that are notorious for not being ideal and having different than expected values. Phase 2 is primarily digital and has a 50% firmware component that also requires testing.

A document explaining the procedure, expected output and measured outputs from all the testing is given in **Appendix C10: Test document.** This document was compiled to allow the engineer to inspect the problem and suggest a solution. A select few of the tests will be presented here in full to allow the reader to fully understand what was undertaken during this task.
The components selected to be presented in full are:

1. Analogue filter input.
2. SPI read and write testing.

The first stage to any of the testing done in this project is creating an ideal environment. This eliminates any unknowns in the testing procedure and allows the engineer to inspect the characteristics of the component being tested. This ideal environment for the analogue filter was a signal generator producing a swept frequency sin wave on the input. An oscilloscope was connected to the output to inspect the sine wave's attenuation applied by the filter. This set-up is shown in **Figure 52**.

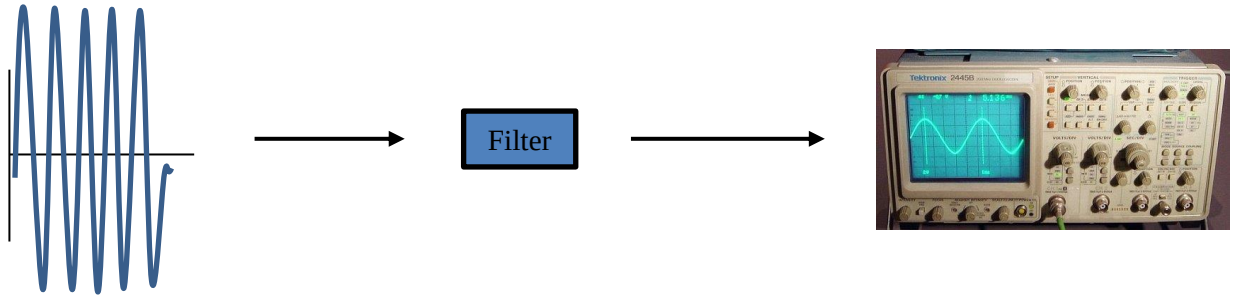


Figure 52: Filter test setup

The SPI testing environment was a connection to a cDAQ with an oscilloscope program loaded to view the four signals. This set-up is shown in **Figure 53**.

Figure 53: SPI test setup

The setup shown above can only test a write to the SPI. As the write function is part of the same SPI software stack it was assumed that the read function was a simplistic change in command and testing of it separately was not required.

The tests on the filter design showed that the signal was being attenuated much earlier than calculations predicted. The SPI test showed the communication frequency was set to high in the firmware and therefore the signal fall time was not met and gave oscillations and skewed waveforms.

The solution for the analogue filter is documented in **Appendix C11: Phase 1 Changes required.** To summarise this solution the capacitors used in the filter design had a tolerance of 10%, the effect on the RC filter meant each filter attenuated at a different value. This was solved by using a more precise capacitor in the final design. The SPI speed was initially measured at 63 MHz by reducing the speed down to 6 MHz the transitions became non-oscillatory and non-skewed this result could also be seen in all signals less than 32 MHz which leaves a large bandwidth if requirements for this communication link are increased.

PCB Development

All the hardware discussed had to be implemented on a PCB designed in Altium Designer. During PCB development there were many considerations, which are shown below:

- Clearances – as voltage can breakdown the air between the tracks and short.
- Track thicknesses – as current can heat up tracks and damage the PCB.
- Power planes – to increase routing flow, EMI rejection and minimise ground loops.
- Analogue and digital circuit placement – to reduce interference.
- And mounting whole placement – So the PCB may be placed on top of the sbRIO.

Online tools were used to calculate clearances and track thicknesses. **Invalid source specified.**. They are all based on Maxwell's equations. A ground plane was added via a polygon pour on the bottom layer. The analogue and digital circuits were placed on different sides of the PCB, thus using the PCB material FR4 which has suitable electromagnetic attenuation characteristics. The mounting holes where carefully measured on the sbRIO and mechanical drawings where inspected. The holes were placed on the PCB before component placement and routing. The outcome can be seen in **Figure 54:**

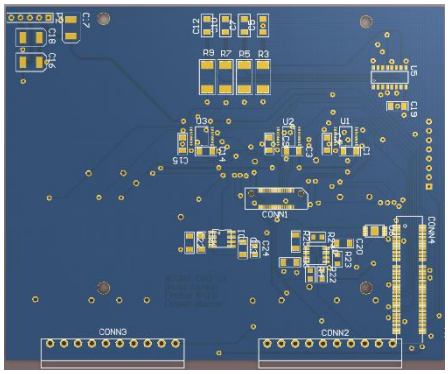


Figure 54: PCB 3D view

Approximately 90% of the 102 components assembled on this PCB had to have footprints created by the students as they did not exist in Altium designer's library.

Final hardware testing

The hardware test procedure and results are shown in the testing document (**Appendix C10: Test document**). The four circuits tested were; overvoltage circuits, low-pass filter circuits, multiplexer circuits and the opto-couple circuit.

The overvoltage circuit test failed. The voltage was meant to clip at 5.1V but in the test it clipped at 5.78V. The reason for that was the design did not consider the voltage of the diode rectifier, which is 0.7V. To fix this, the cut-off was designed to be 4.3V instead of 5.1V.

The analogue low pass filter test also failed. The 10Vp-p was meant to decrease to 7Vp-p at 2KHz. Instead it decreased to 6Vp-p at 2KHz. This means that the cut-off frequency was lower than 2KHz, at around 1.7KHz. The reason for this is mainly found to be in the precision of the components. To fix this the old capacitor of 10% tolerance was replaced with a capacitor of 1% tolerance.

Multiplexer test also failed. One of the select pins was not functioning. The reason for this was that the pin was not soldered. This was fixed by increasing the footprint size. The selection pins were also connected to pull down resistors. In the final design the multiplexer will be soldered in an oven which will ensure that soldering problems do not occur.

The opto-isolator circuit completely failed. This was because the current and the voltage could not be provided with the chosen resistor. After some consideration, this circuit was replaced with a voltage level shifter. The purpose of the opto-isolator circuit was for overvoltage protection. The voltage shifter would fail at higher voltage, providing overvoltage protection. The voltage shifter solution avoids unnecessary complexities and is better value for money.

3.3.4 Embedded Controller

The Single Board RIO (sbRIO) from National Instruments has been used as the embedded controller. This was acquired at the end of semester one. Justification for using it over other platforms can be found in **Appendix G2: Interim Report**. The Single Board RIO has 96 digital lines which have been used for SPI communications, an encoder interface and LED status indicators. Its 100Mb/s Ethernet connection has been used in a connection to a remote platform which it can transmit data to, receive settings from or be controlled in a variety of functions. This platform runs software to communicate with the controller and can be connected to it in a wind turbine or through a network.

All processing on the sbRIO is shared between a microprocessor running a real-time operating system and an FPGA connected directly to the 96 digital lines. The FPGA runs code that interfaces with the digital lines responsible for SPI communications, the encoder interface and the LED status indicators. The microprocessor is responsible for the main functions of the condition monitoring system. An interface with the FPGA allows it to send commands and information to it and receive streams of sensor readings from it. **Figure 55** describes the division of code between the two processing sides and the communications between them.

