Proof of Concept Data Logger for Non-active Power

Christine Martindale

Department of Electrical Engineering,
University of Cape Town, Cape Town,
South Africa

Submitted in complete fulfilment of the requirements for a Master of Science in Engineering in the Department of Electrical Engineering, University of Cape Town, Cape Town, South Africa

November 2013

Keywords: Load data logger, non-active power calculations, smart meter, generalised power theory, high speed data capture, power measurement, metering

Declaration

I declare that this dissertation is my own work and that apart from the normal guidance of my supervisor; I have received no assistance apart from that which has been stated.

Signature of the author: Signed by candidate

Christine Martindale

Department of Electrical Engineering,
University of Cape Town, Cape Town,
South Africa
November 2013

Abstract

This dissertation discusses a metering concept design that is based on equipment cheaper and smaller than a laptop which is able to meet the requirements of power measurements, such as those for non-active power, that need relatively high frequency, simultaneous sampling. An Analog Devices ADE7878 energy measurement IC is used for measurement of a three phase system instantaneous voltage and current. A STMicroelectronics STM32F4 ARM Cortex M4 is used for the digital signal processing. The software uses direct memory access and high speed data capture to allow enough time to perform the new general power theory proposed by Gaunt and Malengret [1]. The results are tested to ensure there is no data corruption and basic calibration is performed. The system has an upper limit of 2 000 Hz sampling frequency at which power calculations and instantaneous, simultaneously sampled voltage and current values can be output in binary format to an SD card without loss or corruption of data. This project considers only the ability of the system to accurately perform the power calculations.

Acknowledgements

I would like to thank the following people:

- My supervisor, Robyn Verrinder for her support and guidance.
- My co-supervisor Professor Gaunt for his support and guidance as well as for funding the project.
- Amir Patel for his assistance with the SD card library, as well as James Gowans and Amir for their advice on using the STM32.
- Chris Wozniak for his help setting up experiments in the power laboratory.
- Grant Stowe for meeting with me to discuss his power profiler.
- All those in the Power Systems Research Group and the Robotics Research Group for all the helpful discussions and comments.

I would like to acknowledge funding received from the University of Cape Town postgraduate funding award in the form of the SA College Croll Scholarship as well as for financial support from the THRIP grant TP2011072900054.

Contents

Declaration ii
Abstract iii
Acknowledgements iv
Contents v
Nomenclature viii
1 Introduction 1
1.1 Brief Background to Study 1
1.2 Problem Statement 1
1.3 Significance of Study 2
1.4 Scope and Limitations 2
1.5 Plan of Development 2
2 Literature Review 3
2.1 New Power Calculations 3
2.2 Existing Technology 7
2.2.1 Data Loggers 7
2.2.2 The Market 8
2.3 Smart Technology 9
2.3.1 Smart Meters 9
2.3.2 Customer Value of Smart Meters 10
2.3.3 Privacy Issues 12
2.3.4 Smart Grids 13
2.4 Research 15
2.4.1 Load Profiling 15
2.4.2 Frequency Measurement 16
2.4.3 Event Capture 16
2.5 Specific Technologies 17
2.5.1 Memory 17
2.5.2 Communications 18
2.5.3 Energy Measurement 19
2.6 Chapter Conclusion 19

3 System Development 20
3.1 Analysis of Profiler and the NI SCXI-1100 System 20
3.2 Specifications 20
3.3 System Requirements 22
3.3.1 Data Size Limitations due to Communication Capability 22
3.3.2 Non-Volatile Memory requirements 23
3.3.3 Volatile Memory Requirements 24
4 Design 25
4.1 Overall System Design 25
4.2 Measurement 27
4.2.1 Choice of Measurement IC 27
4.2.2 Circuit 28
4.3 Choice of Digital Signal Processor System 33
4.3.1 Processor System 33
4.3.2 Memory 35
4.3.3 Circuit 36
4.4 Software 36
4.4.1 HSDC 37
4.4.2 DMA 38
4.4.3 Calculations 38
4.4.4 Data Management 40
4.4.5 SD Library 41
4.4.6 Files used in the Program 42
4.4.7 C# Code 44
5 Experimental Procedure 45
5.1 Data Corruption Tests 47
5.2 Basic Calibration 50
5.3 Testing Accuracy of Actual Data 52
6 Results 53
6.1 Waveform Shape 53
6.2 Result Data 55
6.3 Graphs 58
6.4 Limit 61
7 Discussion 62
7.1 Limits on Frequency 62

7.2 Recalculation of required RAM 63
7.3 RAM 64
7.4 Important Requirements 65
7.4.1 Measurement Frequency 65
7.4.2 Speed of Memory 65
7.4.3 DMA 65
7.5 Limitations on Accuracy 65
8 Conclusions 67
9 Recommendations 68
List of references 69
Appendix 74
A. Programs Used 74
B. Microcontroller code 74
B.1 Full System Set Up 74
B.2 Read HSDC 105
B.3 Produce HSDC 109
C. C# Code 119

Nomenclature

DMA	Direct Memory Access
SD card	Secure Digital memory card
HSDC	High Speed Data Capture
SPI	Serial Peripheral Interface Bus
I2C	Inter-Integrated Circuit, multi-master, serial single-ended computer bus
CT	Current Transformer
VT	Voltage Transformer
HV	High Voltage
New Power Calculations, Non-active power calculations, generalised power calculations	The Calculations developed by [1], [2], [3]
Data logger	Unless otherwise specified this term refers to a data logger for power application purposes

1 Introduction

1.1 Brief Background to Study

There are various data loggers for power applications and electricity meters available; however most are costly and not easily modified for unique calculations. This feature is very useful when using a data logger for research which may require calculations or data which differ from the norm. Stowe produced a load data logger for the Power Systems Research Group in 2012 which calculated standard electrical power and RMS values for 3 voltage and 12 current channels, but had no additional functionality to measure simultaneous voltage and current [4].

Data loggers are of particular interest in the field of smart grids and meters where information about load profiles, faults and system losses are necessary for system control. The data logger is the device that monitors and records this information in the smart control systems.

Malengret and Gaunt (2012) proposed a novel general theory for power calculations that gives a measure of the true extent of power losses through an alternative method of calculating power, particularly non-active power. A data logger capable of measuring simultaneous instantaneously sampled voltage and current would be able to perform these calculations. This logger could then be used in applications where these power losses are of particular concern, for example under harmonic distortion, unbalance and dc components [3].

1.2 Problem Statement

To design a metering concept that is based on equipment cheaper and smaller than a laptop which is able to meet the requirements of power measurements, such as those for non-active power, that need relatively high frequency, simultaneous sampling.

1.3 Significance of Study

The data logger will provide a platform to further develop ideas and uses for smart technology, to further test and use the general power theory proposed by Malengret and Gaunt [2]. The proof of concept logger will allow a working portable prototype to be developed. This can then be used in research into different types of smart control and smart meters and their uses in a system or smart grid. This will ultimately lead to the development of more energy efficient solutions.

1.4 Scope and Limitations

The topic of a complete data logger system is covered in detail by Stowe's dissertation, 2012 [4]. Therefore the inclusion of the new power calculations into a system like this is considered here.

The data logger was developed within the South African context in terms of cost and range of frequency and voltage to be measured.

This report will only cover the proof of the ability to record and calculate non-active power, apparent power, real power and power factor. It will not address frequency measurement and will therefore assume a constant 50 Hz.

The alternative data logging systems that are used for comparison are the ones already in use by the Power Systems Research Group in 2013 and the one developed by Stowe [4], also within this research group.

The software is developed to demonstrate the proof of concept and so it does not have complex error checking, a real time clock or a user interface.

1.5 Plan of Development

Chapter 2 begins with looking at the literature available in the field of data loggers and smart meters then looks specifically at the required technologies. The following chapter analyses some specific loggers and their features. It then investigates the requirements for this proof of concept. Then the design of the logger is discussed considering separately the design of the measurement system, processing system and the software specifically. Chapter 5 outlines the experimental procedure and testing followed by the results of these procedures in Chapter 6. There will then be a discussion of these results and this dissertation's conclusion.

2 Literature Review

This literature review first gives an overview of the new power calculations, and then considers the different types of data loggers and which data loggers are commercially available. The use of data loggers in smart technology and energy research is investigated, as well as their impact and significance in those areas.

2.1 New Power Calculations

For comparison the conventional power calculations for a 3 phase 4 wire system are shown below.

Instantaneous power:

$$p = e_1 * i_1 + e_2 * i_2 + e_3 * i_3 \quad \text{(Eq. 2.1)}$$

Where p is the instantaneous real power, e_1, e_2, e_3 are the instantaneous wire voltages from the point of reference and i_1, i_2, i_3 are the instantaneous phase currents.

RMS voltage per phase:

$$E_1 = \sqrt{average({e_1}^2)} \quad \text{(Eq. 2.2)}$$

Where the above mentioned average is taken for values of e_1 over a whole cycle and $E_1$is the RMS voltage for phase one. This formula is repeated for all three phases.

RMS current per phase:

$$I_1 = \sqrt{average({i_1}^2)} \quad \text{(Eq. 2.3)}$$

Where the above mentioned average is taken for values of i_1 over a whole cycle and $I_1$is the RMS voltage for phase one. This formula is repeated for all three phases.

Total apparent power:

$$S = E_1 * I_1 + E_2 * I_2 + E_3 * I_3 \quad \text{(Eq. 2.4)}$$

Where S is the apparent power for the three phase system.

Reactive power:

$$Q = \sqrt{S^2 - P^2} \quad \text{(Eq. 2.5)}$$

Where Q is the reactive power of the whole three phase system and P is the average real power.

Power factor:

$$pf = P/S \quad \text{(Eq. 2.6)}$$

Where pf is the power factor of the whole three phase system [5].

A series of papers [1], [2], [3], [6], [7] presents a new way to calculate power. The motivation for this is that there are two causes of inefficiency in a power system: distortion power and reactive power [7]. Distortion power is from distortion and unbalance and reactive power is from inductive and capacitive reactance. In conventional power, Q is only related to reactive power which does not incorporate distortion power. Thus the term "non-active" power was introduced to include both losses. However, these could not be uniquely defined, so it is split as follows "*non-active power can be separated into the portion that can be compensated between wires without the need for energy storage and the component that requires energy storage*" [7]. These components are orthogonal to each other and can be represented in the power triangle as shown in **Figure 2.1**, below.

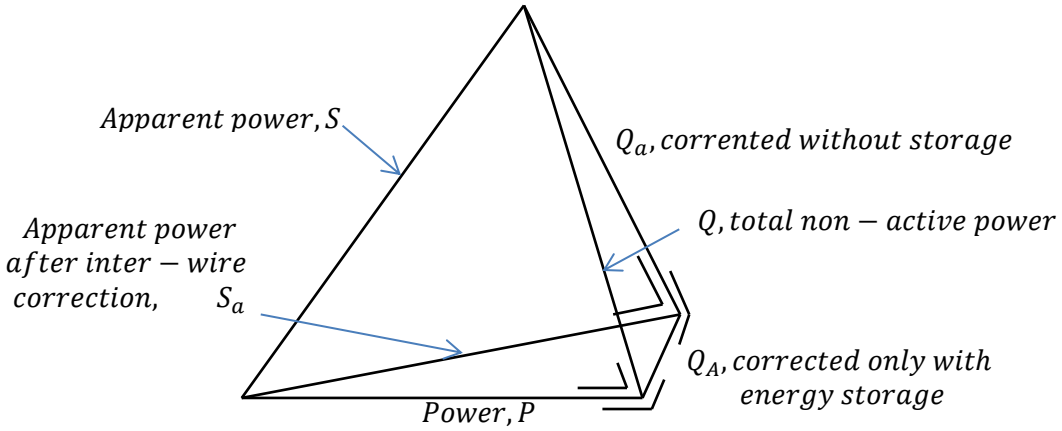


Figure 2.1. Complete power triangle, adapted from [1]

The new power calculations are based on the conventional ones, however they take into account the resistance of the wires, the neutral current and a reference voltage that is different from the conventional reference voltage. Resistance of the wires and neutral current will both affect the efficiency of a system, especially one under conditions of unbalance and distortion. These losses need to be calculated from a suitable reference. The new general power theory uses a reference point calculated from the instantaneous measurements taken throughout a cycle whereas the conventional approach summates a cycle from a fixed reference point.

The figure below shows a typical 3-phase 4-wire system.

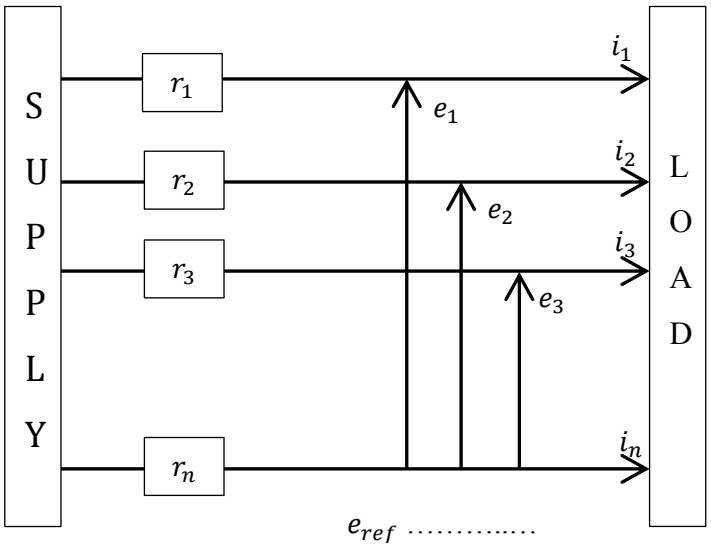


Figure 2.2. 3-phase 4-wire system, adapted from [7] **where e_1, e_2, e_3 are the instantaneous line to neutral voltages for the three phases respectively; i_1, i_2, i_3 are the instantaneous currents for the three phases respectively; e_{ref} is the instantaneous reference voltage and r_1, r_2, r_3, r_n are the line resistances and neutral resistance respectively.**

The two papers that use this diagram [1], [7] both use the simplification where $r = r_1 = r_2 = r_3$, however r_n may differ. Using this simplification, the generalised power theory calculations are as follows:

The neutral current, i_n , found using Kirchhoff's Law:

$$i_n = -(i_1 + i_2 + i_3) \qquad \text{(Eq. 2.7)}$$

The resistance-weighted norm of the currents, $\|\boldsymbol{i}'\|$:

$$\|\boldsymbol{i}'\|^2 = \left((i_1)^2 + (i_2)^2 + (i_3)^2 + \frac{r_n}{r} * (i_n)^2\right) * r) \qquad \text{(Eq. 2.8)}$$

The virtual, resistance-weighted reference point, e_{ref}, (also called the null point):

$$e_{ref} = \frac{(e_1 + e_2 + e_3)}{(3 + \left(\frac{r}{r_n}\right))} \qquad \text{(Eq. 2.9)}$$

The resistance-weighted norm of the voltages, $\|\boldsymbol{v_2}'\|$:

$$\|\boldsymbol{v_2}'\|^2 = \frac{(e_1 - e_{ref})^2 + (e_2 - e_{ref})^2 + (e_3 - e_{ref})^2 + (0 - e_{ref})^2 * \frac{r}{r_n}}{\boldsymbol{r}} \qquad \text{(Eq. 2.10)}$$

Apparent power, S :

$$S = \|\boldsymbol{V2}'\| * \|\boldsymbol{I}'\| \qquad \text{(Eq. 2.11)}$$

Where $\|\boldsymbol{V2}'\|$ is the average over a whole cycle of $\|\boldsymbol{v_2}'\|$ and $\|\boldsymbol{I}'\|$ is the average over a whole cycle of $\|\boldsymbol{i}'\|$**.** Though, as in conventional power, it can be the average over several cycles.

The instantaneous power P is the same in both general power theory and conventional power theory and so can be calculated either way. Using the virtual null reference is as follows:

$$p = (e_1 - e_{ref}) * i_1 + (e_2 - e_{ref}) * i_2 + (e_3 - e_{ref}) * i_3 + (0 - e_{ref}) * i_n \qquad \text{(Eq. 2.12)}$$

The power factor and non-active power are calculated as usual:

Non-active power (as opposed to reactive power):

$$Q = \sqrt{S^2 - P^2} \qquad \text{(Eq. 2.13)}$$

Where Q is the non-active power of the whole three phase system and P is the average real power.

Power factor:

$$pf = P/S \qquad \text{(Eq. 2.14)}$$

Where *pf* is the power factor of the whole three phase system.

Non-active power, real power, apparent power and power factor can all be calculated as instantaneous values or averages.

The impact of the new power calculations is that under conditions of unbalance, distortion and direct currents the conventional methods of power calculations underestimate the losses in the power system [7]. Being able to accurately assess the losses using non-active power calculations can create large savings. "*Metering can distinguish between the component of non-active power that can be corrected by inter-wire switching by power electronics and that which can only be corrected with energy storage components, giving practical interpretation to both components of non-active power*" [1]. Furthermore tariffs can be developed to discourage loads that reduce the efficiency of transmission [1].

The raw data needed for these calculations are instantaneous and simultaneously sampled voltage and current where all phases being used in the calculations are sampled simultaneously. The effective wire resistances will need to be determined. This information is not normally accessible from standard meters. A meter can be developed to perform these calculations using instantaneous, simultaneous readings of phases; however the determining of the effective resistance is not easily incorporated into a meter. The most basic meter to achieve these calculations is one that requires the user to input the phase wire resistance, that all the phases' resistances are equal and that the output is the result of the calculations and not the actual instantaneous values.