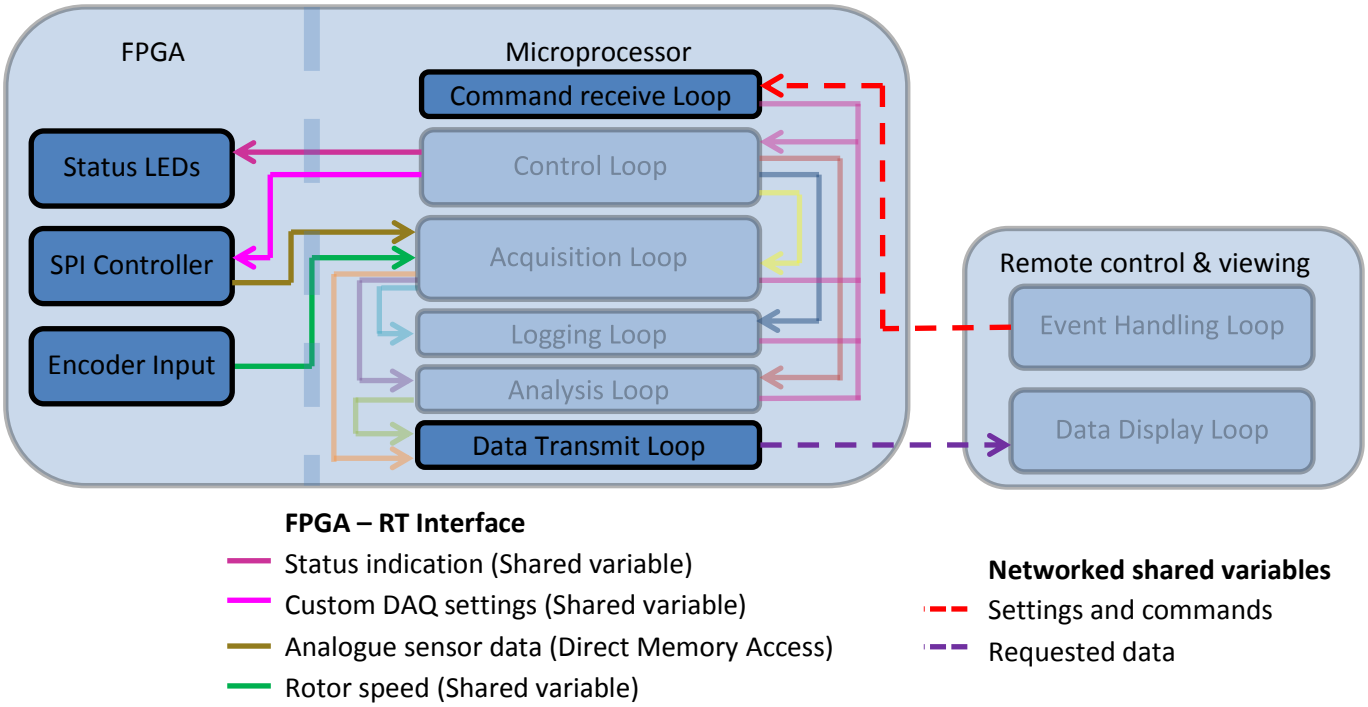


Figure 55: Code modules and connections in two parts of sbRIO and in the remote control and viewing platform

In this diagram, sections that have been faded out are exact or near exact copies of modules originally developed in the prototype system. Non faded blocks show brand new functions that were been made for the embedded controller. The following sections describe the blocks in the two sections of the sbRIO and those in the remote control and viewing platform.

Remote Control and Viewing Platform

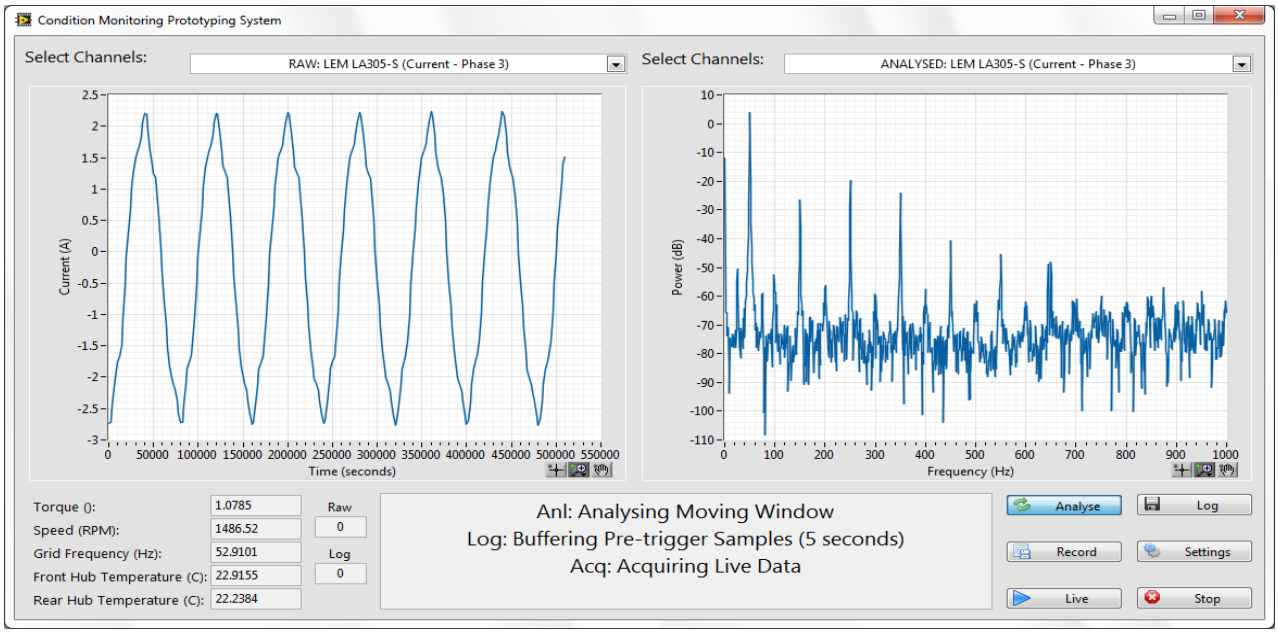


Figure 56: Screen shot of the UI running on a laptop connected to the RIO via Ethernet connection

The remote control and viewing platform re-uses code modules from the prototype system with almost no changes. It uses these with the same user interface design as the prototype system to provide the same functionality. **Figure 56** shows the interface displaying raw and processed data. Further explanation of the

functions on this user interface can be found in **Appendix B5: Prototype and Embedded Controller UI Manual and Code Documentation.**

The main difference to the original code modules developed in the prototype is in the way data is passed to the embedded controller. A series of network shared variables are used to do this. When these are configured the application on the remote platform runs a shared variable engine built into LabVIEW. This enables values of shared variables intended for the transmission of settings and commands to be updated. It also receives changes made to the shared variables that it is listening to which are used to receive data.

Making the change from adding/receiving information to/from queues, to using shared variables instead did not take long to implement. The shared variables use the same data types as the queues so the process was just to swap the queue reference to a shared variable reference.

Retrieving past settings from file is the one form of data communication not solved by shared variable. Unfortunately the large cluster of many different data types used to hold system settings is not compatible with shared variables. To get around this problem the remote platform applications runs a startup procedure that does two things:

1. Uses the file transfer protocol (FTP) to retrieve the controllers current settings from their set location. Then saves these in a local register for later use.
2. Request the current status from the controller. This would not be received until something changed if it were not requested on start up.

After this the remote platform will run as it did in the prototype system.

Microprocessor

Figure 55(above) shows the code modules that run on the microprocessor. The faded blocks in the centre of the diagram show where near exact copies of the original control and functional loops developed in the prototype have been used. The only changes that needed to be made to these are in the sections were code uses hardware or local storage.

- Hardware: In the prototype system, data came into the acquisition loop using the CDAQ. Now that data enters the system through the FPGA changes include.
    - “Start” and “Stop” states in the acquisition loop that used to initialise the CDAQ, now have no function.
    - “Update Settings” state which used to re-configure the CDAQ, sends new settings to FPGA through shared variables and then raises a flag to indicate they should be sent to DSP.
- Local storage: In the prototype system the root folder for logged data and settings could be changed by the user. In a deployed system there is little sense in this and therefore the control to change these was changed to a static value.
- Retrieval of logs: As the embedded system stores log files locally there needed to be a means to grab log files. No extra code is needed for this as the sbRIO hosts a web interface through which log files can be retrieved using FTP.

Two new blocks shown in **Figure 55**(above) were developed for the microprocessor to replace the event handling and data display loops. Like the loops they replace that were moved onto the remote platform, these have low complexity. A network shared variable engine is run on the sbRIO as it is on the remote platform and these two new loops use this to complete the remote link. The commands receive loop listens for new commands and passes

them onto the existing control loop queue. The data transmit loop sends data using the requested data shared variable.

Testing of the embedded system code has shown that the functionality designed for it in the prototype has been successfully transferred over. Testing without a link to the custom DAQ, which is still under development, has been made possible by using the playback feature built into the embedded software. This is done by recording data in the prototype system then copying the files to the sbRIO for playback. When communication between custom DAQ and sbRIO has been achieved, extra work will be required to move from recorded to live data. Code modules for the FPGA-RT interface have already been developed so integration should only require making small changes and then testing.

Though the final MATLAB fault detection software is now complete its delivery has come too late to integrate and test in the embedded controller's software. Tests have been possible by using FFTs in LabVIEW as they were in the prototype system. These have been used to test the communications channel o the remote platform but not the performance of the system. To fully test the system and tune analysis parameters for optimum results, future work will include the integration of these two pieces of software and further tests.

All code developed for the microprocessor in the embedded system can be found in **Appendix S7: Embedded Controller Source Files**.