2.2 Existing Technology

2.2.1 Data Loggers

There are both data loggers and energy meters on the market. Energy meters are designed with billing purposes in mind. Data loggers are typically designed for quality testing, research and fault finding. The purposes of the two types overlap, so both are considered when looking at commercially available data loggers.

The most basic form of a data logger is a meter that records the accumulated usage of a utility in kWh; a basic accumulation meter [8]. The purpose of this meter is for billing. Another form of meter is an interval meter that measures energy usage over particular intervals. This type of meter becomes classed as a smart meter when remote communication capability is added [9]. There is no precise definition or standard for smart meters at present. Other features included in some smart meters are user interaction, user interfaces, ability to communicate and power on and off household appliances. This type of meter can help the client monitor and control their own usage and provide the utility with more detailed data and predictions [10].

There are various types of billing methods used by utility companies:

- Direct load control by smart meters – Customer credited for each time the meter allowed to remotely switch off an appliance. This is referred to as a home automation network (HAN).
- Basic accumulation meter – Records how much electricity has flowed through it. Consumption measured by subtracting latest reading from previous reading.
- Hourly pricing – Price announced a few seconds before the hour. No time to turn off loads.
- Block index pricing – Usual for commercial meters where there is a fixed price for a specified load. If the customers are over the limit, they pay extra at a higher rate and if under, they get a rebate.
- Critical peak pricing – User price increased at a critical peak, for instance when there is a fault. There must be a limit to the number of allowable critical peaks because there will be no warning or time to change load.
- Time of use pricing (TOU) either with a smart meter or a basic interval meter.[8], [11]

This information on energy usage is not the only useful information that can be produced on a meter. A data logger can record many other quantities, including current usage, real power and apparent power, power factor and signal quality.

2.2.2 The Market

Itron, an American metering company, develops meters for specific countries and their requirements. For example, the frequency and voltage levels differ between South Africa and the United States and therefore need different technology. Itron has been awarded a smart meter contract by the City of Johannesburg [12]. Itron, like many manufacturers, do not want their meters to become obsolete so develop and sell them modularly. It is therefore difficult to price and compare the meters on the market because, for example, the energy meter will be sold separate from the communications and power units. Each module can be updated and modified as required. An alternative to modular solutions is to bid for a large tender and build to different specifications for each contract and not sell generic modules. Thus there are very few generic meters on the market. The data loggers used for fault detection and quality measurement are easily available and are discussed in **Table 2.1**.

Table 2.1. A list of available data loggers, their price and what special features they have aside from the RMS calculations and conventional power calculations.

Name	Price in ZAR*	Comments
Fluke VR1710 single phase	9 380	No display but SD card memory, mainly for fault fixing
Fluke 1750B three phase	124 840	Quality recorder, no display but SD card memory, records waveforms at 5 MHz
Fluke 1735 three phase	30 420	Quality analyser, records data for 45 days, measures neutral and three phases
Fluke 1743 three phase	50 120	Power quality analyser, No display, records flicker, voltage events, THD, Data downloaded online
PITE 3561 three phase	15 440	40 days recording
Esis LDW 6092K three phase	15 080	SD card, hand held, no radio frequency transmission
ISO-tech IPM 3005 single phase	2 790	Flexible power quality tester, battery run, hand held
TRMS SYSTEM Multimeter	66 240	2 GB memory, 120 hours battery only, hand held
ISO-tech IPM 3600 three phase	10 560	Power Analyzer memory 500 kB, but no min/max measurements, measures neutral
Itron Centron Bridge meter	No price	Part of modular system, this is only the meter with no communications
Qualistar CA 8334B	31 060	Power quality analyser
Yokogawa WT1800 High Performance Power Analyser	467 400	SD or flash memory, USB communication, mains powered rather than battery operated, can measure instantaneous values and harmonics
NI SCXI-1100	16 520	32 channel ±10V analog input module; requires additional voltage and current transformers

* Prices as of July 2013, based on the exchange rate of 9.83 ZAR to the USD, rounded up to nearest 10 ZAR

When Stowe's data logger was originally designed in 2004, its purpose was to be a cheaper option and with more functionality than those commercially available [4]. The estimated cost in November 2012 was R3 550 [4].

Instruments capable of recording data for the new power calculations are the National Instruments (NI) SCXI-1100 analog input module, the Yokogawa WT1800 and the Fluke 1750. Due to the availability and cost the NI SCXI-1100 and the power profiler from Stowe's dissertation [4] will be compared for the purpose of the new power calculations in Chapter 3.

Bilik and Kaminsky [13] designed a flexible network analyser that is based on a NI CompactRIO connected to a PC rather than microprocessor based solution. The data logging and analysis application was programmed in NI Labview (National Instruments, 11500 N Mopac Expwy, Austin, TX) [13]. This allows for the software to be modified as power calculation standards change and for research purposes. This is useful for data logging in an environment where a PC per NI CompactRIO is available, however to take measurements in the field, it may be preferable to have some level of on-board pre-processing to reduce communication costs.

2.3 Smart Technology

Smart meters are a specific type of data logger. The specific requirements of smart meters, their value to the customers and the privacy issues that arise when using them are considered in this section of the literature review. A brief overview of smart grids, as the wider field that smart meters form a part of, is then given.

2.3.1 Smart Meters

Smart meters are being used in many countries to reduce energy consumption, among other things [9]. These meters are also beginning to be integrated into systems called smart grids.

Smart meters need:

- An interval meter
- Two-way communication
- A disconnect switch
- Real-time usage data in the residence or business accessible by an interface
- Optional time-of-use (TOU) billing
- Data available online daily [9]

McKinstry et al. [14] explain that the most basic need for smart meters is their ability to be read remotely, which means that meter readers are no longer needed. Further benefits of smart metering are that the utility can improve their billing methods ostensibly for better profits. The consumer and utility can develop and implement technologies to better use electricity by smart control of select household appliances. Nationally the need for smart meters is as a method of reducing unnecessary power consumption and thereby pollution [14].

2.3.2 Customer Value of Smart Meters

Smart meters for use by an individual or company can help the customer to consume power more efficiently, spread the load usage through the day and reduce billing costs. Utility companies can make use of smart meters for checking and ensuring the quality of supply, using the data for load research and for more profitable billing systems. It also takes away the need for meter readers. Smart meters in general can be used to reduce overall usage and peak demand by using intelligent load control systems, as well as balance the per phase load on a three phase system.

There is a negative customer image about smart meters because many do not understand what they are and how secure they are [15]. Prominent concerns about smart meters are privacy, health issues and hacking [11-13]. Hacking and privacy issues will be addressed in the next section. There is a concern that exposure to radio frequency waves may impact negatively on health, however the RF waves emitted by a smart meter are of the same level as Wi-Fi and cell phones as it uses similar technology.

The public utility commission of the state of California (PUC) decided to adopt a set of rules in order to increase and ensure the customer value. The following requirements for meters were used:

"***Web presentation***" – "*updated daily: detailed energy usage, bill-to-date, month-end bill forecast, and projected month-end energy price*"

"***Tier alerts***" – "*When customers move from one price tier to the next, the utilities are to provide notifications via a form of rapid communication*"

"***Real-time data***" – to control real time use

"***Third party data services***" – to have a third party analyse data for ways to increase efficiency and reduce cost

"***Rate option calculator***" – "*have the option of switching to a time-of-use rate*" [17]

An article by E-meter on smart meter policies around the world states that:

"*This will help consumers save money, use energy more efficiently, and reduce greenhouse gas emissions and air pollution"* [9].

Web presentation is mentioned above as a method for customers to control their energy usage. As an alternative, Weiss et al. [18] suggests a method of using smart phones as an easy access method to improve energy usage.

For successful implementation of smart meters there needs to be customer education and a sufficient understanding of what data needs to be collected as well as its usefulness. Victoria, in Australia, had the problem that smart meters were rolled out but had done insufficient research into how to pay off the installation costs and whether the installation costs would be covered by the ultimate energy saving [19], [20].

Another benefit of smart meters is the reduction of electricity theft. This source of savings has the potential to pay back the cost of the required smart meters. Brazil is an example of a country where smart meters mean that the sometimes dangerous job of meter reading is eliminated. Customers caught stealing electricity can no longer become violent towards the meter readers [21]. Non-technical losses (NTL), such as theft, affect the quality of supplied power, the load demand on the generator and the cost of energy to the genuine customer. In third world countries where NTL are high, smart metering is welcomed as it decreases the cost of electricity [22]. One method discussed by Depuru et al. [22] for preventing electricity theft is to calculate the NTL by using the data from the household smart meters and comparing this to a central source smart meter. If the NTL are greater than 5%, then the genuine customers would be turned off and the illegal consumers would be supplied with the output of a harmonic generator for a few seconds to destroy their appliances and then the legal customers' supply returned. This sounds like an effective method for discouraging illegal consumers but may provide too much unwanted disruption to the genuine customers as well as a potential fire hazard depending on the illegal load. It would be a large risk if there was genuine meter failure in a genuine customer's meter. Smart meters do not prevent electricity theft altogether. There are various ways in which a smart meter could be manipulated:

- By damaging it so that it stops giving readings
- Tampering with the current transformers (CTs)
- Tampering with the internal calibration
- Using the neutral line where this is not monitored by the meter [23]

All of the above, aside from damage to the meter, require technical knowledge, thus smart meters would at least reduce NTL.

The customer value of smart meters is seen as crucial by Kaufmann et al. [10] and thus the paper sets out to investigate the customer value and social acceptance of smart metering in select groups from Switzerland. This study shows a social acceptance and customer value of decreased energy use. However, it is limited in group selection and assumptions due to lack of demographic data.

McKinstry et al. [14] suggest there are two main benefits of a smart meter – the ability of customers to manage electricity consumption and bill by using smart control of non-vital appliances and choice of when to use energy based on tariffs.

2.3.3 Privacy Issues

Although smart meters sound like the best solution to reducing general power consumption, there exists a problem of whether the control of personal appliances and information made available by smart meters to the utilities will be socially acceptable. McKinstry et al. [14] suggest that many policies and regulations will be necessary to gain social acceptance of remote control of appliances. Several groups of countries, including the European Union, have begun putting together policies for smart meters, though none have as yet been completed.

The information from a smart meter and a smart grid can be made secure by basic encryption. This is sometimes not secure enough to be publicly acceptable due to the possibility of hacking. Another method is to make the data anonymous, such that the bill is attributed to a specific meter, but frequent, detailed information is scrambled and attributed to only a general area. This concept is explored by Efthymiou and Kalogridis [16]. The basic assumption in this paper is that billing data needs to be low frequency but attributed to a specific meter, however data used for distribution and generation control needs to be high frequency but can be anonymous. As explained by Khurana et al. [24], the complexity and scale of security management will be increased by orders of magnitude. There will be an increase in the number of intelligent devices in the system which will need device identification and security keys. All of this will need to be constantly managed and updated [24].

The detailed data a smart meter produces can reveal complex usage patterns such as eating and sleeping habits, numbers of people in the house and when the householders are out of the house. These data were generated and analysed by Molina-Markham et al. [25] and then a privacy enhancing architecture for the smart meter was formed to protect the customer's privacy

without jeopardising the usefulness of a smart meter. This involves a neighbourhood gateway and a "zero-knowledge protocol" where the utility receives the billing information, but not the actual power traces. This may have limitations as smart grids evolve to the point of needing more detailed information.

Aside from the knowledge that can be gained by data mining, there is a future privacy issue if smart meters are used to control household appliances such that non-critical appliances are turned off at peak times. This control of personal, household appliances by the smart meter may not be wanted by the customer and may be an intrusion of privacy. This could be overcome if the user can control the categories of critical and non-critical appliances so that if they need the washing to be done now, they can take the washing machine off the non-critical list so that the smart meter cannot turn it off.

2.3.4 Smart Grids

"*There is a growing need to reform the world's electrical grids, both from an aging infrastructure point of view and to address new environmental and societal challenges*" [16]. The European Union has a goal of reducing energy consumption by 20% by 2020 and to this end believes that smart meters and smart grids will contribute to this [10]. The EU is merely one example of the many nations aiming to reduce CO_2 emission and pollution by reducing energy consumption. The change to infrastructure required to implement smart grids will be costly. If a nation wants to implement smart grids it needs to incentivise or force the utility companies to allow smart grids and smart meters because in many cases it is too costly for them to consider it as profitable.

The two-way communication of customer usage and optimisation of planning, maintenance and operations brought about by smart grids will improve efficiency and reliability of electrical systems [24].
Specifically a smart grid can:

- Reduce inefficiencies in delivery and so lower generation requirements
- Reduce the number and length of power outages
- Integrate renewable energies
- Provide an improved method of system management [26]

If implemented correctly this should reduce the need for new generation capacity and make a more reliable power grid [26].

As mentioned above, power-outages can be minimised or eliminated by using the information made available by smart meters and produce a self-healing grid. This was shown experimentally by Samarakoon and Ekan [27]. However those experiments required synchronised smart meter data. Another use of the data shown by Samakaroon and Ekan [27], was its use in predicting load profiles. Smart meters and grids can be used to identify and isolate faults [28].

McKinstry et al. [14] propose the concept of a "*cell*" which "*possesses all the components parts of a national electrical network, albeit on a smaller scale*". It makes use of green energy generation and smart meters. Although each cell would ideally be self-sufficient, there would still need to be larger scale power transfer capabilities to allow for large cities and where green energy can be generated in relation to the need for that power.

The following countries are among those that have already begun issuing smart meters and so developing the potential for smart grids: Australia, Canada, USA, New Zealand, UK, Japan, Italy, Brazil, Switzerland [9].

Due to consumer backlash, some of these areas have allowed a disconnect switch to allow consumers to opt out of TOU billing. It varies across the countries whether the utility company pays for the meters or the customer, whether or not the government subsidises the meters, whether there is the functionality of home area networks (HAN) which allow for use of smart appliances. Due to the fact that very few smart appliances exist currently, some countries have decided this is not a necessary functionality.

The USA, UK and Europe have begun developing standards and policies for smart meters [29], [30]. Great Britain has compulsory requirements for the meter, which are in addition to the requirements in most countries:
"*Smart meters must enable prepayment billing. Also, every consumer must receive an in-home display (IHD) for real-time data — a device popular with many consumers (but not all)*" [9].

BBC news [31] stated that the UK will roll out 47 million smart meters by 2020. Their simple calculations show that it will take 6 to 12 years to pay back the cost of these meters. The widespread use of smart meters is obviously a step in the direction of reducing power consumption, however if it takes 12 years to pay back, there is a chance that the smart meters installed now will need to be updated before they have been paid back. This highlights the great need for low cost smart meters that can be modified rather than needing replacement.

2.4 Research

There are various areas of research that are linked to energy efficiency and smart grids in which data loggers play an important role. Some of the various uses of the data from a logger are discussed here.

2.4.1 Load Profiling

Smart grids and smart meters can be optimised if more information is known about the loads and how to optimise them. There have been some attempts to predict load profiles such as in the paper by Yao and Steemers [32]. These predictions were compared to statistical data and found to be fairly accurate. These predictions can be made more accurate if particular appliances could be singled out. Stowe [4] used initial current intake and the pattern of use to identify different appliances. Weiss et al. [18] also looked at identifying appliances and disaggregating them from the overall profile.

Switching transients are another method of analysing load profiles. Duarte et al. [33] show that using Continuous Wavelet Transform (CWT) analysis produces more accurate results than using short time Fourier Transforms (STFT). CWT is also less computationally expensive by an order of magnitude. The feature vectors computed are used to train Support Vector Machines, which can identify loads as they turn on or off.

Load profiles in literature can refer to two different profiles:

- The load profiles of specific appliances
- The load profile of the combination of all appliances that use that energy meter, that is the load profile of a building or household

There are two ways to automate home appliances for most efficient use of them: a home automation network (HAN) and non-intrusive appliance load monitoring (NIALM) [34]. HAN uses two way communications with various appliances and a control algorithm. NIALM controls the appliances from the breaker level but requires load profiling and identification of the separate loads.

There has been a fair amount of research into load profiling by disaggregating specific profiles from single point measurements. Some algorithms are accurate for specific appliances; however there seems to be no single algorithm that works for all appliances unless each specific appliance is tested and the control algorithm trained to recognise specific appliances as opposed to categories of appliances [35]. There are many methods for load monitoring, a more

mathematically based one is described in [34], although this again is not able to identify all types of loads. Some work has been done towards splitting appliances in groups of similar load signatures [36], though these are very broad categories and have a high percentage of false positives.

Load profiling on a large scale has been used for research where the previous day's smart meter readings are used in conjunction with a robust state estimator and some real-time data to successfully predict the voltages of a distribution network [37]. This research could lead to using smart meters in the control of distribution and generation on a large scale.

2.4.2 Frequency Measurement

Most smart meters measure the frequency of the supply. The knowledge of the frequency of a system is normally used to control the frequency. One smart meter controlling the frequency of one household's contribution to the national network is unlikely to have a significant impact on the supply's frequency. It would also be difficult to tell if the changed frequency was due to the load or the supply. Samarakoon et al. [38] found that *"direct load control through the smart meters is unlikely to be able to provide primary frequency response because of communication delays"* [38]. An alternative use of the frequency provided by smart meters to control the primary frequency response is investigated, it would need a data sampling rate of 200 ms [38].

A possible solution to support primary frequency response is to use load blocking. When an event occurs such that the frequency is changed, the smart meter is used to change the overall household load by blocking various categories of appliances. The conclusion of Samakaroon and Ekanayake [39] was: "*The scheme averted possible system wide load shedding by limiting the frequency drop before it reaches the load shedding-frequency. Though the frequency excursion is severe, the proposed smart meter control scheme is effective even in the case of future high wind scenario".*

2.4.3 Event Capture

There are two main events of interest that a smart meter could record:

- Power failure/fault
- Voltage dip/swell

Both of these events can be captured by frequent RMS readings or frequent instantaneous readings of voltage and current. This would require the meter to have a battery backup to give it enough energy to store the information about the failure and enough memory capacity to store the high frequency data about each event. Due to the fact that an event cannot, in general, be

predicted, the meter needs to be able to keep a running record of data before any possible event. This translates to having sufficient non-volatile memory.

Reid [28] uses information about events to perform automated switching to assist in isolating faults.

A voltage dip is defined as "*the decrease in RMS voltage level to 10%-90% of nominal*" [40]. Voltage dips are separated into three categories:

- Instantaneous (half a cycle to 30 cycles)
- Momentary (30 cycles to 3 seconds)
- Temporary (3 seconds to 1 minute) [40]

Half a cycle in this case is 10 ms. The power profiler only calculates the RMS every 200 ms because this is the standard for Class A instruments measuring parameter magnitudes [4]. Thus the profiler can only measure some momentary dips and all temporary dips. This could be modified in software, if required, because the hardware is capable of recording RMS values at shorter intervals. As explained in a paper by Naidoo and Pillay [41], there can be a lag of up to one cycle in the identification of the beginning of the dip. There are three other ways of detecting voltage dip: (i) peak voltage, (ii) Fourier and (iii) missing voltage technique [41]. Although the missing voltage technique seems to be the most accurate [42], the hardware on the profiler is best suited to RMS voltage dip detection.

2.5 Specific Technologies

This section considers the specific areas of importance when designing a new data logger given the above uses and concerns.

2.5.1 Memory

The amount of memory required to store 30 days' worth of data is calculated in section 3.3.2. This section will discuss the different types of memory available and how to store data.

Data can be stored in volatile or non-volatile memory. Volatile memory is lost when it is not powered whereas non-volatile memory retains its data. Volatile memory is much faster to access and therefore is used by a program for all the data it requires during operation. It uses non-volatile memory for long term storage of data.

Data flash is read and written using the concept of pages. A whole page is read from memory, placed in RAM, edited and then restored to non-volatile memory as an entire page. Memory used

in flash drives, hard drives and SD cards is stored using a complex file system. The system uses a file allocation table (FAT). There are many open source libraries for creating file systems as this can be a complex process. There is also another layer of complexity that can be incorporated to save memory, that is, compression of data before storage. Ringwelski et al. [43] investigated various lossless compression algorithms including off the shelf computer algorithms. They suggest using LZMH algorithm which provides the best trade-off between "*compression time, processing time and resources demands*" and level of compression. There is an advantage to not compressing data which is that if the memory storage device is an SD card then it can simply be removed and read by a computer from the file system without requiring an additional process.

Secure digital (SD) is a simple electrically erased and reprogrammed memory device using the SPI protocol. Secure digital (SD) cards are subject to the standards of the SD Association and have two methods to write to them – SD mode and SPI mode. SD mode is much faster, but is more complex. SD cards are very useful for their size and compatibility with other electronic devices, as well as the easily accessible SD card libraries.

When data are sent to and from a memory chip there needs to be some form of error checking, this is commonly CRC (cyclic redundancy check) and filter checksum. These are standardised error checking methods. If the meter was to be used commercially, its data exchange protocols would need to be standardised for example using:
DN EN 62056: Electricity metering – data exchange for meter reading, tariff and load control [44].

2.5.2 Communications

For short range communications Stowe's power profiler [4] uses Bluetooth, this is sufficient for its purposes. In smart meters it is more common to use Wi-Fi and in some cases Zigbee is preferred; Zigbee and Wi-Fi are popular because of their potential to communicate with smart household appliances. Kulatunga at al. [45] and Luan et al. [46] both designed smart meters using Zigbee communication, with standardised communication in mind.

Long range communications are often GSM, GPRS or 3G/EGDE. 3G and EDGE communications have a limitation of location. There are some areas that do not have good enough signal coverage to allow either form of communication. The most versatile communication is therefore GSM/GPRS. The GSM chip used in the Power profiler is GC864 which is now out of production. There are several similar chips like GL864, GE864 and G30 [47], however all of them have different pins and footprints and therefore the whole GSM daughterboard will need to be redesigned.

2.5.3 Energy Measurement

The energy measurement chip used in Stowe's power profiler [4] is the Analog Devices ADE7758 energy measurement IC. This chip calculates the voltage and current RMS values on three phases and the conventional power calculations – active, reactive and apparent. Although it has a 24-bit second-order sigma-delta (Σ-Δ) analogue-to-digital converter (ADC) per channel, there is only direct access to the value of one ADC at a time; therefore no instantaneous values can be accessed. Other chips commercially available are discussed in more detail in section 4.2.1

2.6 Chapter Conclusion

The usefulness of the data logger can be clearly seen by looking at the international interest in smart meters and smart grids and their benefits. The data logger will allow research in this area especially that of how to use the data available to create intelligent grid control systems. Most areas of a data logger design have been discussed by Stowe [4]. The area left outstanding is the ability to perform new power calculations.

The literature reviewed focuses on communicating with the supply and informing the customer. A smart meter would also benefit from local control and communication. The data logger developed for this dissertation will have the capability of local control and communication.

Load profiling is useful for predicting loads for use by control systems; however it is not useful for direct control of loads. The analysis and prediction of load profiles on a large scale for distribution control is computationally expensive and need not be done at each meter. Load profiling has been investigated by many; however with the development of HAN it is not particularly important to be able to separate out specific loads exactly, rather just be able to use the load profile as a whole. Event capture and the general power theory calculations are both important as inputs for control systems. Event capture will allow analysis of power quality and the general power theory will allow an understanding of the potential losses of a system. Frequency measurement is important, however would be better analysed with a more specialised instrument.

The data on a logger needs to be compatible with most existing computers, and so a large memory system with a file format is best. The most generic and suitable communications system for South Africa is GSM because of its large area coverage.

3 System Development

This chapter first analyses some existing data logging systems in relation to the requirements for the new power calculations. Then the specifications for a data logger which is able to perform the new power calculations are considered and their resulting system requirements.

3.1 Analysis of Profiler and the NI SCXI-1100 System

The requirement for instantaneous and simultaneously sampled voltage and current measurements as well as known wire resistances is the main difference between the conventional power calculation requirements and that of the new power calculations.

The National Instruments NI SCXI-1100 instrument is capable of these measurements; however it requires, in addition to the logger, a laptop, as well as current and voltage transformers. The data produced by the NI SCXI-1100 is calibrated with itself and the voltage input, however it is not calibrated for a 230 V three phase environment which requires current and voltage transformers. Thus it requires calibration which will be unique to the current and voltage transformers used. The code for the system is written in National Instruments Labview and has to be modified for each set up of the current and voltage transformers.

The power profiler developed by Stowe is a small low cost data logger; however it is unable to collect the instantaneous, simultaneously sampled values of voltage and current needed for the new power calculations.

Therefore this dissertation will look at an alternative system which is able to collect and manipulate the instantaneous, simultaneously sampled voltage and current values to produce the new power calculations.

3.2 Specifications

According to the IEC 61000-4-30, the international standard for testing and measurement techniques, there are two classes of measurement class A and class B. Class A is for precise measurement. Class A instruments require specific time intervals for calculations, namely

parameters are calculated over a 10 cycle interval and the aggregated time intervals are 3 s, 10 min and 2 h, the calculation of which is very specific. The 3 s aggregates are used to calculate the 10 min aggregate and the 10 min aggregates to calculate the 2 h interval. There are also requirements for the accuracy of class A instruments. Class B are specified for both time intervals and accuracy requirements [48]. Stowe's logger cannot be a class A instrument because it differs from the specified aggregated time intervals. However, the basic principles of accuracy required for class A are what is required for an accurate measurement instrument.

The parameters for this project are calculated over a two cycle interval. Due to the fact that they will be calculated by accumulating the values, it makes no real difference to the proof of concept whether the parameters are calculated over 2 or 10 cycles.

The data logger is being designed for a South African context. The expected nominal voltage is 230 V RMS AC, with a frequency of 50 Hz. The new power calculations have particular impact on unbalanced and distorted systems in which harmonics play an important role. As a result of personal communications with Professor Gaunt, on behalf of the GIC Research Group, the significant harmonics occur up to the 10^{th} harmonic. This is because after a certain point it is believed that the current transformer's resolution is not accurate enough. This means that the frequency that should be measured, using the Nyquist criteria, is at least 1 000 Hz (50 Hz * 10 * 2).

The frequency of the system, 50 Hz, is assumed to be constant. This value does fluctuate, however it is constrained by the standard NRS 048-2 to ± 2 % or ± 1 Hz and so for the purposes of a data logging system it is assumed constant [49]. However the level of fluctuation will need to be accounted for when performing accuracy calculations. For a class A instrument the frequency must be measured real time. Frequency measurement is not addressed here; however it could be included by using Fourier analysis on the pervious whole cycle, or by using a separate module.

The CT's need to be easily incorporated into the system to be measured, thus clip on CTs were used. When measuring on the secondary side of a transformer there are two CT ranges, 1 A and 5 A. The 5 A range is used in smaller substations and so the choice of current range was twice the current of interest. The system needs to be capable of measuring at least 10 A RMS.

The voltage must not fluctuate more than ± 10 % from the stated voltage for more than 10 consecutive minutes, with an absolute maximum deviation of ± 15 % according to NRS 048-2

[49]. If the system of interest is distorted or unbalanced then the resolution of the measurement is important. The standard set up within the Power Systems Research Group is that the voltage transformers drop the voltage to 110V three phase which is 63 V phase to neutral. This voltage has a variation of ± 5 % within which there must be as much information as possible, thus a resolution of at least 0.5 V per unit.

The new power calculations emphasize the neutral line and the current that flows there and hence the calculation of system losses. High voltage VTs always produce phase to neutral voltages so the secondary side is always a star arrangement with a neutral line. For the rare cases where the secondary system is a delta arrangement the new power calculations allow for the neutral resistance to be set to infinite. Thus the system can be used for both star and delta configurations.

The power calculations performed need to be real power, apparent power, non-active power and power factor. In order to calculate these, the instantaneous, simultaneously sampled voltage and current must be measured and the results accessible.

3.3 System Requirements

These are the calculations that were used as estimates for the choice of hardware.

3.3.1 Data Size Limitations due to Communication Capability

It is easy to collect more data in greater detail than is required, however the limitation comes in when that data needs to be stored. In this instance the limiting factor is the ability to transmit the data over long distances. As discussed in Chapter 2, if the size of the data to be transmitted is a few MB then GSM is a useful technology, however if the size of data per month is in the range of GB then in South Africa 3G technology is cheaper if data bundles can be bought. However there are many areas in South Africa that do not have 3G coverage. Both GSM and 3G take time to send and receive data and cost money to send and receive data. Therefore the data needs to be as small as possible while not losing useful information.

The data could be collected by a data logger, and then sent to a computer for processing. Given that the system will measure with a frequency of at least 1 000 Hz, 1 000 sets of data would need to be recorded and sent per second. However if the data are processed as they are collected, then the volume of data to be sent could be reduced. The standard intervals for information on power calculations from a class A instrument begin at 3 seconds (150 cycles) [48].

3.3.2 Non-Volatile Memory requirements

The method for calculating memory requirements was adopted from [4].

Assumptions for the purposes of a memory requirement estimate:

- Three phase data plus neutral
- Sampling rate 1 000 Hz
- Power values stored as doubles (8 bytes) because they will need to be floating point values
- Instantaneous values stored as signed 32-bit integers (4 bytes)

Table 3.1: Non-volatile memory requirements per second

	Number of channels	**Size in bytes**	**Quantity**	**Total in bytes**
flag	1	1	1	1
Time stamp	1	4	1	4
Instantaneous voltage	4	4	1 000	16 000
Instantaneous current	4	4	1 000	16 000
Voltage RMS	4	8	1	32
Current RMS	4	8	1	32
Conventional Active Power	3	8	1	24
Conventional Reactive Power	3	8	1	24
Conventional Apparent Power	3	8	1	24
New Active Power	3	8	1	24
New Non-Active Power	3	8	1	24
New Apparent Power	3	8	1	24
			TOTAL	32 213

So memory needed for 30 days is:

(60*60*24*30)*(32 213) / (1 024*1 024*1 024) = 77.76 GB

If the waveform data are only recorded for time spans classed as events and it is assumed that the events make up the equivalent of 1 day per 30 days then the memory needed is:

2.57 GB + 0.51 GB = 3.08 GB

Where 0.51 GB is the memory required for storing all the data in table 3.1 except for the instantaneous values, calculated as:

(60*60*24*30)*(213) / (1 024*1 024*1 024) = 77.76 GB
Thus there is a need for at least 4 GB memory on board.

3.3.3 Volatile Memory Requirements

To achieve the smaller amount of non-volatile memory by only recording the important calculation results and then recording the event information in more detail, there must be sufficient volatile memory or RAM to store the data while the program is running. There will also need to be sufficient RAM for all the temporary values and conversion factors used by the calculations. For aggregated values, there can be accumulation registers for the calculated instantaneous results. This means that the instantaneous values need not be stored. This does not require much volatile memory. The microprocessor should be fast enough for this.

For every 1 s of detailed data there needs to be 32 kB of RAM to use to store the data before it is transferred to non-volatile RAM. The RAM requirement will be affected by how the data are sent and received. Thus for an estimate there needs to be three seconds worth of data, 96 kB, plus code and buffer memory, so a minimum of 128 kB.

4 Design

This chapter discusses the choice of technologies and the design of hardware and software.

4.1 Overall System Design

The design of a system for non-active power calculations would need to include a power supply, communications and a user interface. However for the illustration of its ability to perform non-active power calculations the system only needs to include the ability to record, process and store instantaneous voltage and current measurements as well as the results of the power calculations.

This system requires:

- Conversion of analogue signals to digital data
- Record this data in volatile memory
- Manipulate / pre-process the data
- Store data

This is illustrated by **Figure 4.1** below.

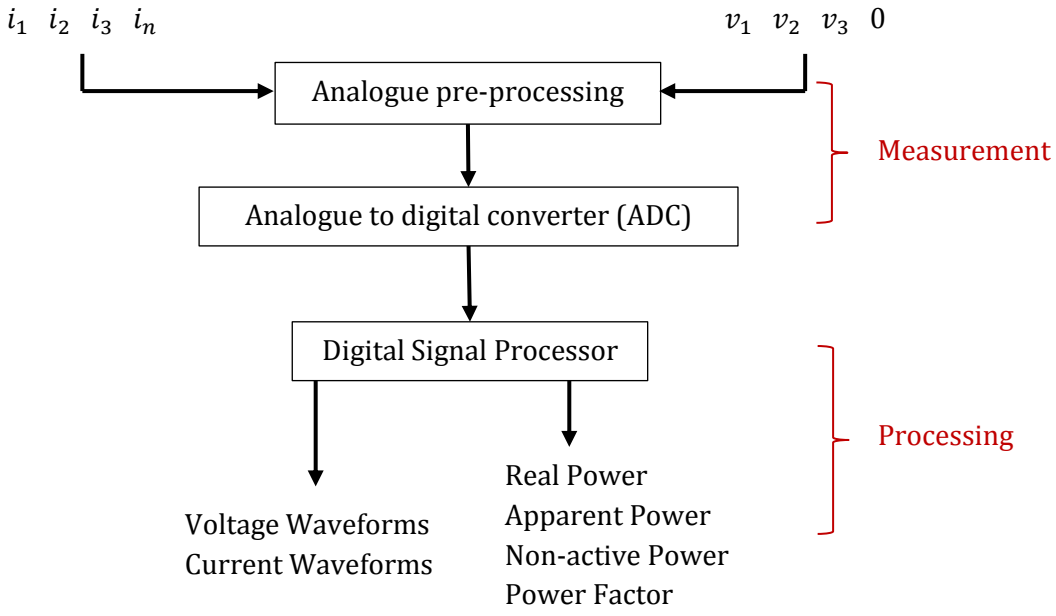


Figure 4.1 Block Diagram of System Design

The required specifications that were discussed in the previous section are summarised below for use in the design.

Table 4.1 System Specifications

Nominal voltage	230 V rms
Deviation from nominal voltage	± 15 %
Voltage resolution	0.5 V per unit
Current range	± 10 A
Frequency	50 Hz (assumed)
Highest harmonic to be sampled	10^{th}
Minimum sampling frequency	1 000 Hz
Phases to be measured	3 phases

The measurement and processing sections from **Figure 4.1** are discussed in more detail below.

Measurement

The current and voltage must be sampled simultaneously for current and voltage and across all phases. To convert analogue voltages to digital values the most obvious solution is an analogue to digital converter (ADC), however the input signals must first be pre-processed. The current and voltage must be stepped down and the current must be converted to a voltage. The voltage is stepped down using a resistor divider network. The current is stepped down using a current transformer (CT) followed by a resistor divider network in order to read the size of the current as a voltage value. The output of the ADC needs to be recorded such that the voltage and current of all phases of interest are sampled simultaneously to satisfy the requirements of the general power theory calculations. This circuit then needs to be isolated from the processing side for device protection to allow real-time USB access to configure the device and when running live for data collection and debugging.