FPGA

The following sections describe the program written for the FPGA module as mentioned in the **Figure 55**. A manual for each written programs in these sections are documented in the **Appendix B5: Prototype and Embedded Controller UI Manual and Code Documentation.**

Encoder

The encoder produces a square wave pulse that corresponds with the rotational speed of the wind generator turbine. Depending on the rotational speed of the turbine, the period of the square wave can be either high or low. By measuring this period, the rotational speed of the turbine can be calculated.

The LabVIEW program measures the square wave pulses produces by the encoder. The pulses entered one of the digital inputs in the sbRIO and the program written for the FPGA processes it. The program uses a detection technique for the rising and the falling edge of the square wave to obtain the period of these pulses.

This technique works by detecting the first rising edge of the square wave pulses. After that, it will start the first counter. This first counter return the time for the duration of the program before the first rising edge occurred. The program then continues by detecting the second rising edge of the pulses before starting another counter. The second counter returns the value of time for the duration of the whole program. The time difference obtained from the returned value of the second and the first counter shows the period for one pulse of the square wave signal. However, the encoder produces a maximum of 1024 pulses. This means that to obtain the period of the whole square wave pulses, the period obtained from before need to be multiplied with the value of 1024. **Figure 57** below depicts the process inside the sbRIO.

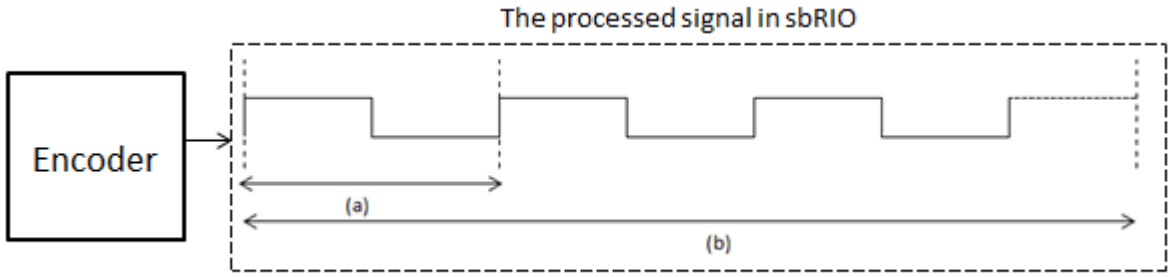


Figure 57: Pulses generated by encoder where (a) is the measured period of one pulse and (b) is the measured period of the pulses specified by user

The previous discussion only consider the period of one pulse. But to get a more accurate reading from the encoder, the amount of measured pulses need to be increased. The program allows the user to specify the number of measured pulse and then using this number to be divided with the value of 1024. The obtained result from this division is then used in the multiplication process with the measured period to obtain the period of the whole square wave pulses. This final result is a more accurate value of the signal's period. The equation used for this process is as followed.

$$\boldsymbol{period\ of\ 1024\ pulses}, \boldsymbol{T} = \frac{\boldsymbol{1024 \times t_m}}{\boldsymbol{n}} \qquad \textbf{Eq 28}$$

t_m=period of the specified pulses; n=number of measured pulses

The calculated period from the above equation is then inverted to obtain the value of the rotational speed of the turbine in rpm.

The code for encoder is tested by linking the output of the encoder attached onto the test rig to one of the digital input of the sbRIO. By changing the supply voltage of the power supply for the test rig, the speed of the turbine can be varied. The resolution of the rotor speed measurement is changed by varying the amount of measured pulses. Increasing the amount of the measured pulses increases the resolution of the measured speed allowing the user to change the required resolution.

LCD graphic display

The solution for an alarm signal in fault detection is to use an LCD capable of displaying the type of faults and limited amount of information on the LCD. However, due to several faults, unknown errors and a time constraints, this solution is unable to be implemented in the system. This section describes the troubleshooting done to figure out the problem with the LCD graphic display.

Solution 1: An LCD driver was downloaded from the National Instrument website known to be compatible with the purchased LCD screen. However, an error occurred when uploading the driver to the FPGA module. The error was brought to attention National Instrument (NI) through several emails and phone calls. An engineer from the NI had been sent to get a further look on this error but no solution for it had been found. After debugging by the student working on this task produced no results, this solution had to be dropped.

Solution 2: Another LCD driver was found in the NI forum and was downloaded to be used with the purchased LCD. However, the same type of error as with the previous solution was found in the downloaded driver. The error with this driver was fixed after a week of debugging. But the LCD was still unable to communicate with the establish driver for unknown reason.

Solution 3: A request was made to the electronic workshop to use an LCD from the second year project. When the LCD was tested with the driver from solution 2, it did not produce any sort of display. So the team decided to try using a microcontroller to program the LCD instead of the sbRIO t. By using an obtained C code that came with the microcontroller, the LCD was tested with it. However, the LCD was still unable to display any character in its screen.

Solution 4: The final solution was to use the pic board and the C code which was used from the second year project to program the LCD. However, it still produces the same result as solution 1 and solution 2.

After four weeks of unfruitful result with the LCD, an alternative solution was proposed. The alternative solution was to use LEDs to signal the faults. The extra information of the fault is no longer displayed on the LCD but on the GUI instead.

LEDs

Six LEDs were used were each LEDs represents the type of fault that the system is able to detect. The interface between the LEDs and the sbRIO's FPGA is as shown in **Figure 58**.

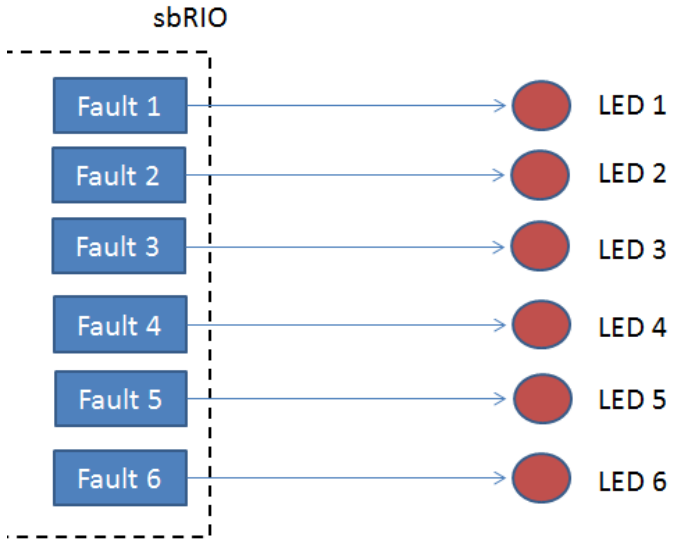


Figure 58: The interface between the LEDs and the RIO

Communication link (SPI and I^2C)

The first solution for the communication link between the DSP and the sbRIO was the I^2C protocol. The solution is based on the DSP becoming a master and the sbRIO as a slave. The difference between these two titles (master and slave) is that the master initiated the communication protocol between the two while the slave waits for the master to send the initiating signal. However, the driver provided by the National Instrument which had been used to program the communication protocol for the sbRIO only provided a function where the sbRIO acts as a master. So further programming of the driver was done to convert the sbRIO from master to the slave. This is done by adding some functions where the program can reads the initiating signal from the DSP and sends the data after an acknowledgment signal was given.

During development the team discovered a limitation of the DPS in using the I^2C protocol which has been described in the **Technical Objectives: section 3.3.2**. A new solution is then proposed where the communication link is changed from I^2C protocol to SPI protocol.

The idea of the solution is still the same where the sbRIO acts as the slave and the DSP acts as the master. The only difference is that the SPI requires four data lines whereas the I^2C only requires two data lines. A new driver for the SPI protocol is also needed. This driver was provided by National Instruments. The provided driver has the function where the sbRIO is the slave and the DSP is the master so further programming from before is not needed. But an understanding on the operation of the driver is required before the driver can be used properly.

The driver for the SPI program requires that the CS signal sent from the DSP to the sbRIO is set to false to initiate the communication process. The DSP can then send data through the SPI channel using the MOSI line. For the sbRIO, this process is called the read function where the SPI program in the sbRIO's FPGA reads the data coming from the DSP. To initiate the write function on the sbRIO, the DSP needs to send the value of '80' through the MOSI line. After reading the value '80', the sbRIO is then allowed to transfer data through the MISO line to the DSP.

A few changes were made to the SPI program to satisfy the requirement of the communication process. The SPI channel must be able to transfer 16 bits data between the DSP and the sbRIO. The changes made for this was by changing one the constant in the driver from the value of '8' to the value of '16'. Another changes made was adding a FIFO data queue into the FPGA. This was to allow the data transfer from the FPGA side to the microprocessor side to happen smoothly and without losses.