Processing

These data need to be used in calculations and stored real-time. This requires that the processor is fast enough to record, manipulate and store the data before the next set arrives. If the sampling speed is 1 000 Hz, this leaves only 1 ms for all these calculations. There are various buffering techniques that can be used to allow more time for the calculations but this is only useful if bulk calculations are quicker than single calculations on individual samples.

The approach adopted in this project was to test the suitability of the various sub-sections of the overall system. This was achieved by using existing development boards where possible.

4.2 Measurement

4.2.1 Choice of Measurement IC

For instantaneous, simultaneously sampled values there must be an ADC per channel with a sample and hold register. These ADCs must be synchronised so that each channel is sampled simultaneously and each sample and hold register must be read before the next sample is recorded. This is most easily achieved using an IC which contains an ADC with multi-channel sequential sampling ability. There are various types of commercial ICs that perform conventional power calculations, some of which are shown below.

Table 4.2 Measurement IC Comparison

Company	**IC**	**Price in USD (July 2013)**	**Data Resolution**	**Instantaneous Sampling**
Microchip	MCP3903	2.11	24-bit	sequential
Texas Instruments	MSP430F677x	6.91	10-bit	simultaneous
STElectronics	STPMC1	1.26	16-bit	sequential
Analog Devices	ADE78xx	6.50	24-bit	simultaneous

The MSP430 and the ADE78 series are similarly priced and are both capable of simultaneous sampling of all channels. However, they differ in data resolution.

The resolution required to meet the specification of 0.5 V per unit is 24-bit as shown by the calculations below.

Voltage range: $230 * \sqrt{2} * 2 \simeq 650$ V

10-bit resolution: $650/2^{10} = 0.634\ V\ per\ unit$ (MSP430)

24-bit resolution: $650/2^{24} = 4 * 10^{-5}\ V\ per\ unit$ (ADE78 series)

The ADE7878 has a greater accuracy and fits the required resolution. The ADE7878 is also capable of transmitting the converted data using a high speed data capture (HSDC) protocol that is faster than standard SPI due to its lack of "handshaking". This port allows all 8 ADC values to be sent sequentially at a rate fast enough to achieve a capture speed of 8 kHz [50]. This protocol is further explained later in this chapter. Although the ADC values are sent sequentially they were recorded simultaneously.

The evaluation board with an ADE7878 measurement IC was used as it includes the recommended set up and isolation circuit which will be described below.

4.2.2 Circuit

The block diagram of the evaluation board is shown below.

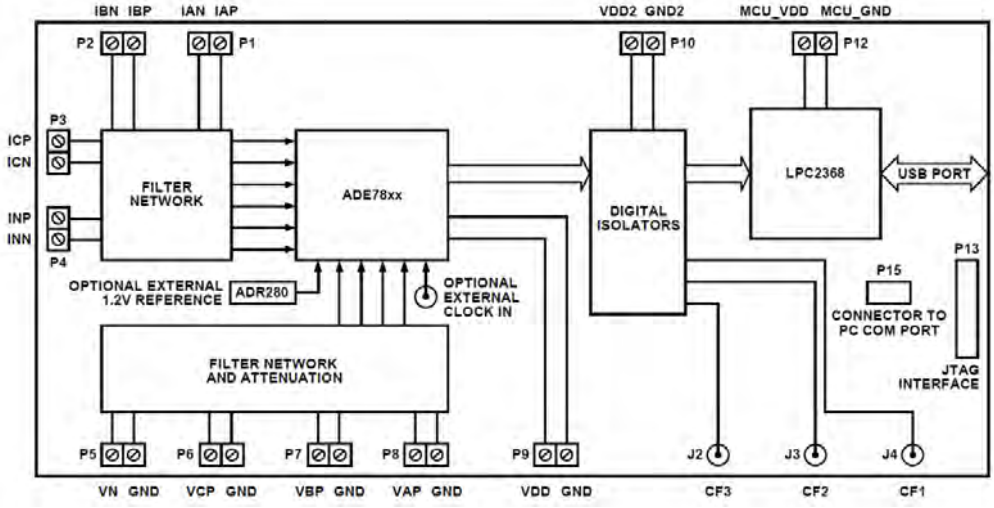


Figure 4.2 ADE7878 Evaluation Board Block Diagram. Taken from [51]

The external clock, external voltage reference, CF1, CF2, CF3, JTAG, COM port, USB port and LPC are not used and the LPC microcontroller is disconnected allowing for an external microcontroller to be attached via P12. The LPC microcontroller is not used because the EVAL board does not allow for reprogramming of this microcontroller and so the new power calculations cannot be added. The digital isolators are there to protect the microcontroller side of the circuit by isolating it from the live side. The current transformers are connected to P1-P4 and the voltage channels are connected to P5-P8. The filter and attenuation networks are shown in greater detail in **Figure 4.3** and **Figure 4.4**.

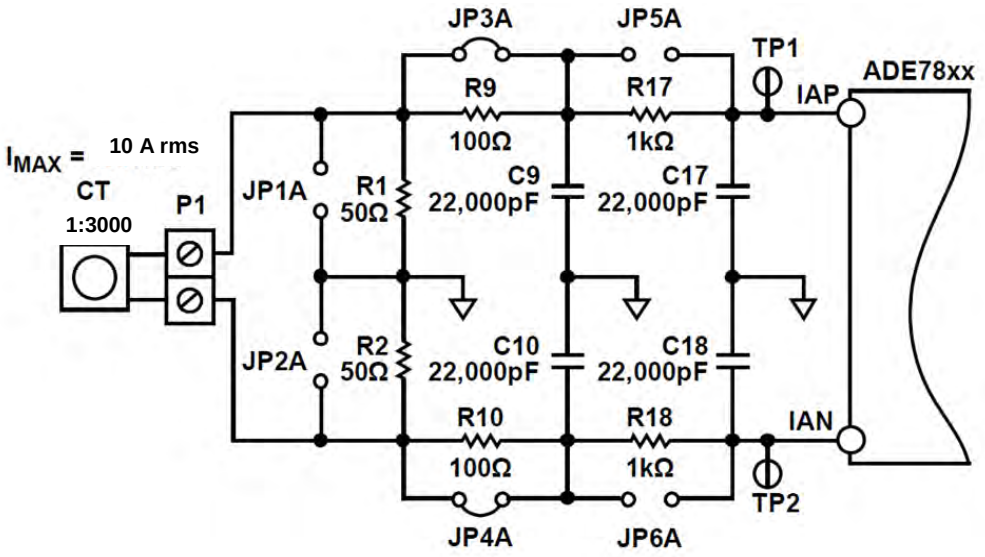


Figure 4.3 Current Transformer Connection. Adapted from [51]

In **Figure 4.3** the resistors R1 and R2 are burden resistors and were therefore calculated as follows:

$$R1 = R2 = {}^{1}/_{2} * {}^{0.5}/_{\sqrt{2}} * {}^{N}/_{I_{fs}} \qquad \text{(Eq. 4.1)}$$

where N is the input-to-output ratio of the current transformer and I_{fs} is the maximum RMS current to be measured. This formula comes about because the maximum voltage for the input pins of the ADE7878 is ± 0.5 V.

The current transformers used were Taehwatrans TS10L clamp-on split-core because of their easy installation (clamp-on design), their high turns ratio and their availability. The TS10L has a turns ratio of N = 1:3000. The designed maximum current was chosen as 10 A. So R1 = R2 = 47 Ω. Thus the actual maximum current allowed in the system is 11.28 A RMS. Given this maximum current and a resolution of 24-bits, the resolution is 1.9 μA per unit.

The anti-aliasing filters, R17/C17 and R18/C18, have a corner frequency of 7.2 kHz (1 kΩ / 22 nF) [51]. The components R9, C9, R10 and C10 are for use with Rogowski coils so are disabled.

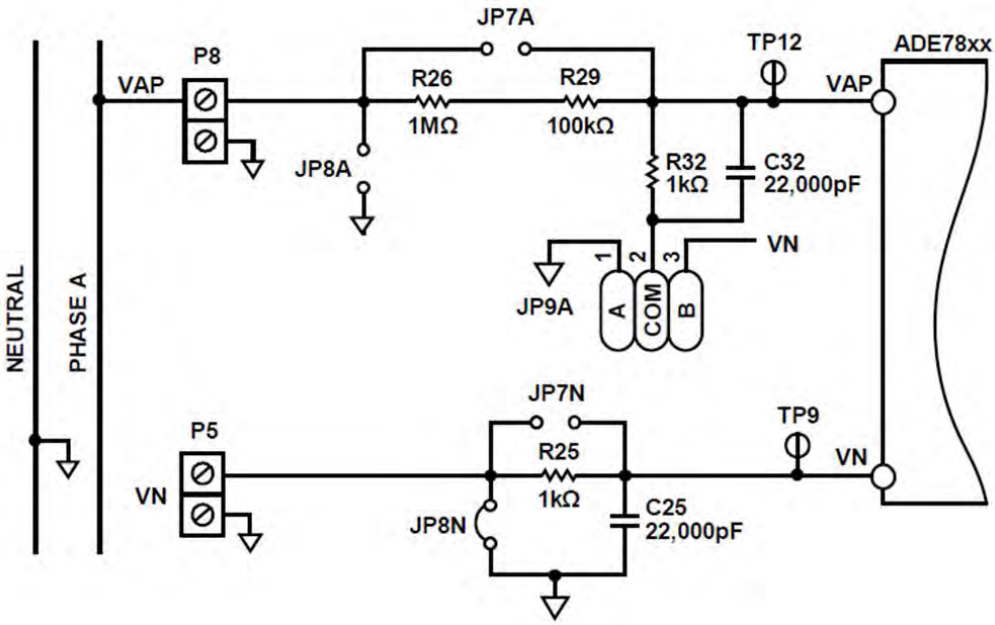


Figure 4.4 Voltage Connection Diagram. Taken from [51]

Figure 4.4 shows how the line voltage is stepped down using a resistor divider network. The particular choice of resistors and capacitors is to ensure the same corner frequency as the anti-aliasing filters to prevent large errors at low power factors [51]. The maximum pin input for the ADE7878 is ± 0.5 V, i.e. peak to peak, so the maximum voltage is 389.3 V rms and the resolution is $6.5 * 10^{-5}$ V per unit.

Thus the overall circuit is shown in a simplified version over the page in **Figure 4.5**.

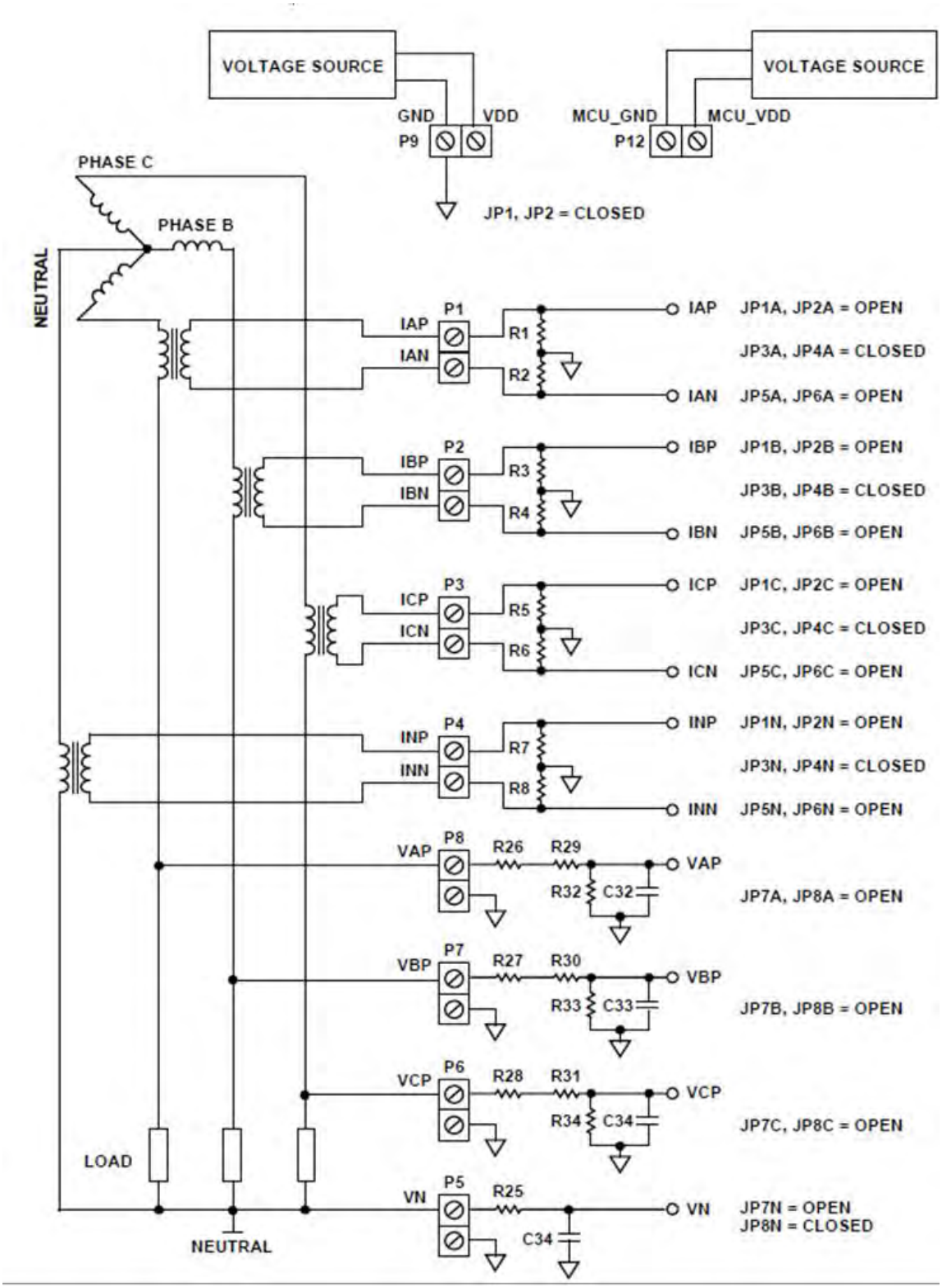


Figure 4.5 Experimental set up of the EVAL-ADE7878. Taken from [51]

Figure 4.5 shows the voltage sources where connector P9 is the voltage source for the source side of the circuit and has earth as the ground reference, whereas connector P12 is a floating voltage ensuring the two sides remain optically isolated. Therefore all the voltages measured by the microcontroller and converted into binary values are relative to earth.

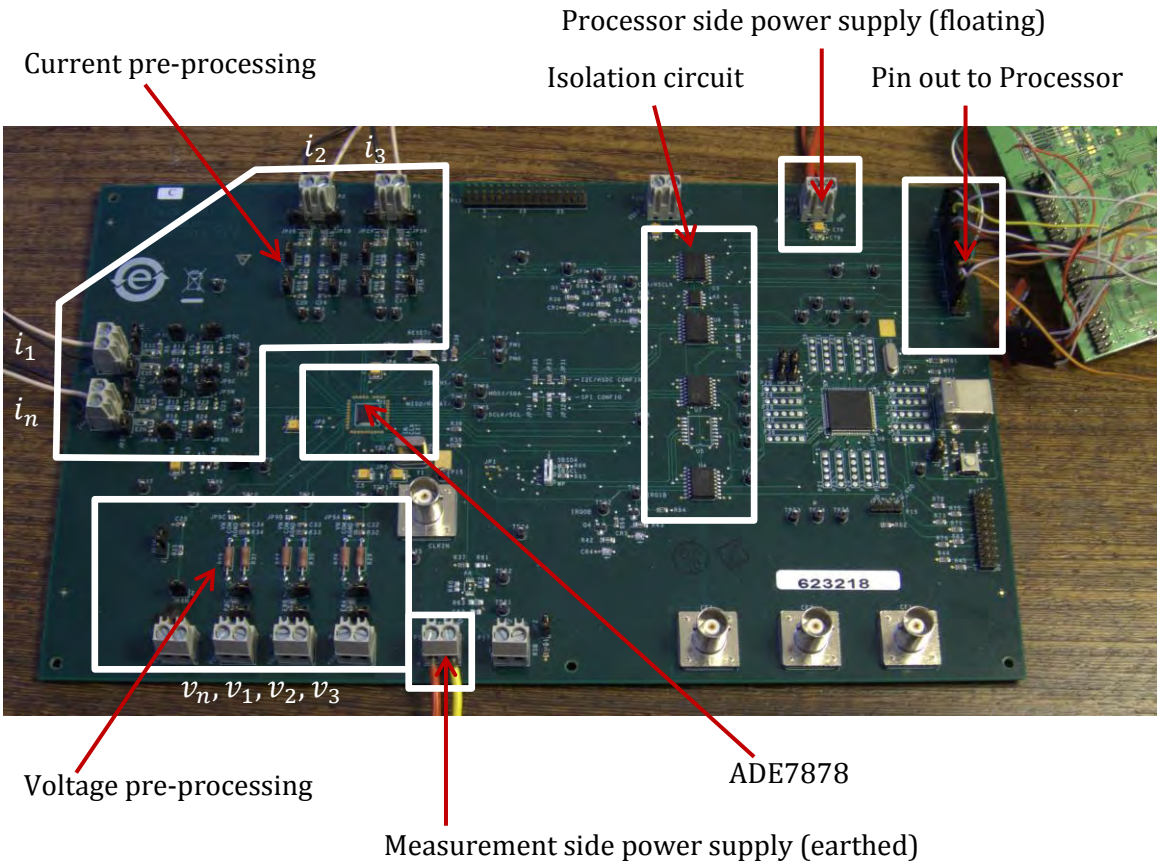


Figure 4.6: Photograph of ADE7878-EVAL Evaluation Board

The above picture shows the evaluation board and where each of the relevant sections is. The ADE7878-EVAL has one isolation circuit IC removed because it was faulty and was not needed for this project. The v_n voltage pre-processing part of the circuit also has a chip missing; it was not needed in this experimental setup.

4.3 Choice of Digital Signal Processor System

4.3.1 Processor System

The three main processors that were considered are the Raspberry Pi, the STMicroelectronics STM32F407 and the Microchip PIC32MX. The table below shows the specifications that were necessary for the microcontroller.

Table 4.3 Digital Signal Processor Comparison

	Raspberry Pi, Model B	**STM32F407VG**	**PIC32MX795F512L**
Flash	-*	1 MB	512 KB
SRAM	512 MB	196KB (64KB core-coupled memory)	128KB
Core	700 MHz ARM11-processors	ARM 32-bit Cortex™-M4 CPU up to 168 MHz 210 DMIPS/1.25 DMIPS/MHz	80 MHz, 1.56 DMIPS/MHz, 32-bit MIPS M4K® Core
I2C and SPI capability	yes	yes	yes
SD capability	yes	SDIO interface	No specific SD interface
DMA	yes	yes	yes
Price for microcontroller (USD, July 2013)	-	6.80	9.79
Price for development board (USD, July 2013)	35	14.25 (Discovery)	129.99 (Explorer 16) 72.00 (Expansion Board)

* Raspberry Pi has SD card slots so the flash can be as big an SD card as the user buys

The flash memory shown in the above table is very low given that the requirement is 4 GB. Thus the microcontroller needs to be able to have external flash, for example SD cards. All three are capable of this extension but vary in ease of use. The Raspberry Pi is the simplest as its SD capability is inbuilt and has very good libraries. An SD card break out board is required for both the STM and PIC, but the STM has a specified port and better libraries. The SRAM is all above the minimum of 128 kB, however the larger the RAM the more the program can be expanded to do. The speed of the core will affect how quickly the calculations can be done, a vital consideration in the case of real time processing for a system running at a minimum of 1 kHz. Another requirement for the core is its capability of easy handling of data stored as floats and doubles, thus a 32-bit processor is useful. The Raspberry Pi is by far the fastest, however perhaps unnecessarily fast, though this is largely unknown until the program has been written. A method that reduces the speed requirement of the core is the use of DMA. DMA is direct memory access

which means that data can be stored without the use of the processor. SPI and I2C capability are necessary given the choice of measurement chip, ADE7878.

The Raspberry Pi is a single board computer capable of running an operating system like Linux. The advantage of an OS is that tasks can be allocated using threading which allows parallel processing. The downside is that there is no way to know when the operating system is going to allocate time slices to a specific thread so the exact sequence of events is unknown which can be problematic in real time processing. A further advantage is the simplicity of using an OS and its ability to be expanded to run long distance communication. A down side to this arrangement is the lack of low level data sheets and specifications for the data access and storage. This may be critical in a system that is collecting and processing data real time.

Both the PIC32 and the STM32 meet the criteria of low level access and control of data, communication and DMA. However, the STM has more RAM and flash than the PIC as well as far better support and libraries.

Considering the cost aspect, the Raspberry Pi is cheap for what it offers however it has many parts that are irrelevant to this project and given the use of an OS, difficult to know exactly how the program is running. The STM is cheaper than the other choices and meets the requirements. Also the expansion board is a fair price comparatively to both the PIC Expansion Board and the Raspberry Pi. Therefore the STM32F4 was selected and used. The STM32F4-DISCOVERY is shown below.

Figure 4.7 Photograph of STM32F4-DIS. Taken from [52]

4.3.2 Memory

Due to the choice of STM32 there will need to be added flash memory. An SD card is useful because the data can be easily transferred to a computer for analysis and testing of results. Other forms of additional non-volatile memory are flash ICs and flash drives. Flash drives require USB which is large and time consuming and flash ICs need another form of communication to access the data for analysis and testing.

For this project, several 2 GB SD cards can be used. An SD card bigger than 2 GB is actually of a slightly different class and standard called SDHC or SDXC, although they are the same physical size they have differing file systems. The SD standard cards use the FAT 12 or 16 file system. [53]. SD cards are chosen for their compatibility with other electronic devices, their file system, their compact size and the ability to use SPI protocol.

In order to connect the SD card to the STM, a Sparkfun Break out board was used. The breakout board is shown below. The micro SD was chosen for its smaller size.

Figure 4.8 MicroSD Break out board. Taken from [54]

Figure 4.9 below shows the connection diagram when using SPI instead of SDIO protocol to write to the SD card.

SD Break Out		**SPI Device (STM)**
(chip select)	**CS**	**GPIO** (output pin)
(clock)	**SCL**	**SCLK** (clock)
(data out)	**DO**	**MISO** (master in, slave out)
(data in)	**DI**	**MOSI** (master out, slave in)
(chip detect)	**CD**	**GPIO** (input pin)

Figure 4.9 SD Card Connection Diagram

The chip detect pin is not connected in this project because the chip detect function is done in software. The function in question checks to see if it is able to mount the disk. If an error of disk not mounted occurs it could either be that the SD card is not present or that the card is corrupted. So the chip detect pin could be used to improve the error messages.

The SD card used was a Verbatim microSDHC 8 GB class 10.

4.3.3 Circuit

Figure 4.10 shows the connection diagram for the measurement board, the microcontroller and the SD break out board.

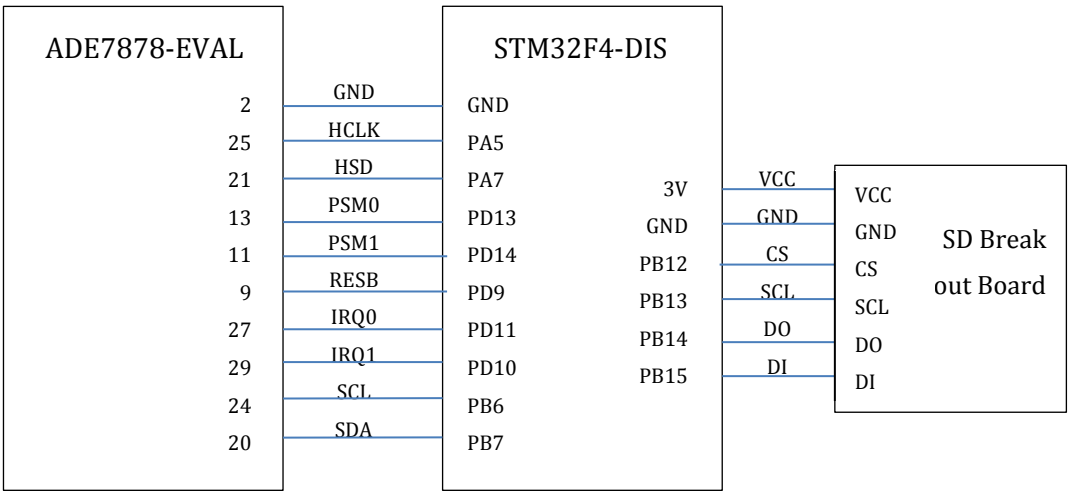


Figure 4.10 EVAL-STM-SD Connection Diagram

4.4 Software

This section is set up in order of how it was designed. The data needs to be collected at a fast rate so the high speed data capture (HSDC) protocol was used. Then it needs to be stored in buffers while leaving enough processing power for calculations, this was done using direct memory access (DMA). The data must be used in various calculations. The data needs to be carefully managed as each protocol uses the data in a different form and there is limited space for buffers used in the calculations. Once the calculations are finished, the data must be stored. Finally the required initialisation code and flow chart are looked at. The stored data will also need to be accessed and so a basic program for use on a laptop is designed.

Please refer to Appendix A for the software used for programming and debugging this system, Appendix B for the microcontroller code and Appendix C for the C# code.

4.4.1 HSDC

The ADE7878 is set up so that the recorded data are stored in internal registers on board the chip. These registers can then be read individually using SPI or I2C. A quicker method of collecting the data is using a feature called high speed data capture (HSDC). The reason it is faster is that normal SPI has "handshaking" protocols between the sending of the contents of each new register whereas HSDC assumes that the receiving device is ready and waiting and performs no "handshaking". This means that there is no inherent error checking as with normal SPI, so other approaches and the way the software is set up must be used to ensure no data are lost. This also means that the communication time is less. HSDC uses SPI with the ADE as the master where the master sends the contents of specific registers one after the other at a set frequency of 8 kHz. There are three different sets of data that can be sent:

- Instantaneous voltage and current, conventional power (16 values)
- Instantaneous voltage and current (7 values)
- Conventional power (9 values)

All values are sent as 24-bit values, sign extended to 32-bits. The values can be sent as:

- 8-bit packages separated by gaps of 7 clock cycles
- 32-bit packages separated by gaps of 7 clock cycles
- No gaps

The clock frequency can be set to 4 MHz or to 8 MHz. The time required for these different combinations varies from 28 µs to 238.25 µs [50]. The fastest combination, at 28 µs, is 8 MHz clock speed, no gaps and sending only instantaneous values. Given that the ADE samples at 8 kHz, this leaves (125-28) = 97 µs for calculations and storing values per data set.

The HSDC to SPI set up is shown below in **Figure 4.11**.

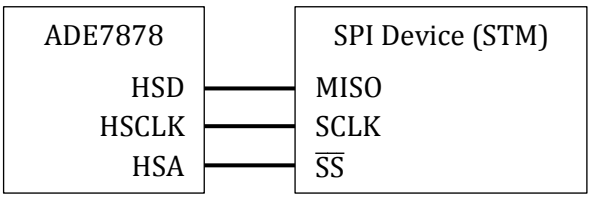


Figure 4.11 HSDC Connection Diagram

Figure 4.11 above, shows the standard connections. The line HSA / $\overline{SS}$ is the chip select line which goes low when the HSDC is communicating. This line was not used because to check the line and have an interrupt routine for it was too slow for the STM to collect uncorrupted data.

Thus from now on the connection lines shown will just be HSD and HSCLK and the chip select will occur in software.

4.4.2 DMA

One of the main issues in metering is having sufficient time between taking samples to perform the calculations. A method of recording data that does not involve the processor will allow more processing time. The STM has 8 DMA (direct memory access) channels with a capability for a double buffer system. The double buffer system allows the DMA to be storing data in buffer A while the processor is using the data in buffer B. The larger the buffer the more processing time in a processing block. In a system where the data must be processed in real time, the complete set of calculations must be carried out in each processing block.

If a buffer has a size of 44 800 bytes, and the data being recorded are just the instantaneous values, then the time for processing on buffer B while buffer A is being filled is:

- 44 800 bytes = 11 200 words
- 11 200 words = 11 200 / 7 data sets = 1 600 data sets
- 1 600 data sets taken at 8 kHz = 200 ms

Thus DMA allows (200 – 0.028) ms of uninterrupted processing time, a vast improvement on 97 µs.

4.4.3 Calculations

The recorded instantaneous values are in units that are consistent with each other but will need to be multiplied by a conversion factor to make them useful values. However the conversion factors are relative to the calculation, for instance the RMS conversion factors are different from the power calculation conversion factors. The instantaneous values are left in unit-less values because they are needed in that form only to show shape.

RMS

RMS calculations need to be made over a period of cycles or half cycles. There are two ways to find the end of a cycle, to count zero crossings or to have a timer based on the known frequency of the wave. The problem with counting zero crossings is that if the wave is distorted then there may be more than two zero crossings in a cycle, and the advantage of the generalised power theory is that it can measure distorted waveforms. Thus the known frequency value was used to calculate cycles. Frequency was assumed constant for this project because its purpose is to measure a nominal voltage of 230 V rms, which is required to be within ±1 Hz of 50 Hz [49]. The real time frequency could be calculated using Fourier analysis of the last full cycle before the one

being measured or an external module to measure frequency could be used. As stated earlier the measurement of frequency is not addressed in this proof of concept.

The method used for calculating RMS was to accumulate the square of the instantaneous values for the length of two cycles (0.02 * 2 = 0.04 s). Then divide the accumulated value by the number of data points used to make it. Finally find the square root of this value. This value will be in meaningless units so will need to be converted to volts or amperes in RMS and stored.

```
Frequency = 1 000 Hz
Therefore 20 data points per cycle

If (counter < 39)
      Accumulator = Accumulator + data^2
Else
      Accumulator = Accumulator + data^2
      RMS_value = SquareRoot(Accumulator / 40) * Conversion_Factors
      Set counter to zero
```

Figure 4.12 Example Pseudo Code for RMS Calculations

The conversion factors (the calculation of these is explained in section 5.2 basic calibration):

Binary to Volt RMS = $0.95520 * 10^{-4}$

Binary to Ampere RMS = $0.33804 * 10^{-5}$

For the calculation of RMS the phase gain calibration is used with the following conversion factors:

Current phase 1	1
Current phase 2	0.98906
Current phase 3	0.96789
Voltage phase 1	1
Voltage phase 2	0.99841
Voltage phase 3	0.97979

Generalised Power Theory

Figure 4.13 below shows a section of code used for the calculations of the generalised power. The calculations were taken from [1]. The instantaneous power values were averaged over two cycles as with the RMS calculations.

The assumed constants are:

- The value of the neutral wire resistance (r_n = 2 Ω)
- The value of the phase wire resistances (r_1, r_2, r_3) and that the three phase wire resistances are equal ($r_1 = r_2 = r_3 = r = 4\ \Omega$)

- The frequency value (f = 50 Hz)
- The conversion factors (The calculation of these factors is explained in section 5.2 basic calibration.):

  Binary to P, Q and S $= 0.33290 * 10^{-4}$
- That the conversion factors between each phase are all 1 (i.e. that the units in each phase are all equivalent) If they were not equal to one then the first step in the calculations shown below would be to multiply each current and voltage by its respective conversion factor to force one unit of current as measured for phase one to be equivalent to one unit of current measured for the other phases.

neutralWireResistance = r_n = 2, phaseWireResistance = r = 4

// these values were taken as an example

$$x = \frac{r_n}{r}$$

$$i_n = -(i_1 + i_2 + i_3)$$

$$sum(i^2) = ((i_1)^2 + (i_2)^2 + (i_3)^2 + x * (i_n)^2$$

$$\|\boldsymbol{i}'\| = \sqrt{(sum(i^2) * r)}$$

$$e_{ref} = \frac{(e_1 + e_2 + e_3)}{(3 + (\frac{1}{x}))}$$

$$sum(e - e_{ref}) = (e_1 - e_{ref})^2 + (e_2 - e_{ref})^2 + (e_3 - e_{ref})^2 + (-\frac{e_{ref}}{\sqrt{x}})^2$$

$$\|\boldsymbol{v_2}'\| = \frac{\sqrt{sum(e - e_{ref})}}{\sqrt{r}}$$

$$p = (e_1 - e_{ref}) * i_1 + (e_2 - e_{ref}) * i_2 + (e_3 - e_{ref}) * i_3 + (0 - e_{ref}) * i_n$$

$$s = \|\boldsymbol{v_2}'\| * \|\boldsymbol{i}'\|$$

$$pf = p/s$$

Average p, s and pf over 2 cycles to get **P, S** and **pf**

$$Q = \sqrt{S^2 - P^2}$$

P, S, Q and PF are multiplied by the relevant conversion factors

Figure 4.13 Example Pseudo Code for Power Calculations

4.4.4 Data Management

The EVAL sends data as a 7 word package where each word is sent MSB (most significant bit) first [51]. The DMA is set up to receive these as individual bytes because it can only receive the

values as half words or as bytes [55]. The STM's DMA can only use little-endian addressing. This means that if the result is stored as a word the bytes, it will be in the wrong order and thus meaningless when used like that in calculations, thus they are stored in bytes and converted into words in the right byte order as illustrated below.

Byte order as stored by the STM's DMA:

B0	B1	B2	B3

Where B0 is the least significant byte and B3 the most significant byte.

This little endian addressing needs to be changed to big endian:

B3	B2	B1	B0

Now the address of B3 can be used as a pointer to the 32 bit integer consisting of B3, B2, B1 and B0. This integer is then converted to a double.

Figure 4.14 Little Endian and Big Endian Addressing

At the same time as swapping the bytes, the values are stored in an array of doubles to allow for the calculations shown in the previous section which involve floating point values and also the accumulation of values which are each potentially 32-bit long integers.

The DMA uses a double buffer system. These buffers are 11 200 * 4 bytes long. The set of data in each buffer is changed to big endian addressing and stored as doubles (8 bytes long) in a separate array [56]. The buffers are set to their maximum capacity given the amount of RAM on the STM32.

The ADE7878 samples at 8 kHz, however only 1 kHz of data are needed and the HSDC cannot send less than the 8 kHz of data so the STM only records every 8^{th} set of data. Thus the array of doubles contains only an 8^{th} of the number of elements that the buffers used by the DMA and HSDC contain.