An additional program is needed to integrate the SPI channel in the FPGA to the microprocessor. This program is used to send settings required by the user for the DSP and also to process the data coming from the DSP.

Integration of RT and FPGA

This chapter describes the program written for the integration of microprocessor and FPGA in using the SPI protocol. The flow chart in **Figure 59** shows the algorithm of the program. The microprocessor is responsible in controlling the input setting that are sent to the DAQ board through the FPGA. It also processes the data collected by the FPGA that is sent from the DAQ. This data originates from the channel selected by the user using the microprocessor.

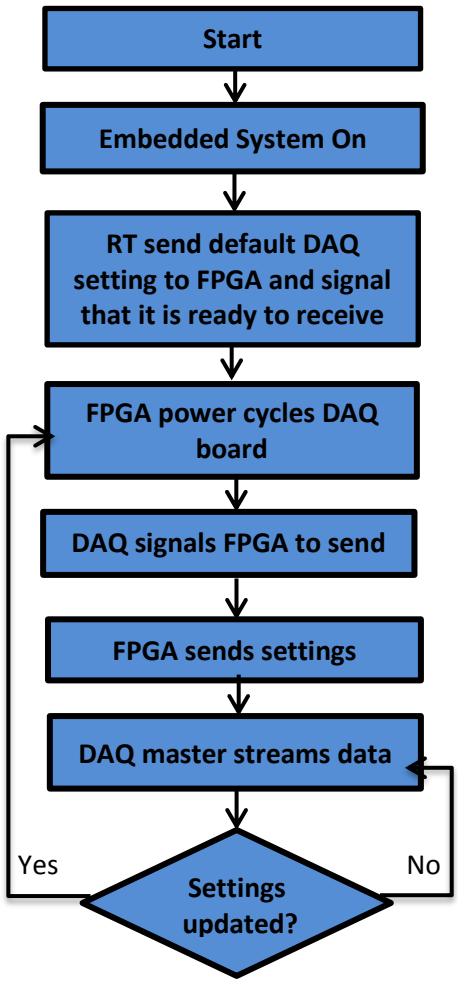


Figure 59: Flowchart of program integration RT and FPGA

To send new settings to the FPGA, the microprocessor creates a Boolean array with the relevant data. The Boolean array is then passed to the DSP on the SPI line using a three packet sequence. The chosen number of bits for each setting is as shown in **Table 7**.

Setting	Number of bits
Sampling rate	2 bits (1kHz, 2 kHz, 3kHz, 4kHz)
Filter Type	3 bits
Filter upper cut off	11 bits
Filter lower cut off	11 bits
Channel select	18 bits

Table 7: Number of bits for each setting

The selection on the number of bits for each setting is decided by the number of options presented by each setting. For the sampling rate, the team decided to use specific sampling frequencies that go from 1 kHz up to 4 kHz in steps of 1 kHz. This means that it has four different arrangements for the bits which make it suitable to assign 2 bits for the sampling rate.

Each of the 3 settings packets holds 16 bits containing information on all the settings, either partially or in full. The third packet has three empty bits at the end which have not been assigned to any value. The reason for having 16 bits in each packet is because the amount of bits that can be transferred through the developed SPI

communication link between the DAQ and the RIO is only up to 16 bits. The representation of the data transfer from the RIO to the DAQ in the three packets sequence can be seen in **Figure 60**.

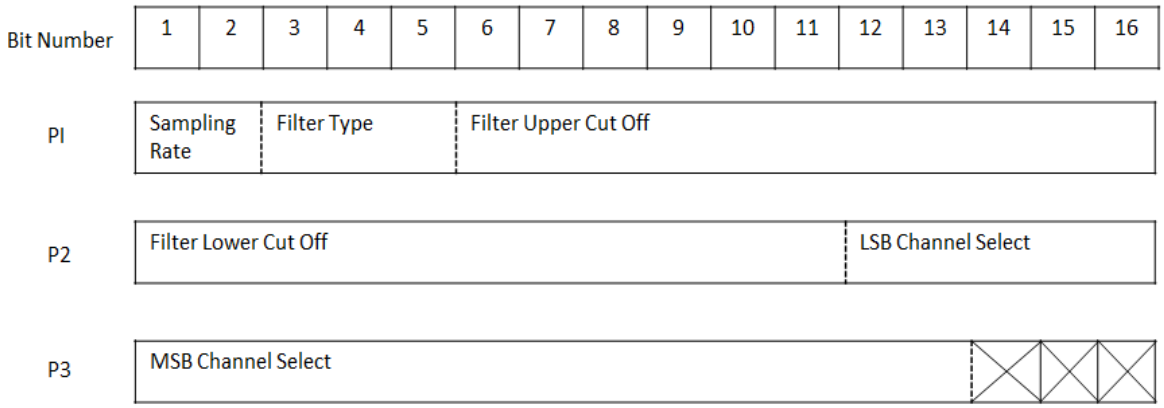


Figure 60: 16 bits representation of data transfer from the RIO (slave) to the DAQ (master)

The channels to be sampled are set by the channel select setting. 18 channels are used in the system. A Boolean array of true and false is used to select each of these channels. The channels that receive a 'true' signal are free to pass will be added to the list to be sampled by the custom DAQ board. The selected channels to be sampled are in the sequence of the least significant bits to the most significant bits. For example, if the channel to be sampled is set as channel 1, channel 2 and channel 3, these three channels will be the input to the Boolean array. It will set from the start of the first three least significant bits in the channel select settings and the sampling begins starting from channel 1, channel 2 and thirdly, channel 3 .The sampling process repeats this sequence until the next setting update.

3.3.5 Power Management Board

In the final standalone system, all system components need to be powered by supplying different voltages to different parts of the board. A solution for this is issue is the development of a power management board. This board needs to supply voltages for the sbRIO, Custom DAQ board, sensors and Digital Signal Processor. The board was developed in the Altium Designer 10 suite and made on a Printed Circuit Board (PCB).

All these boards need a variety of input voltage levels capable of meeting a required current. The team working on the power board liaised with the hardware and sensors teams to get the specifications needed for their various components to operate correctly. These were the major factors in the design decisions regarding the board. A complete list of requirements for the power management board is listed in **Table 8**:

Component Powered	Desired Voltage Output (V)	Current Output (A)
Custom DAQ PCB board	-15	0.2
Custom DAQ PCB board	+15	0.2
RIO	9	2.74 A
Digital Signal Processor	1.3	0.25A
Expansion supply	5V	1.5 A
Custom DAQ PCB board	3.3	0.5

Table 8: List of requirements for the power management board

There is also additional scope for an additional 5V outputs depending on if the user wants an increased number of sensors to the system.

After the necessary specifications were decided; a general plan of action for the development of the board was decided upon by the team which would offer general guidance on how the task would be carried out. A four-step plan was drawn up to help structure and plan out the process:

- Design a schematic for the design of the circuits which are on the power management board and takes into account the design decisions made.
- Source the correct components and troubleshooting any issues.
- Implement the circuit onto a breadboard or Vero-board to test the circuits and help the troubleshooting process before the final PCB is developed.
- Create a PCB version of the schematic designed earlier in the Altium Designer suite and populate the board with the components procured from Farnell.
- Testing of final PCB to proves that the management board is provides the right levels and can provide enough current.