4.4.5 SD Library

An SD card can be read and written to in pages with no file system, however then the data are not accessible by a standard laptop. A file system takes more time to write, however the reason for choosing an SD card was the ability to easily move and read the stored data on a laptop.

The STM has a very good SD library for its SDIO port; however the library is orientated towards writing pages. An SD card file system library developed by Chan and Thomas [57] and modified by Patel [56] was used. The code written by Chan and Thomas has copyright that allows use and

modification. Patel modified it for use with the STM32 platform. In this research it was modified for use with C code and not freeRTOS.

The SD library is split into three parts:

- File writing (ff.c and ff.h, written by Copyright (C) 2012, ChaN)
- Disk writing (diskio.c and diskio.h, Copyright (C) 2010, Martin Thomas, ChaN)
- Date and time (fattime.c and fattime.h, Martin Thomas, 4/2009)

The SD library also has two header files: ffconfig.h and integer.h which just contain definitions.

The date and time section adds a specific date and time to each file. This is left in for completeness of the library, however it is set to a static date and time, 2012/12/06 12:30 PM, in this project.

The file writing section has functions that open, write, read and close the file as well as mount and unmounts the disk. These functions use methods from the disk writing section such as disk initialise, read and write. The disk writing section includes the STM specific code. The STM specific code uses two of the DMA_SPI channels which must be on a separate channel or stream to the DMA_SPI used for HSDC.

4.4.6 Files used in the Program

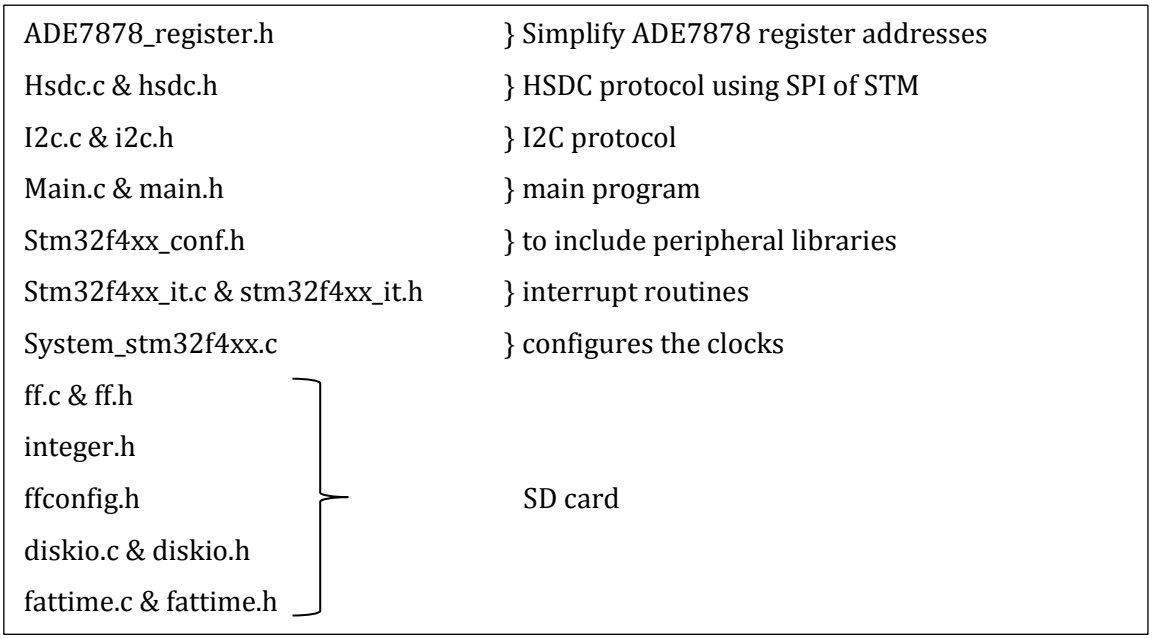


Figure 4.15 Files in Project

The peripheral libraries used are those for DMA, GPIO, SPI, I2C and RCC (system clock). They come standard with the STM32.

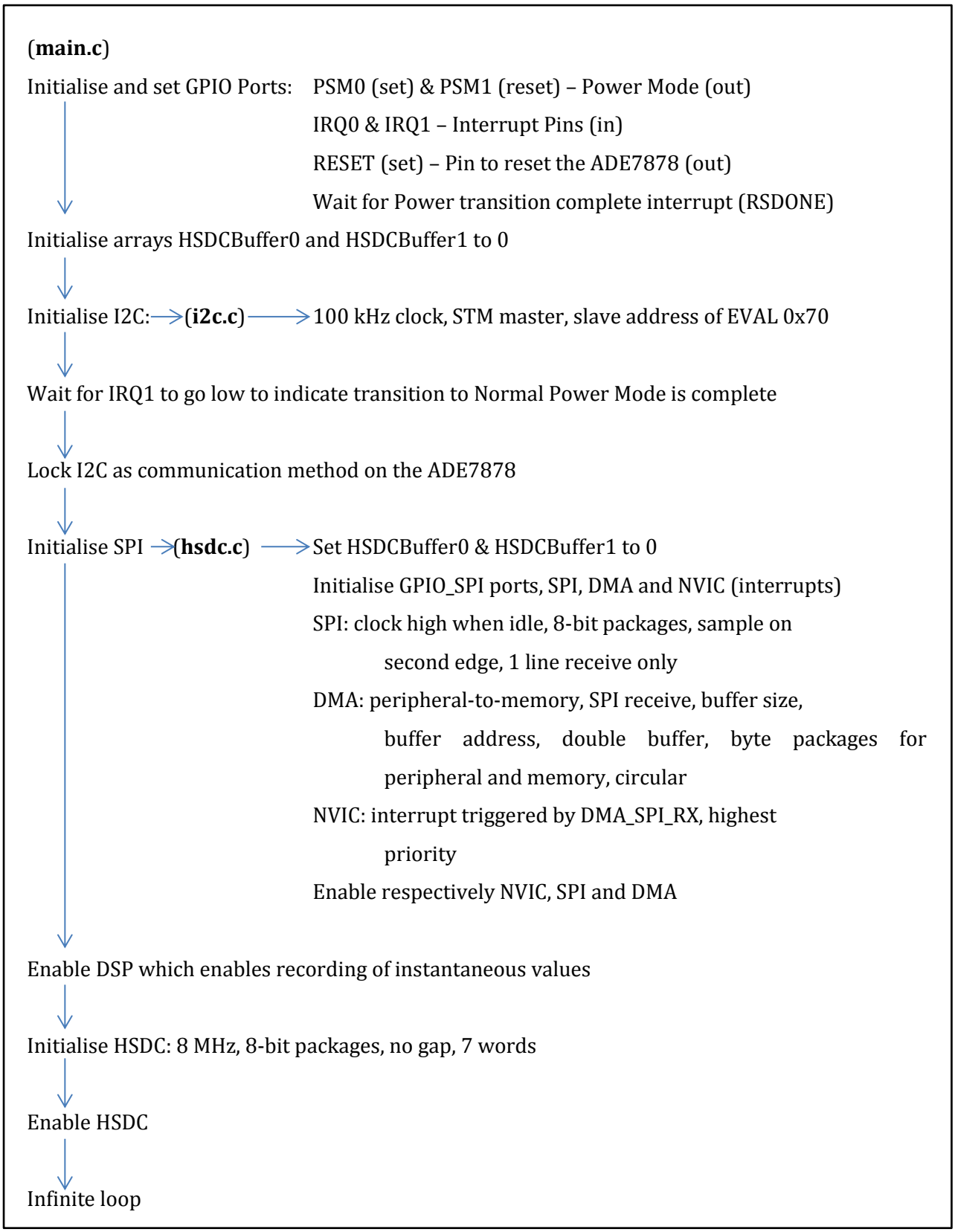


Figure 4.16 Main Program Flowchart

(**stm32f4xx_it.c**) DMA/SPI Interrupt Routine

Interrupt triggered

Test to ensure correct DMA stream and transfer complete

Discover which buffer in use by DMA and manipulate the other

Cycle through HSDCBufferx

Each 8th set of data: swap the bytes to correct order and save in double array

Perform calculations on array and store results to TempHSDCBufferx

SD: mount disk

Make a .dat file

Open .dat file

Write TempHSDCBufferx to .dat file

Close file

Mount null disk

Figure 4.17 Interrupt Routine Flowchart

4.4.7 C# Code

The data are stored on an SD card in .dat files. A C# program was designed to convert these binary files into .csv files. This program consists of opening the binary files, reading it using the BinaryReader method and storing the results doubles as in a .csv file which separates the data into the following format:

Table 4.4 Data Layout

Line Number	Instantaneous Current A	Instantaneous Voltage A	Instantaneous Current B	Instantaneous Voltage B	Instantaneous Current C	Instantaneous Voltage C	Instantaneous Neutral Current
1…400							
401…405	RMS Over 2 cycles	RMS	RMS	RMS	RMS	RMS	RMS
406…410	Real power, non-active power, apparent power, power factor Averaged over two cycles						

5 Experimental Procedure

All tests were conducted on the full system as discussed in chapter 4 and illustrated below in **Figure 5.1**, **Figure 5.2** and **Figure 5.3**.

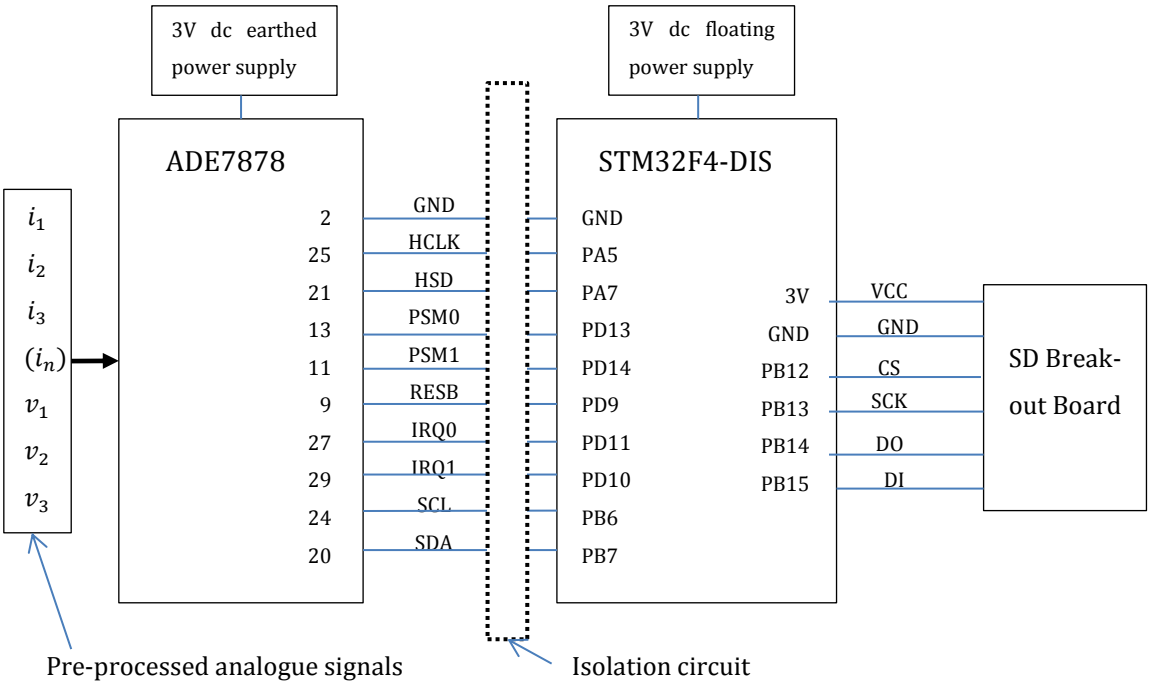


Figure 5.1 Full System Block Diagram

The system was powered by an Agilent E3620A 0-25 V, 0-1 A, dual output DC power supply. The supply to the ADE7878-EVAL evaluation board and the pre-processing circuit was +3 V and ground was connected to earth. The processing side of the circuit, the STM and SD board, was supplied by +3 V and a floating ground. **Figure 5.1** shows the connection pins used.

The neutral current i_n is in brackets because it does not need to be sampled. The ADE7878 will always record a value for the neutral current whether or not a CT is connected. At present the software just ignores this value, however it could be used as an error checking mechanism which is explained at the end of section 5.2.

Figure 5.2 and **Figure 5.3** are photographs showing the experimental arrangement. The ammeter, Yokogawa 201313 45-65 Hz, and the voltmeter, Yokogawa 201319 45-65 Hz, were connected to phase 1.

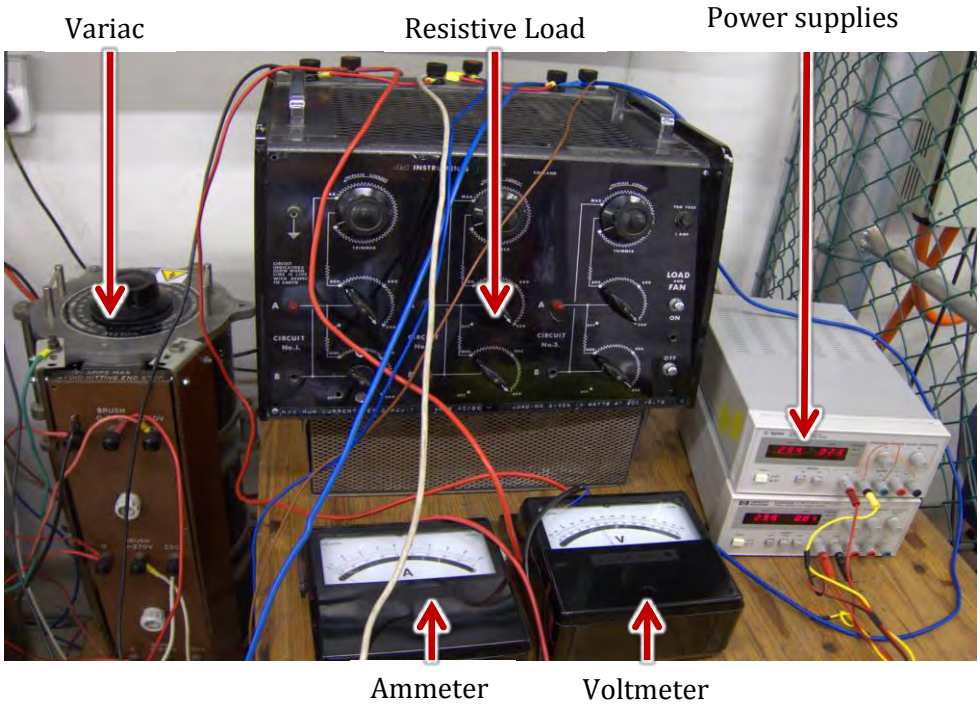


Figure 5.2 A photograph of the instruments used in the experimental setup showing the Variac, the resistive load, the two power sources, the voltmeter and the ammeter

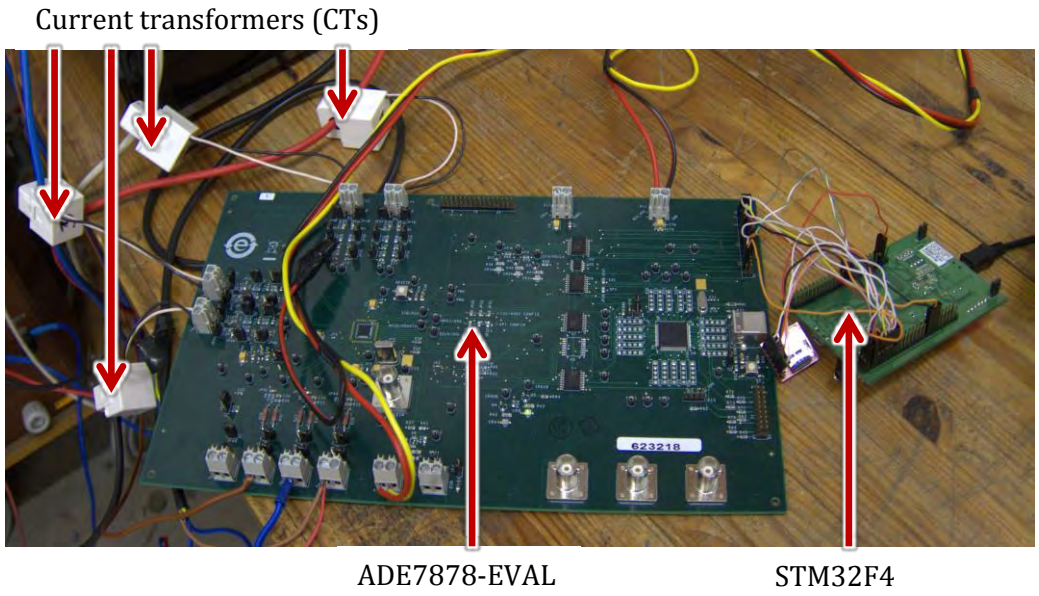


Figure 5.3 A photograph of the experimental setup showing the ADE7878 EVAL board connected to the STM32F4 Discovery Board

When running the tests in debug mode the STM32F4 is powered by the debug USB cable, however the processing side of the EVAL board still needs to be powered by the +3 V floating ground supply.