A schematic for a circuit was designed by the team which could incorporate all the specifications and also provide solutions to various design decisions. **Figure 61** shows the initial schematic of the circuit diagram is below:

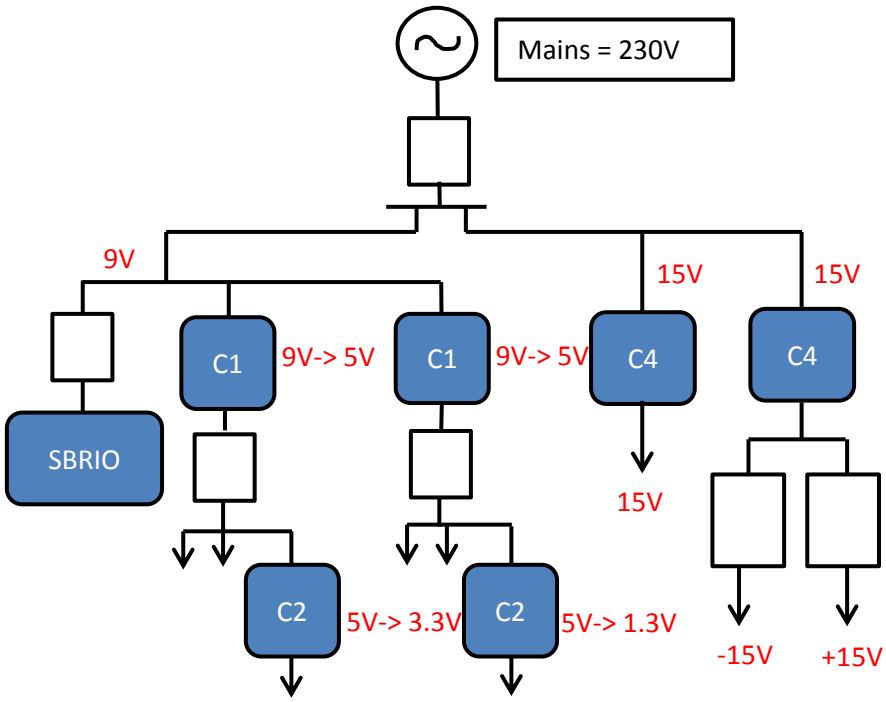


Figure 61: Schematic of the power management board circuit

The idea was to convert straight from 9V to 5V, 3.3V and 1.3V. However, this wasn't possible as most step-down transformers generally had an input of 5V. So these step-downs had to be done in stages. This issue was taken into account when the team decided on the design. The introduction of fuses; shown in **Figure 61** by the plain black unlabeled boxes, are there mainly because of safety to help isolate potentially harmful defects on the board. With the introduction of these fuses, if one part of the circuit fails, the rest of the circuit should still operate normally.

With the design of the schematic decided on paper it had to implement on Altium Designer so it could be imported into the PCB design suite. **Appendix C12: Schematic Diagram created in Altium Designer** shows how the team implemented the board design into the schematic document in Altium. In order to complete the schematic accurately, a schematic library had to be created by the team to populate the board with components that are

actually used. The only components that were already in libraries on Altium were the Headers used for the RIO input and the different voltage outputs.

Once the specifications were finalised the next step was to source the components needed to complete the circuit. A meeting was arranged between the power management board team and a member of the overall project team who had more experience building power circuits and PCB development to discuss a general plan for the board's development. The meeting established some key parameters for the board:

- Using surface mounted components instead of through-hole would be better as it eliminates the stray inductance which is prevalent in through-hole PCB's. Using surface mounting also allowed the team to use both sides of the boards and save space on the board to avoid any cross inductance between components.
- Advice on the $\pm$15V transformers was given; there would also be a need for a separate rectifier needed in conjunction with the transformer.

Many choices and decisions were made before coming to the final choice of components which were used in the power management board.. **Table 9, Table 10, Table 11** and **Table 12** shows the options considered and justifications for choices finally made.

9V →5V Converter Chip

Initially the set-down from 9V to 5V was going to be executed through the use of voltage dividers. This would be very inefficient due to the amount of power dissipation and may unnecessarily add heat to the system. The best method in order to carry out this step-down conversion was found to be using a chip. Many were found through the team's Farnell search but only 3 were considered; these are listed in **Table 9** below:

Component Name	Input voltage (V)	Input Current (A)	Output Voltage (V)
MAX756CSA	2.5	0.2	5
MAX1675EUA	2	0.3	5
LTC1174CS8-5	9	0.4	5

Table 9: 9V to 5V Converter Chip options

The inputs were far below the 9V which was required. The MAX series chips were given the correct outputs but the inputs showed they were in fact step-up transformer chips which weren't viable so only the LTC1174CS8-5 was suitable.

5V→3.3V Converter Chip

This transformer chip is needed in order to power the interfacing PCB. This level or voltage is a standard in electrical devices so finding suitable chips produced more populated results for this than for others. Possible chips which the power management team had considered are listed below in **Table 10**:

Component Name	Input voltage (V)	Input Current (A)	Output Voltage (V)
LT1303CS8-5	6	0.2	3.3
LM2852YMXA-1.2/NOPB	5	2	3.3

Table 10: 5V to 3.3V Converter Chip options

Due to using almost exclusively surface mounted components, the choices for these chip were limited, so the only suitable one found was the LM2852YMA-1.2. The LT1303 was briefly considered, but there would have been a reduction in performance from inputting a lower than recommended voltage.

5V→1.3V Converter Chip

The converter chip is used to power the Digital Signal Processor (DSP). Finding suitable converter chips at this level proved to be quite difficult as the standard near this level is 1.25V. The team's findings and choices were listed below in **Table 11**:

Component Name	Input voltage (V)	Input Current (A)	Output Voltage (V)
BD9102FVM-TR	5	0.8	1.24
MAX1692EUB+	5.5	0.6	1.275

Table 11: 5V to 1.3V Converter Chip options

The choice for the 1.3V initially was the BD9102FVM as it performed at the chosen input voltage. The performance of the DSP won't be negatively affected by the lower output voltage. DSP runs at 120 MHz at 1.3V input, but according to the datasheet it can run at a suitable speed at 1.05V input so 1.24V wouldn't be an issue. Upon ordering the part it was out of stock and would cause large delays in the process if the team was to wait for it to become available. So the MAX1692EUB+ was then chosen and can operate at the 1.3V without harming the chip due its tolerance allowances.

Mains → ±15V step-down transformer

The 15V transformer is use to power the custom DAQ PCB amplifier circuits and power transducers. A step-down transformer for the mains to 15V requires a rectifier to be purchased as well. **Table 12** displays the list of the transformers the team considered.

Component Name	Input voltage (V)	Input Current (A)	Output Voltage (V)
LM78L15ACMX/NOPB-IC	23	0.1	+15
MAX630CPA+-IC	16.5	0.15	+15
MYRRA Power Supply - 2 Common Outputs, +-15V	240V (MAINS)		± 15

Table 12: +15V Step-down Transformer

LM78L15ACMX wasn't considered anymore because of its input being too high. The MAX630CPA was suitable but at quite a high cost. Both also required separate rectified to be purchased. The MYRRA power supply was found as a cheaper alternative, as well as outputting +15V and -15V and has an in-built rectifier to save on costs. To provide both plus and minus voltage levels two of these were required.

After component section, pricing was discussed with the auditor to find out what the budget could stretch as the initial forecast would be exceeded; this limited the amount of components that the team could buy so it had to be bought and then tested with care.

As all the chips were surface mounted, it was desirable that the required resistors and capacitors were as well. The surface mounted components all have a general footprint of the 1206 package. The package was chosen because it had the most populated search results so the suitable surface mount components were found quickly. It was also large enough to make soldering the components with this footprint relatively simple.

Mains

The power management board essentially acts as a series of step-down transformers for the mains voltage (230V) connected to the board. Taking this into account, another specification had to be made for how the mains voltage would enter the board. A kettle lead connection was decided upon for this so the mains can be plug directly into it. The main factor behind this decision was safety; the kettle lead plug can easily be removed from the board if there is an issue the board to reduce the chances of complete system failure.

Testing

The chips needed to be tested when received to make sure that they were operating correctly. The testing procedure for these chips was to build the circuits on a Vero-board with through whole passive components. Through study of the datasheets for the chosen chips, the testing circuits were made. Testing these chips individually helped with the troubleshooting process as a simplified circuit is easier to analyse. The circuit is tested using a function generator as an input and measuring the output with a multi-meter.

The dimensions of the chips being tested were very small, too small to fit onto the board without being mounted onto a surface-mounted PCB of their own. Break outs PCBs were made for these PCBs so they could be tested, stencils and solder paste where used to attach the components in these test boards and the final design (**Appendix C13: Stencil of the final Power Management Board**). **Figure 62** shows one tested PCB when mounted. The header is added to give the surface mount chip a through-hole connector for testing.

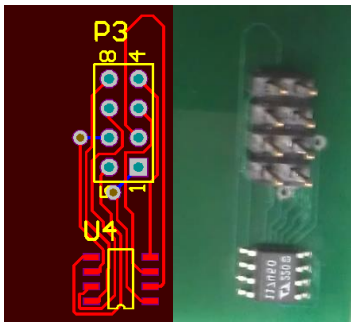


Figure 62: PCB schematic of a LTC1174CS8-5 test board and mounted version

The original idea was to take this final testing PCB and test these on a Vero-board. **Appendix C14: Connection of a Vero-board** shows the connection of one test Vero-board. Using the 4x2 header seen in **Figure 62**, the two pins short each other as they are in the same connection; an 8x1 header was much more suitable. Using an 8x1 also simplified the routing for the PCB chip as is shown in **Figure 63** with the final PCB mounted LTC1174CS8-5 chip.

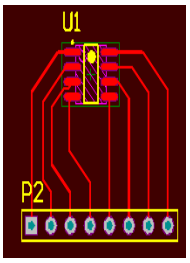


Figure 63: PCB layout of a LTC1174CS8-5

The next stage in the testing is to populate the Vero-board with the appropriate components around to complete the circuit, in accordance to what is written in the datasheets. An example of the circuit is shown in **Appendix C15**: **MAX1692EUB+ on a Vero-board**. It also shows the MAX1692EUB+ circuit.

Before the testing is initialised on the Vero-board; the circuits were simulated on a software package DIYLC. This allows a virtual representation of a Vero-board to be generated so the arrangement of the components and wires can be mapped. Using DIYLC helps increase the efficiency of the testing process as some errors can first be seen in the software and corrected before implementation on the real Vero-board.

Implementation

The next stage was to translate the schematic design from Altium design space, to a PCB board of a set size. The components from the schematic design were imported directly onto the PCB by using functions in Altium for that purpose. The PCB components for the Power management Board weren't imported in any logical order and the size of the board remained undefined. The team devised a series of actions to be completed for final placement:

- Select the correct board size depending on the specification
- Double-check that all connections between the components and chips are correct.
- The power supply and the headers for the other connecting boards are place in the correct positions.
- The components be arranged in way in which makes the routing of wires simpler.
- The mains connecting wires have to be kept away from the chips with small voltage tolerances.

The PCB board layout is important to the process; it showcases where all the components will connect with regards to their orientation and also shows the track widths needed in order to connect them. The track width determines what current can be carried through the routing wire and where calculated using Maxwell's equations as used in the custom DAQ PCB development. If this had not been taken into account it may have led to overheating of the tracks. Ensuring routing connections are all to the right components was also vital; if that had gone wrong it would have led to several circuits being shorted and malfunctioning.

The power management board is meant to fit in an enclosure with the other PCBs to make up the final embedded system. All the boards inside the enclosure were initially meant to have identical dimensions so they could be attached together with the screws and spacers in a sandwich formation. The orientation of the power supply depends on the position of the power inputs of the other boards the power management board is powering. It was envisioned that all the inputs that will need to be powered by the board will be on the same side for a logical and compact design.

With this goal in mind, changes had to be made to that design ideal due to the large dimensions of the ECE40 transformer. The power management board size needed to be altered as the component placement would have been too complex. A solution was proposed and implemented which would help retain the compact design originally decided, simplify the routing process and keeping the high voltage routing tracks away from the smaller voltages. The solution was to increase the size of the enclosure by the width of the transformer so the PCB could be extended to fit it.

A schematic library with all the components used on the power management board was required before routing. Each of the components in the schematic library needs to have a "Footprint Library". These footprints show the PCB's solder pad dimensions necessary for component placing. Components datasheets were used to create these.

The component arrangement of the PCB designed in the first iteration and second iteration are shown in **Figures 64 and 65**:

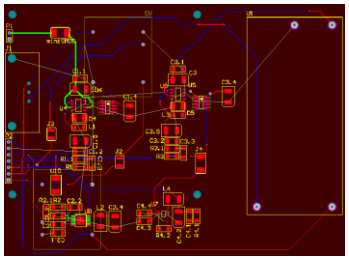


Figure 64: 1st iteration of PCB Design for the Power Management Board

Figure 64 shows the first design which was proposed for the power management board; the ECE40 transformer is given its own space on the right away from the rest of the components. The tracks which are wired to the mains are located on the edges of the board or on the bottom layer of the board. On the top layer of the board all of the passive components and small chips have been placed. The headers for the various voltage outputs and for the sbRIO have been strategically placed in line with the receiving terminal order to keep the design neat. Situated on the bottom layer were the ±15 transformers, this simplified the design and also meant that the higher voltage components are kept separate from the lower voltages ones.

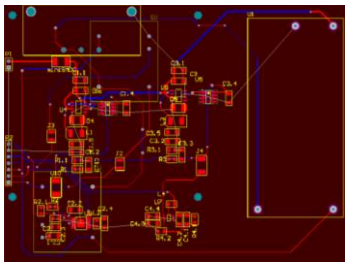


Figure 65: 2nd iteration of PCB Design for the Power Management Board

In the first iteration of the design, a fault was found in the footprint of some of the components in the Altium software including the power supply placement. The second iteration was necessary in order to correct these errors and to optimise the placement of the major components such as the power supply. The footprints errors in Altium were corrected and the power supply placed in a new location.

In these board designs, there aren't any ground connections made with the tracks. Connecting all the grounds to each other by routing the tracks was an extremely complicated process; due in most part to the amount of switching between layers needed to accommodate the routing. The solution for this was to add a poly pour layer. This layer is added to the top layer of the board and connects all the unconnected pins to ground automatically, simplifying the process. The setup of the board is in **Figure 66**:

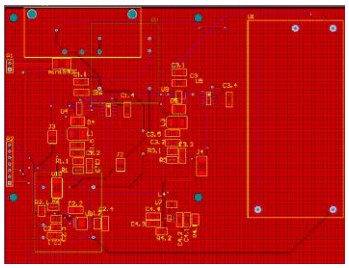


Figure 66: PCB Design of the Power Management Board with a Poly-pour layer added

3.3.6 Enclosure

The enclosure for the embedded solution was designed from measurements of the near complete embedded system using SolidWorks. It will provide a protective cover for all the PCB boards used in the embedded design when the boards have been combined together. This section discusses the tasks carried out in the design.

The first task was to take height, width and length measurements of the fully sandwiched PCB boards when all four PCB boards are screwed together. Not all the PCBs had been finished at this stage so the thickness of each PCB board added with the size of the biggest component there is on each board was used. The whole measured dimension from this method was then added again with the required spacing limit between each board and also the required spacing between the top and the bottom of the enclosure. The dimension for the length and the width of the enclosure are based on the power management board because it is the biggest board that will be placed inside the enclosure. **Table 13** shows the dimensions obtained either by taking measurements with a vernier caliper or reading through the components datasheet.

The dimension measured	The size of the measured dimension (in mm)
The total height of the custom DAQ board	22.04
The total height of the sbRIO	38.79
The total height of power management board	55.56
The spacing set between each board	4.32
The combined height of the three board including the set spacing limit	125.03
The width of power management board	135.525
The length of power management board	102.87

Table 13: Measured dimension of the length, width and height of the sandwich PCB board

From the table above, the size of the sandwiched PCB board will be 132.525 X 102.87 X 125.03 mm. The next task was to list all of the external connections on the enclosure as well as measuring their dimension. The external connections were only located on the front panel, top panel and the back panel of the enclosure. The size of these connection was needed to create the required entry holes. The list of the external connections and the size of it is shown in **Table 14**.

Name of external connection	The size of the connection (width x height in mm)
USB on the sbRIO	12.5 X 6.6
Ethernet on the sbRIO	16.13 X 13.84
LED on the sbRIO	11.43 X 6.36
Connectors on the custom DAQ	39.04 X 7.3
Power supply socket on power management board	30.25 X 27.8
Six LEDs holes on the top panel	5mm in diameter for one LED

Table 14: The measured size of the external connection

The location of the holes for all connectors needs to be measured in order for these connectors to be place properly on the enclosure. The location is determined by comparing the height of the PCB from the bottom of the enclosure to the position of the connectors on that PCB board. By doing this, the location of the connectors was measured.

The next task was to get the internal dimension of the enclosure including the location of the PCB mounting holes. The internal dimension size was set by adding a spacer limit to the width, height and length of the sandwiched PCB

boards. The chosen spacer limit was set at 4.32 mm. This value was then added to the dimension of the combined PCB board. From this, the total internal dimension of the enclosure is 141.165 X 107.19 X 129.35 mm.

The location of the PCB mounting hole was measured from the PCB design of the power management board. The holes required were measured directly using the measurement tool in Altium designer software. The measurement was used to determine the location of the PCB mounting holes.