The resistive load was set up as shown with all dials on the lowest setting. The specific resistances were measured using a hand held multimeter, True Agilent u1272A, and are shown in the table below:

Table 5.1 Resistive Loads

	Phase 1	Phase 2	Phase 3
Resistance in ohms (Ω)	293	71	57

5.1 Data Corruption Tests

The testing of the communication – I2C and HSDC – was done with the set up in **Figure 5.1** with the exception that the analogue signals were not connected.

I2C communication was used between the STM and the ADE with the STM as master. SPI is also an option when using the ADE7878, however only if HSDC is not to be used. The clock speed was 100 kHz. The method of ensuring that the I2C communication was working correctly was to read from a register on the ADE7878 that had a known and fixed value. The register used was the check sum register with an address of 0xE51F and a default value of 0x33666787 [50].

The default mode at power-up is sleep mode (PSM3) which does not allow registers to be written to. Thus using external pins (PM0 and PM1) the ADE must be set to normal power mode (PSM0) where registers can be edited. Once the power mode transition interrupt, IRQ0, has been pulled the registers can be written to.

Due to a two stage pipeline, when writing to registers on the ADE7878, the last value to be written to a register must be written three times to ensure it has been written to RAM. The register chosen to test writing to a register was the CONFIG2 with an address of 0xEC01. This is the register used to ensure the communication method chosen cannot be changed without a reset. First the register's current value was read then the lock communication bit was set by a bitwise AND operation with 0x02. Then this register was read again to ensure that the write had occurred. Another simple register that can be written to, is the voltage gain register (AVRMS) with an address of 0x4301. Again this was written to and then read to check that the write occurred.

Testing the SD card reading and writing was done with the system illustrated by **Figure 5.4**. The software used was the SD Library and a main.c file. The main.c file simply wrote an array of

binary numbers to a .dat file and stored this on the SD card using the SD library. This was read using the C# code described in the previous chapter.

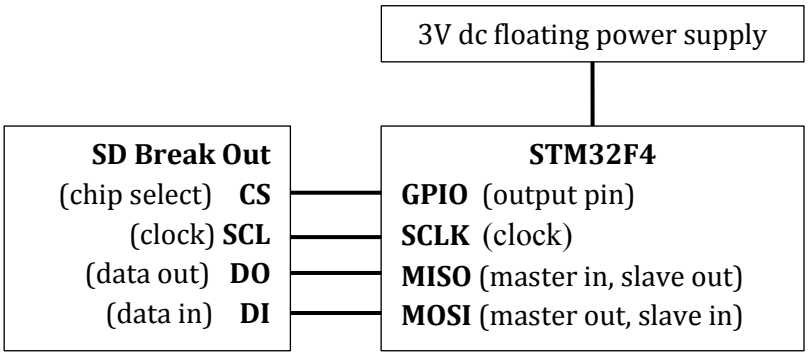


Figure 5.4 SD Card Test Set Up

Testing the HSDC interface was also done initially with an oscilloscope, Agilent Technologies DSO3062A digital storage oscilloscope, 60 MHz, 1 GSa/s in normal sweep mode, to ensure that under different settings, for instance 7 words sent with 7 clock cycle gaps in between, the data output were as expected, from the ADE7878 data sheet [50]. The ADE measures analogue data and converts it to digital using a delta-sigma ADC so there is no way to force the analogue signal to give a known digital signal before it has been calibrated. Even after calibration it would be impossible to force an exact 24-bit digital value.

Thus another system was needed to send known digital data in the same protocol format as the ADE's HSDC. The block diagram of this system is shown below.

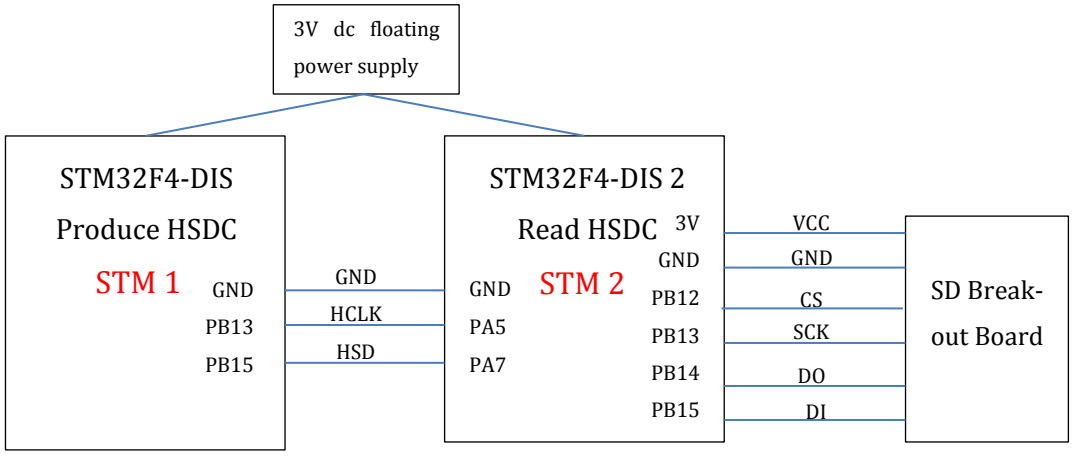


Figure 5.5 Block Diagram of HSDC Test set Up

The clock speed of the SPI was set to 5.25 MHz. This speed is lower than HSDC, however if a data set sent at the lower speed of 5.25 MHz is uncorrupted then it is likely that for a higher speed of 8 MHz the same set of data will remain uncorrupted. This is because the factor of greatest

influence is that the program will still be processing data when the DMA changes buffer which would corrupt the data that was being written. The HSDC-like set up was then implemented on a second STM as shown in the diagram above and various different set ups and frequencies tested for the limits of the system.

The software used for STM 1 is unchanged except for the main.c and main.h files. The main.c file for this set up contains only the command to begin receiving HSDC whereas for the previous set up it had all the ADE7878 initialisation code as well. The program used for the STM 2 is illustrated by **Figure 5.6.**

Initialise SPI – master mode, 1 line transmit

Initialise Pushbutton interrupt

Initialise timer and interrupt– the clock speed is 84 MHz, thus a prescaler of 13 was set so that the timer had a clock of 6 MHz, the period counter was set to 749 which causes an interrupt every 125 μs

---

Interrupt routine:
When pushbutton pressed:
    Set Pushbutton = 1
When timer reaches end of 125 μs period:
    If Pushbutton = 1
    Then Clear pending interrupts
        Set data to known array (numbered sets of increasing value the same size as the corresponding buffer size to be tested)
        Initialise DMA_SPI transmit: memory to peripheral, memory increment, normal mode, single buffer
    Enable DMA to transfer contents of data array once

Figure 5.6 HSDC Test Algorithm

The new power calculations were tested by writing the instantaneous values to the SD card and performing the calculations in the C# program. The results were tested analytically against the calculations used in [1]. Then the calculation code was moved to the microcontroller and added

line by line checking that the data were not corrupted. This was first done with the HSDC-like system. Then the data were tested by ensuring that the shape of the data was as expected when all the files were concatenated. The shape expected being a continuous sine wave. The shape of each sine wave was looked at and is described in section 5.3.

5.2 Basic Calibration

A simple hand held meter, True Agilent u1272A, was used to calculate the RMS conversion factors. This method of calibration is very crude and assumes that the RMS value will remain the same over the whole period of saving and recording the data as the hand held meter will give one reading and the data from the SD card will give several. So the Variac (Voltac type B, Yokoyama electric Works Ltd. Variable V. Transformer) was set to a value of about 230 V RMS and the resistive load set to allow about 1 A RMS to flow in phase 1. Then the full system was started and the value on the meter read. These were then compared and an RMS conversion factor was calculated. The meter value was taken to 3 significant figures and the average of the RMS values collected by the system over the time period of the test was used. The conversion factors for the new power calculations were not calibrated; they were just estimated based on the RMS factors. This is inaccurate; however considering that the wire resistances were given assumed values, the conversion factor is unimportant provided that the actual calculations are arithmetically correct.

The conversion factors used were:

Binary to Volt RMS = $0.95520 * 10^{-4}$

Binary to Ampere RMS = $0.33804 * 10^{-5}$

Binary to P, Q and S = $0.33290 * 10^{-4}$

In the paragraphs below an explanation of the necessary thorough calibration are described.

The ADE7878's calibration application note, [58], recommends the two following methods of calibration:

- Using a reference meter and the CF pulse generators and Fourier analysis
- Using an accurate source and then adapting the values in the ADE7878's registers

The registers are used for the actual program so it makes sense to calibrate using the registers as well. The accuracy of the source will have to be determined by another energy meter that is more accurate than the required accuracy of this power meter. The application note recommends calibrating at nominal current and nominal voltage at a power factor close to 1 and also at a power factor of 0.5 [58]. This will enable the meter to be calibrated at a range of non-

active powers. The range of accuracy at different power factors would also need to be investigated once a conversion factor is chosen. The choice of conversion factor will give the designed measurement range, and the other measured conversion factor will be a show of the accuracy out of the designed range.

The following steps would be carried out consecutively. The sequence is similar to that in [58], however as the gain registers will not be used the results would be stored in the various conversion factors.

1. Phase gain matching:
   Apply the same input currents and voltages to all phases then calculate as below:
   Conversion factor IB = AIRMS/BIRMS
   Conversion factor real power B = real power A/ real power B
   This will mean that each unit on each phase will have the same weight and so one overall conversion factor can be applied
2. RMS Gain
   Conversion factor binary to volts [V/LSB] = Voltage Input [V] / VRMS [LSBs]
3. As above for all other conversion factors
4. Phase error due to CTs. When an inductive load of power factor 0.5 is applied the current on the same phase lags the voltage by 60 degrees so the expected lag can be compared to the actual lag and thus the phase error stated. Each phase may have a different phase error as each has a separate CT.

The conversion factors are used at the end of the calculations because they are calculated relative to the RMS and calculated power and not relative to the instantaneous voltage and current values. It is inaccurate to use the RMS current conversion factor to convert instantaneous current to amperes. Thus the instantaneous values of current and voltage are shown in binary or per unit to express only the shape of the graph. The phase gain matching enables the use of one conversion factor for all phases.

These conversion factors can also be made as a sliding scale of conversion factors. This means that each separate range of values will have a separate conversion factor. This would allow a greater range of accuracy. The conversion factors are defined in software which will allow for a sliding scale of conversion factors based on the input range.

The neutral wire was not measured here; however the circuit can be extended to measure it. The value of the neutral current can be calculated using Kirchhoff's law, thus it does not need to be

measured. However, when the data logger is accurately calibrated, the measured neutral wire current can be used as a means of error and accuracy checking.

5.3 Testing Accuracy of Actual Data

The most obvious test for the data is ensuring the shape of the graph is a sine wave with a period of 0.02 s and given that the sampling is 1 kHz, there should be 20 data points per cycle. The shape was compared to that gained from an oscilloscope, Agilent Technologies DSO3062A. When all three phases are measured they should be 120 degrees apart. This was tested using the Variac (Voltac type B, Yokoyama electric Works Ltd. Variable V. Transformer) to set to about 120 V RMS, no load. The resulting graphs are shown in the next chapter.

The data were also tested against a hand held RMS meter, True Agilent u1272A, with the roughly calibrated conversion factors and found to be correct to 2 significant figures, therefore giving the expected accuracy, for different voltages and currents varying from 0-5 A RMS and 0-230 V RMS.

The results of the new power calculations were compared to the result of placing the binary instantaneous values into the excel spread sheet given by [1]. These calculations were given in detail in sections 2.1 and 4.4.3.

6 Results

The conversion factors are specific to particular calculations for instantaneous power or RMS. Thus when looking at waveforms, the data are displayed in unit less values. The horizontal axis of these waveforms represents the position in the array and represents time relative to when the data were being recorded.

The results shown in this section are for an unbalanced system with resistive load and the effect of the new power calculations is not illustrated by them. The effect of the new power calculations is shown by. This sections aims to show that the data logger is capable of performing the calculations without error. This is done by assuming specific phase and neutral wire resistances and comparing the data computed by the logger to that done manually. This section also demonstrates the type and format of data that can be accessed using the logger.

Please refer to Appendix D for corrections made to the data used in this chapter.

This chapter displays the results from the experiments described in the previous chapter. First the shape of the resultant waveform is investigated. Then the data as read from the .csv file is shown. This data are then displayed in graphs. Finally the limiting factors of the experiments are stated.

6.1 Waveform Shape

A single cycle of data taken with the variac (Voltac type B, Yokoyama electric Works Ltd. Variable V. Transformer) set to about 120 V RMS, no load is shown below in **Figure 6.1**. The data logger was recording at 2 kHz. 120 V RMS was chosen so that the shape could be clearly displayed on the Agilent DSO3062A oscilloscope whose display is shown in **Figure 6.2**.

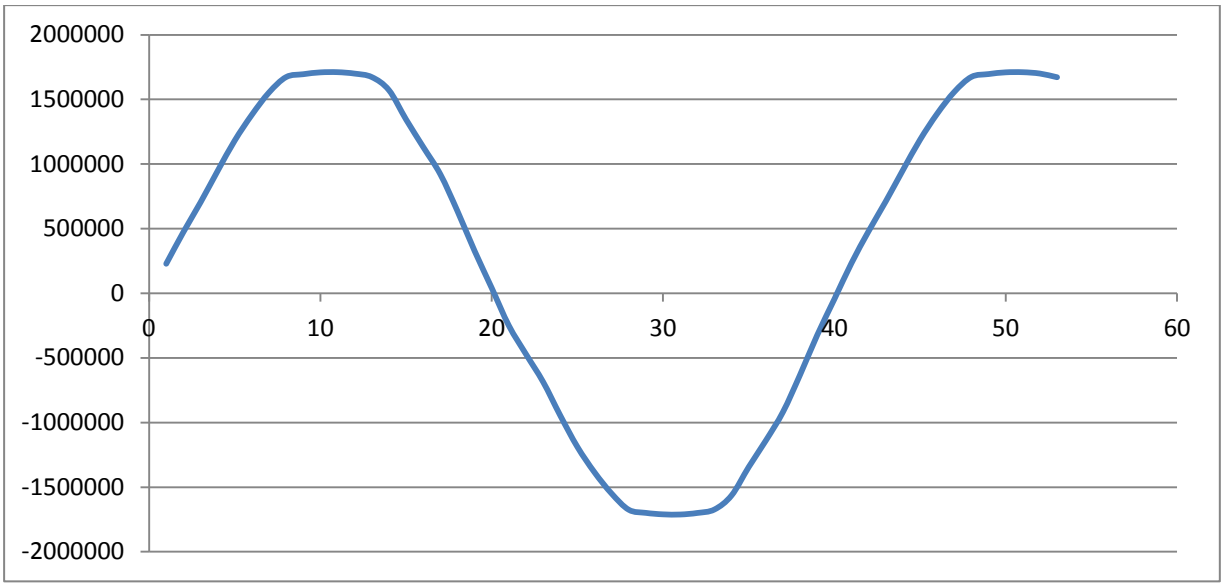


Figure 6.1 Voltage Phase 1 Waveform. This figure shows unit-less voltage relative to time. The time axis refers to sample number.

The above figure, **Figure 6.1**, clearly shows a sine wave with 40 data points per cycle. Given that the data logger was set to sample at 2 kHz this is clearly the expected period 0.02 s, showing a waveform of 50 Hz. The shape of the sine wave is a little flatter than expected by a perfect sine wave; however it is the same shape as that given by the Agilent Technologies DSO3062A oscilloscope shown below in **Figure 6.2**.

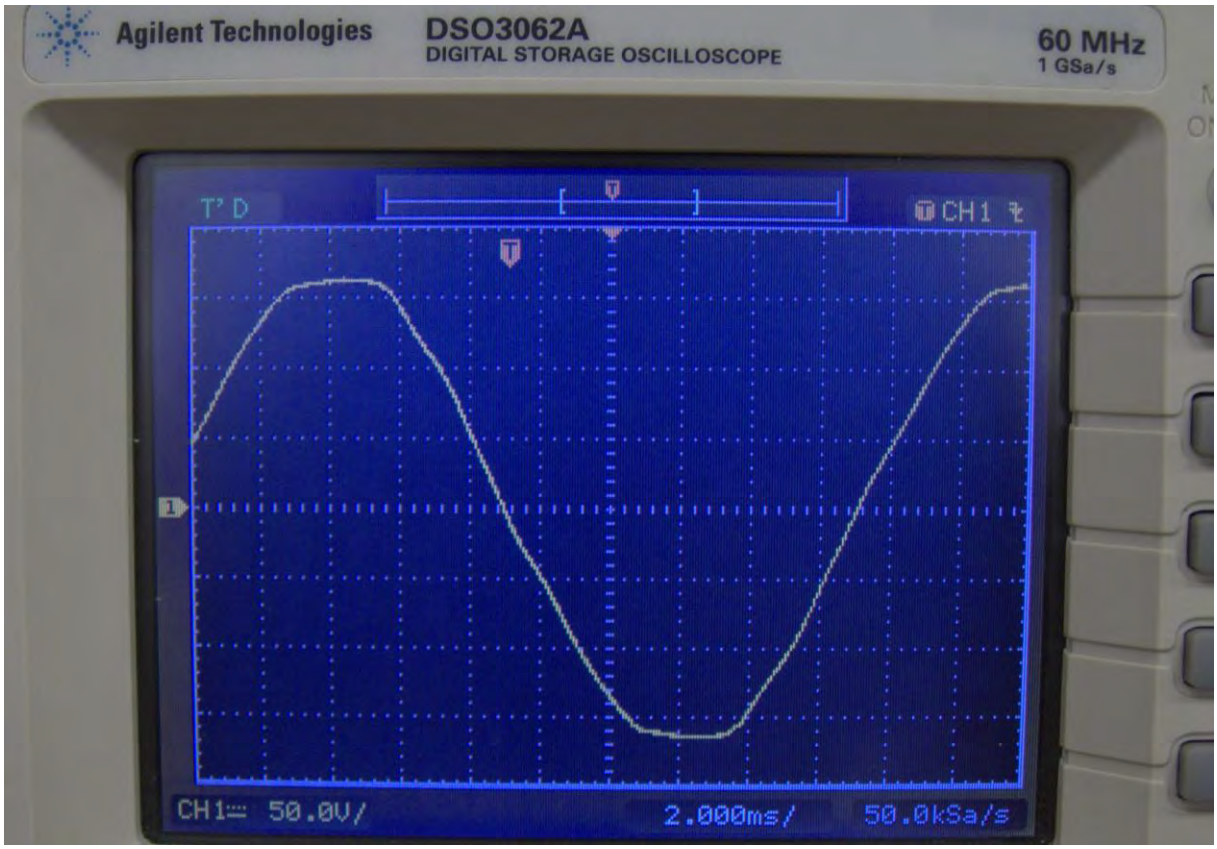


Figure 6.2 Voltage Phase 1 Waveform shape shown on Agilent DSO3062A oscilloscope. This figure shows voltage, at 50.0 V per division, relative to time, at 2.000 ms per division.