After obtaining all require dimensions and measurements, the work on creating a 3D model of the enclosure could begin. After each part was finished, the final assembly was created by combining the parts. A view on the complete 3D model of the enclosure can be seen in the **Figure 67.**

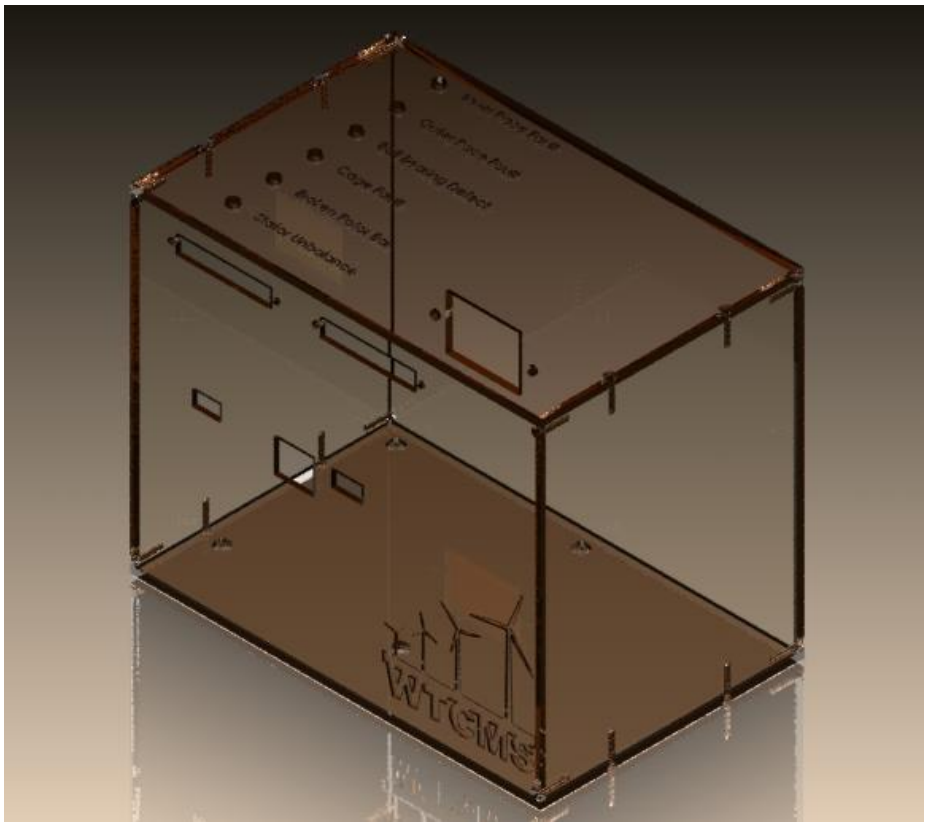


Figure 67: 3D model of enclosure

The 3D model file was then converted to a schematic drawing ready to be sent to the mechanical workshop.

3.4 Summary

Electronics have been designed to provide the interface between the sensors and sbRIO. A power management board has also been developed to address all of the power requirements for the embedded system. The design of these two discrete hardware components was completed with careful analysis of the original project requirements and intersystem requirements. Altium designer was used to create all of the required components and PCBs. Further to this, testing was conducted on both PCBs and the results either resulted in a slight modification or a PCB redesign.

Firmware for the DSP such as FIR, SPI and multiplexer programs where presented, highlighting their key features. A structured step by step explanation is given. The firmware was broken down into these separate functions to simplify integration and to encourage team collaboration. Each part was tested separately and documented, needed changes where then implemented.

The communications link between the sbRIO and custom DAQ PCB was chosen to be SPI. Most of the functionality for this has been tested ready for integration. Along with that communication link, another program to indicate whether a fault has occurred with the use of LEDs was made.

Finally an enclosure was designed in SolidWorks to house all sandwiched PCBs.

Overall Summary

1. Achievements

The interim report gave a single aim and ten objectives (**Appendix G2: Interim Report**). This section measures the final state of the project against these ten points and finally against the projects main aim. Extra comments show where objective have changed slightly and why.

1. Determine the type of signals to be measured and the sensors required.
2. Research fault analysis and the standard alarm protocol commonly used in fault detection.
3. Produce a layout of the condition monitoring system.
4. Design and build the required signal processing and power supply hardware.
5. Develop software to perform FFT on a moving time window.
6. Develop software that could track the fault frequency and raise an alarm when a fault is detected.
7. Design and build a GUI.
8. Design and build output signal circuit.
9. Final system integration.
10. Test the completed system on an experimental rig which is made available for the group to use.

1. Complete - Determining the types of sensors that may be desirable in a condition monitoring system was completed by the beginning of the second semester. For fault detection purposes these included current transducers, accelerometers and an encoder but further sensors were selected for auxiliary measurements. Voltage transducers and RTD sensors were selected to allow estimation of torque in the system and temperature, both identified as useful supporting information to the main fault detection techniques. All sensors have been tested and have proved to be of a high enough specification to accomplish the task they were purchased for.

2. Complete - Research into fault analysis has made comprehensive use of existing research and also practical tests on two rigs. The scope of the research into existing material has been broad covering different types of fault and different methods for detection. Research into alarm protocols in semester one showed expressed issues from the industry with current condition monitoring alarm systems. The information and data recorded in both of these pieces of research have aided the development of fault analysis and detection MATLAB scripts.

3. Complete - A layout of the condition monitoring system was produced though collaboration between hardware and software teams. This saw some changes during early development due to the limitations of hardware or software required for interconnections between components but the final layout has been finalised and has completed the first stages of testing successfully.

4. Partially Completed - One of the main highlights of the project has been the production of the custom DAQ and power management PCBs. Research into the requirements of each PCB made it necessary for team members to understand the connections between multiple subsystems and create a design around this. These have not been fully integrated yet but in initial testing on the individual components, no limitations have been discovered that would affect their ability to carry out their specified functions.

5. Complete - An understanding of what was desirable in a condition monitoring system caused this objective to expand. The software running on the embedded control has been designed to acquire and present a moving window of data to integrated MATLAB scripts but other task have also been built in; multiple logging functions,

system status reports, playback features, flexible settings and live data viewing have also been worked in and tested.

6. Complete - From the research in objective 2.), software to perform FFTs and track fault frequencies was developed alongside the prototype system. The scope was narrowed to a suitable level for a year's work by focusing the development on a select few faults. Alarm protocols for reporting an error to a user in other condition monitoring systems have been incorporated into the MATLAB detection script which includes a threshold system calibrated for the teams test rig.

7. Complete - The GUI and the code behind it allowing users to view and control the condition monitoring system was developed in the prototype system. With some changes to the code this has been successfully deployed to the embedded controller and can be plugged into it in the turbine, or across a network.

8. Dropped - The need for this objective was removed with the abilities designed into the GUI. However with the unused serial connectivity and digital lines, it could be included in future work.

9. To be completed – Though the majority of each of the subsystems have been tested to work, the final integration has not yet been completed.

10. Partially Completed – Two rigs have been used during development but not yet on a completed system. The system components that have been developed alongside the teams rig have been tested to work but are not yet fully integrated.

Though final system integration has not been completed to satisfy the projects main aim, most system components have been tested individually or with the prototype system. The prototype system was not explicitly given an objective of its own in the interim report but it has proved to be a logical step toward the final embedded system. It has aided the development and testing for every sub team and on its own could be of further use as a tool for understanding and developing extra functionality. It is anticipated that with a small amount of effort, the final embedded system will be completed for the demonstration day.

2. Future Work

The fault detection algorithm is applicable to electrical faults, bearing faults and gearbox failures. Gearbox failures are not included in the scope in the project but if they were to be included, the algorithm would be a replica of the one being implemented in the system at the moment. The necessary modifications would be the addition of sensors to the gearbox and the addition of the fault frequencies characteristic equations for gearbox failures to the fault detection algorithm.

Due to the near sudden occurrence of electrical faults, the developed detection algorithms using current analysis will not be able to detect or locate such faults before the generator gets turned off by automatic relay. However, if the voltage output of the generator is investigated, once the electrical fault happens and the relay switches everything off, the generator would keep spinning by inertia and producing a voltage. This provides a few seconds of data which is more than enough to detect the location of a fault. Attempts to improve the speed of data processing could also be made, in the current system this could involve better utilization of the FPGA by pipelining FFTs into it making advantage of the potential for parallel processing.

Because there are differences in the level of fault frequencies that appear even in healthy systems the ideal solution would include an auto-scan function. This could aid setup by running the first time that the CM system is placed in the wind turbine intended to monitor. This function could record the machine energy levels when

running in healthy operation at different speeds and use these measurements to automatically set up thresholds for all the faults being monitored.

From the review of other condition monitoring devices it was found that their sampling capabilities where superior to those possible in the current system. To become more competitive, an improvement in the ADC's sample rate and resolution could be made. The team has taken this into consideration during the ADC selection by picking from a family of ADCs that all share the same footprint. An exchange of ADC could be made easy. Due to a lack of time in the project, the migration of the DSP components from the evaluation board on to the custom DAQ hardware could not be made. This could reduce the size and overall cost of the custom DAQ board, important if it were ever to be produced in volume.

The filter implemented in the DSP can be changed to have different windows for better results. Other methods to implement FIR filters can also be coded that may give better filtration results. In the current state, a superior filtration method is redundant. However, with upgrades in other components of the system, the filtering method may also need to be upgraded to one better than the window method. In some situations, an IIR filter may perform better than FIR filter, which can also be implemented in the system. This could be done by making changes to the DSP code.

Further work on getting a LCD graphic display working with the system could be done to improve the its ability to display its status. The single board RIOs spare DIO lines could be worked on further to accomplish this with code to control them split between the FPGA and microprocessor.

With the final PCB developed, a substantial amount of the final work towards the final power management board has been completed. The work which still has to be carried out is the testing of the chips with function generator and population of the board using the microscope in the A2 Lab. These tasks are scheduled for completion before demonstration day. In the future; further development of the board could include using components with lower tolerances for more accurate readings at the various voltage outputs.

The embedded controller has many functions beyond its initial specification that have been identified as desirable for condition monitoring. However, some additional software features could be made so that the system can be considered more complete and user friendly. A new application that includes the current remote control and viewing functionality but also administrative tools to configure a new system with identifiers and initial settings would be required for a final polished product. A more sophisticated fault communication system could also be built in; the current design will keep the user informed of the most recent logs but does not keep a history.

References

[1] C. J. Crabtree, "Survey of Commercially Available Condition Monitoring Systems for Wind Turbines," Durham University, 2010.

[2] G. C. a. H. H. Z. Daneshi-Far, "Review of Failures and Condition Monitoring in Wind Turbine Generators," International Conference On Electrical Machines, 2010.

[3] K. A. a. W. Chen, “A Review of Electrical Winding Failures in Wind Turbine Generators,” Insulation Conference, 2011.

[4] S. S.Djurovic, “Origins of Stator Current Spectra in DFIGs with Winding Faults and Excitation Asymmetries,” IEEE International , 2009.

[5] S. D. S. D. A. S. D.S. Vilchis-Rodriguez, “Analysis of Wound Rotor Induction GeneratorTransient Vibration Signal under Stator Fault Conditions,” The University of Manchester, 2013.

[6] R. J. P. N. A. O. D. B.Mirafzal, “Interturn Fault Diagnosis in Induction Motors Using the Pendulous Oscillation Phenomenon,” IEEE TRANSACTIONS ON ENERGY CONVERSION, 2006.

[7] D. S. V.-R. S. D. Alexander C. Smith, “SENSITIVITY ASSESMENT OF WOUND ROTOR INDUCTION GENERATOR BEARING FAULT DETECTION USING MACHINE ELECTRICAL QUANTITIES,” The University of Manchester, 2013.

[8] N. A.-N. a. H. Toliyat, “A Novel Method for Modeling Dynamic Air-Gap Eccentricity in Synchronous Machines Based on Modified Winding Function Theory,” IEEE Transactions on Energy Conversion, 1998.

[9] Z. C. a. E. R. Q. Lu, “Model of stator inter-turn short circuit fault in doubly-fed induction generators for wind turbine,” Power Electronics Specialists Conference, 2004.

[10] V. D. a. F. Fuchs, “Detection of Rotor Turn-to-Turn Faults in Doubly-Fed Induction Generators in Wind Energy Plants by means of Observers,” European Conference on Power Electronics and Applications, 2009.

[11] W. L. a. C. K. MECHEFSKE, “Detection of Induction Motor Faults: A Comparison of Stator Current, Vibration and Acoustic Methods,” SAGE, 2005.

[12] N. Grid, “The Grid Code,” National Grid Electricity Transmission plc, 2013.

[13] M. A. P.J.Jover, “A General Scheme for Induction Motor Condition Monitoring,” SDEMPED, 2005.

[14] P. G. B. R. G. R. Martin Blodt, “Models for Bearing Damage Detection in Induction Motors Using Stator Current Monitoring,” vol. 55, pp. 1-2, 2008.

[15] T. G. R. G. Jason R.Stack, “Fault Classification and Fault Signal Production for Rolling Element Bearings in Electric Machines,” pp. 1-3, 2003.

[16] K. B. D. I. S. Izzet Y.Onnel, “Detection of outer raceway bearing defectsin small induction motors using stator current analysis,” vol. 30, pp. 3-4, 2005.

[17] P.Dudek, “Digital Signal Processing Course,” University of Manchester, 2011.

[18] LDS_Group, “Understanding FFT Windows,” SPX Company, 2003.

[19] D. Lyon, “The Discrete Fourier Transform, Part 4: Spectral Leakage,” Journal of Object Technology, 2009.

[20] I. P. WaveMetrics, “Hilbert Transform,” 2012. [Online]. Available:

http://www.wavemetrics.com/products/igorpro/dataanalysis/signalprocessing/hilberttransform.htm.

[21] P.-H. C. Deng-Fa Lin, "Fault Recognition of Wind Turbine Using EMD Analysis and FFT Classification," Third International Conference on Digital Manufacturing & Automation, 2012.

[22] Y. Li, "A discussion on using Empirical Mode Decomposition for incipient fault detection and diagnosis of the wind turbine gearbox," *World Non-Grid-Connected Wind Power and Energy Conference,* pp. pp.1-5, 2010.

[23] W. Yang, J. Jiang, P. Tavner and C. Crabtree, "Monitoring wind turbine condition by the approach of Empirical Mode Decomposition," *Electrical Machines and Systems,* pp. pp.736-740, 2008.

[24] A. C. C. B. a. W. L. G. Swiszcz, "A Data Acquisition Platform for the Development of a Wind Turbine Condition Monitoring System," 2008.

[25] Everything USB, "USB 3.0 Speed & Drive Benchmark," Everything USB, [Online]. Available: http://www.everythingusb.com/speed. [Accessed 2012].

[26] T. Romanko, "Extreme Design: Developing integrated circuits for -55 degC to +250 degC," EE Times, 2008. [Online]. Available: http://www.eetimes.com/design/automotive-design/4010319/Extreme-Design-Developing-integrated-circuits-for-55-degC-to-250-degC.

[27] P. Miller, "Aspects of data acquisition," Texas Instruments, 2005. [Online]. Available: http://www.ti.com/lit/an/slyt191/slyt191.pdf.

[28] H. Soni, "Proceedings of the 2009 International Conference on Signals, Systems and Automation," 2009. [Online]. Available: http://books.google.co.uk/books?id=dJ3EqEZslGQC&pg=PA291&lpg=PA291&dq=common+data+acquisition+system+block+diagram&source=bl&ots=0T_9rulO53&sig=s5CS3d3hnINuTGlEi1DxvqC9gks&hl=en&sa=X&ei=XaRlUcegAamY0QWdkoHwAw&redir_esc=y#v=onepage&q=common%20data%20acquis.

[29] R. B. Standler, Protection of Electronic Circuit from Overvoltages, John Wiley and Sons, Inc, 1989.

[30] E. P. Cunningham, Digital Filtering: An Introduction, Houghton Mifflin Company, 1992.

[31] DSPGuru, "FIR Filter Basics," Iowegian International, [Online]. Available: http://www.dspguru.com/dsp/faqs/fir/basics. [Accessed 2013].

[32] J. Patrick, "Serial Protocols Compared," 2002. [Online]. Available: http://www.embedded.com/design/connectivity/4023975/Serial-Protocols-Compared.

[33] N. Riedel, Electric Circuits, Pearson Prentice Hall, 2008.

[34] N. Dahnoun, "Chapter 14: Finite Impulse Response (FIR) Filter," Texas Instruments, 2004. [Online]. Available: http://www.ti.com/ww/cn/uprogram/share/ppt/c6000/Chapter14.ppt.

[35] DSPGuru, "FIR Filter Design," Iowegian International, [Online]. Available: http://www.dspguru.com/dsp/faqs/fir/design. [Accessed 2013].

[36] SKF, “SKF Condition Monitoring Products,” 2012. [Online]. Available: http://www.skf.com/group/products/condition-monitoring/index.html?alias=www.skf.comfcm.

[37] SKF, “SKF Integrated Condition Monitoring - Interactive brochure,” 2012. [Online]. Available: http://www.skf.com/binary/12-75560/Interactive-brochure-SKF-Integrated-Condition-Monitoring.pdf.

[38] B. &. Kjaer, “Brüel & Kjaer Vibro - Condition monitoring,” 2013. [Online]. Available: http://www.bkvibro.com/en/condition-monitoring.html.