The waveform above, **Figure 6.2**, shows both the flattened peak and the slightly distorted shape between the peak and the zero crossing. This is the same as seen in **Figure 6.1**.

6.2 Result Data

Table 6.1 shows the results from the third data set of an experiment done under the conditions of the resistive load described earlier, with the variac (Voltac type B, Yokoyama electric Works Ltd. Variable V. Transformer) set to 200 V RMS. The data logger was set to record at 2 kHz. The sections of **Table 6.1** in red were added as explanations.

Table 6.1 Excerpt from csv file of results. Lines 1 through 400 display unit-less values. Lines 401 through 405 current values in amperes and voltage values in volts. Lines 406 through 410 display real power in W, apparent power in VAR, non-active power in VA and power factor is unit-less. The conversion factors used for these calculations are displayed in the paragraph following this table.

Line number	Current phase 1	Voltage phase 1	Current phase 2	Voltage phase 2	Current phase 3	Voltage phase 3
1	21113	632906	-1589701	-2811709	1074042	2202792
2	-34187	159189	-1554839	-2759970	1199454	2504056
3	-85716	-309573	-1434138	-2557867	1289414	2729301
4	-127213	-711247	-1201146	-2166410	1354938	2895363
5	-167738	-1085114	-1004123	-1829517	1354461	2927984
6	-212133	-1511376	-782283	-1449201	1345193	2941639
7	-252066	-1910464	-508386	-977801	1329916	2935113
8	-282379	-2248996	-219077	-481117	1304849	2905196
9	-305503	-2519328	54254	-7799	1242163	2790305
10	-322175	-2733368	305683	434674	1078746	2466062
11	-326240	-2824710	521329	818208	889631	2079901
12	-319244	-2845068	748537	1220217	725725	1743511
13	-313638	-2853060	984413	1641179	519152	1306131
14	-305473	-2845642	1187970	2011577	270302	778967
15	-296155	-2816634	1353372	2318487	45604	302665
16	-275495	-2692246	1483788	2567598	-172723	-162009
17	-225983	-2323882	1587581	2757761	-364549	-594390
18	-180022	-1957635	1608905	2809358	-535935	-976462
19	-137501	-1627715	1613193	2831666	-725619	-1398095
20	-85012	-1174347	1606243	2831824	-909212	-1814825
Continued ...						
497	259124	2575771	-1524514	-2638950	239540	314171
498	209468	2174517	-1607150	-2794693	422062	720734
499	167622	1854141	-1612262	-2822425	604248	1125816
400	121056	1485201	-1611927	-2835144	790691	1548585
Calculated per two cycles	Current RMS phase 1	Voltage RMS phase 1	Current RMS phase 2	Voltage RMS phase 2	Current RMS phase 3	Voltage RMS phase 3
401	0.79732	200.114	3.87166	197.7679	3.29135	200.152
402	0.797286	200.1242	3.871737	197.7763	3.29096	200.1405
403	0.797195	200.1099	3.871425	197.7551	3.291424	200.1595
404	0.797362	200.1004	3.871092	197.79	3.291995	200.223
405	0.796781	200.013	3.872917	197.8719	3.291928	200.2189

Calculated per two cycles	-	Real power	Apparent power	Non-active power	Power factor	-
406	0	1637.68702	1955.896912	1069.35212	0.825282	0
407	0	1634.579975	1952.547594	1067.984276	0.825207	0
408	0	1631.672877	1949.346033	1066.580319	0.825057	0
409	0	1629.55255	1946.776643	1065.12806	0.824959	0
410	0	1627.353891	1944.31838	1063.99872	0.824849	0

The power calculations are displayed respectively, i.e. the calculation results for the first two cycles (rows 1-40) are shown in rows 401 and 406.

The constants are found in stm32f4xx_it.c. the constants used for the results shown in this chapter are as follows:

```
double conversionFactorBinaryToVolt = 0.95520 / 10000;
double conversionFactorBinaryToAmp = 0.33804 / 100000;
double conversionFactorBinaryToP = 0.95520 / 10000 * 0.33804 / 100000;
double conversionFactorBinaryToS = 0.95520 / 10000 * 0.33804 / 100000;
double conversionFactorBinaryToQ = 0.95520 / 10000 * 0.33804 / 100000;
double conversionFactorBinaryToPF = 1;

double conversionFactorV1 = 1;
double conversionFactorV2 = 0.99841;
double conversionFactorV3 = 0.97979;

double conversionFactorI1 = 1;
double conversionFactorI2 = 0.98906;
double conversionFactorI3 = 0.96789;

float neutralWireResistance = 2, phaseWireResistance = 4;
```

Please refer to the CD for the .dat and .csv files used for these results.

6.3 Graphs

The data used in this section is the same as section 6.2. An example of current and voltage waveforms recorded by the data logger are shown in **Figure 6.3** and **Figure 6.4. Table 6.2** and **Table 6.3** show the RMS values for the first two cycles of those waveforms respectively.

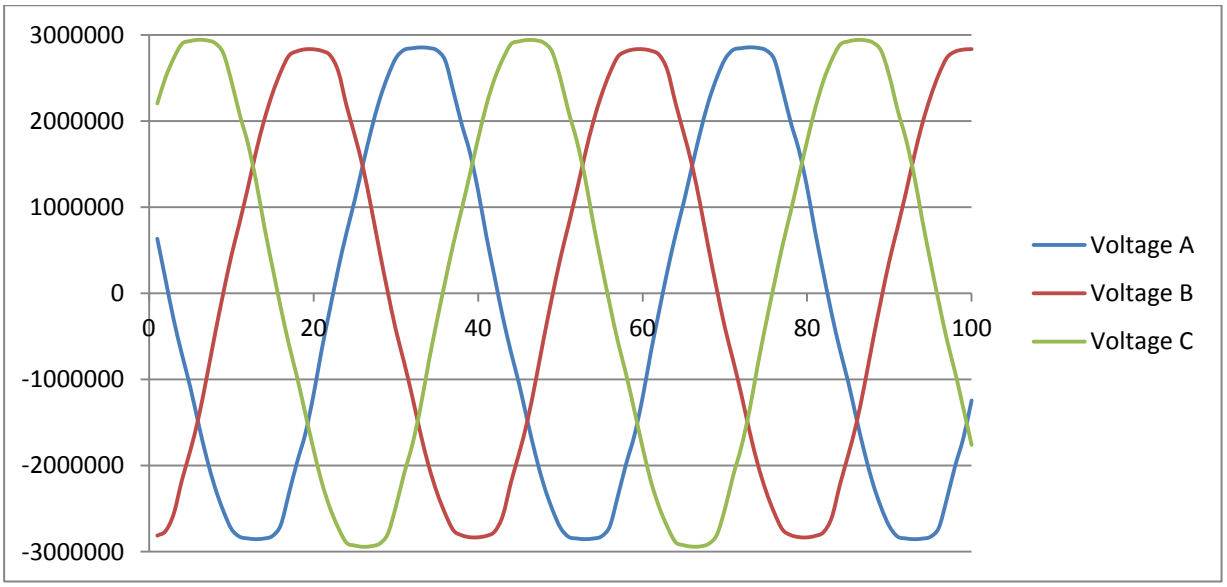


Figure 6.3 Three Phase Voltage Waveforms. This figure shows unit-less voltages relative to time. The time axis refers to sample number.

Table 6.2 RMS Voltage

	Voltage 1	Voltage 2	Voltage 3
Voltage in Volts RMS	200.1140	197.7679	200.1520

As expected the three phases are about 120 degrees apart and showing about 200 V RMS.

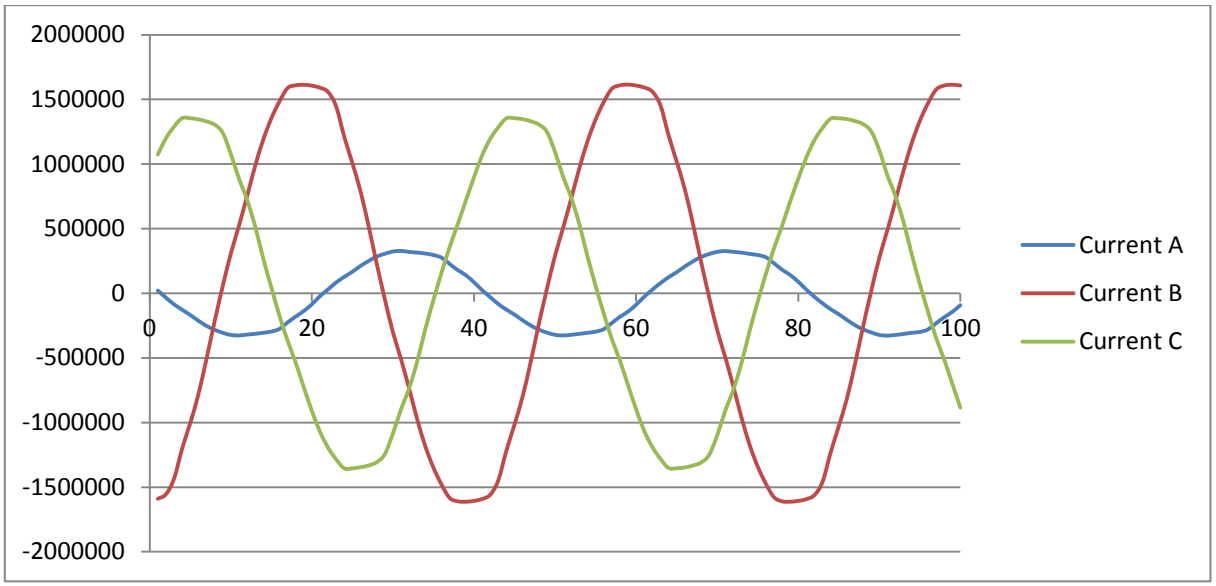


Figure 6.4 Three Phase Current Waveforms. This figure shows unit-less currents relative to time. The time axis refers to sample number.

Table 6.3 RMS Current

	Current 1	Current 2	Current 3
Current in Amperes RMS	0.79732	3.87166	3.29135

These currents are as expected given the resistances on each phase shown in **Table 6.4** which is repeated below:

Table 6.4 Resistive Loads

	Phase 1	Phase 2	Phase 3
Resistance in ohms (Ω)	293	71	57

Table 6.5 shows the non-active, real and apparent power as well as power factor for the above set of data.

Table 6.5 New Power Calculations

P (W)	S (VA)	Q (VAR)	PF
1637.68702	1955.896912	1069.35212	0.825282

The Q in **Table 6.5** refers to the non-active power as defined by Malengret's algorithm. The power (P) is positive and the power factor is close to 1 which is as expected for a purely resistive load.

Power factor calculated using the P and S values shown in **Table 6.5** gives a power factor value 0.838033 not that of 0.824964 given by the table. The reason for this is that the data for PF is taken from averaging the instantaneous PFs over two cycles. This is best illustrated by showing a running average of PF up to two cycles:

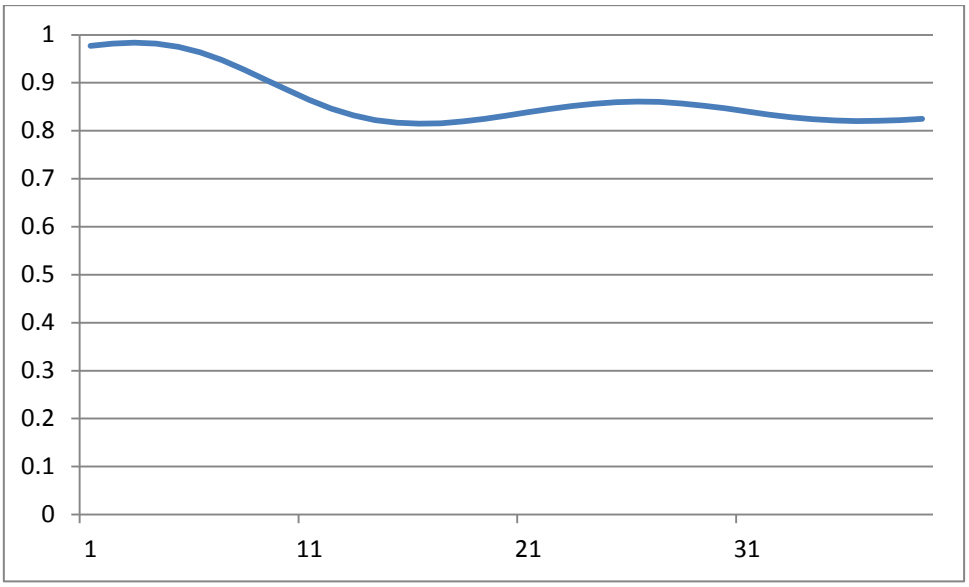


Figure 6.5 Moving average Power Factor. This figure shows power factor relative to number of values averaged over.

Figure 6.5 shows that the running average of PF will converge to the calculated PF, provided none of the external factors change. The resulting PF's accuracy requirement will depend on the accuracy requirements of the meter for a specific application. However this accuracy can be increased by averaging over more than 2 cycles which takes no more memory than the system currently uses and thus is possible.

6.4 Limit

The limits of the system were tested by using a variety of settings for the HSDC protocol and a variety of DMA buffer sizes.

The DMA buffer has an upper limit of 65 535 bytes [55]. The smallest that this buffer could be set to without data corruption was 10 000 bytes. When the DMA buffer was less than this the data were completely incoherent. The reason for this is discussed in the following chapter.

The range of frequencies that the data logger was capable of recording accurately using each setting was tested. The frequencies of data recorded by the logger that were tested were limited to factors of 8 kHz because that was the set frequency of the HSDC protocol. Below 400 Hz is not of interest because this will only measure up to the 4^{th} harmonic. The upper limit is the highest frequency of data recorded at which the code will compile and none of the data are corrupted. The reasons for this are discussed in the next chapter.

First the HSDC was set up to send the instantaneous current and voltage readings as well as the conventional power calculations. The highest frequency for this was found to be 500 Hz. At higher frequencies the data are only recorded from one buffer so sets of about 44 kB of data, which is the size of a buffer, was missing. At 800 Hz the instantaneous data could be recorded without corruption only if no calculations are carried out. This is further discussed in the following chapter.

When the HSDC is set to send only instantaneous values and the ADE7878 uses a two buffer system for recording then the frequency of data recording has a maximum of 1 000 Hz. When a single buffer system with only instantaneous values was used, the maximum was 2 000 Hz. The reasons for these choices are discussed in the following chapter.

7 Discussion

The first section of this chapter will consider the reasons for sampling frequency limits. Then the actual RAM requirements of the system in comparison to the estimate made in Chapter 3 will be investigated. It will then consider the limitation created by the HSDC due to the fact that it samples at a fixed rate of 8 kHz. The most influential requirements of a data logger in retrospect are summarised. The limits on the accuracy of the system are discussed.

7.1 Limits on Frequency

There were two reasons for the upper limits of frequency of recorded data in the previous chapter:

- A compile error (occurred when HSDC set up as instantaneous values only)
- Data corruption (occurred when HSDC set up as instantaneous values and conventional power and when the DMA buffer size was less than 10 000 bytes)

The compile errors were investigated and the reason for them found to be that the processor had insufficient RAM. This is discussed further in section 7.2.

The data corruption, when the DMA buffer size was less than 10 000 bytes, was incoherent. When the buffer size was too small the DMA would begin writing over the buffer that the calculations were using. This was because the calculations were taking longer than the time to fill one DMA buffer.

The data corruption, when the HSDC was set up to send instantaneous values and conventional power, was found to occur in a distinct pattern. The first data set would be uncorrupted within itself, but the second data set would not follow on sequentially. From the second data set onwards the sets would not be sequential but would rather have a constant amount of data missing. The amount of data missing would be equivalent to the size of a single buffer. The first gap between data set one and two is due to the set up time of the DMA. This initial set up could not be made shorter. However no matter how long the program ran for this style of error only occurred between set one and set two. A delay before set one was incorporated but this did not

solve the problem. Thus the first set of data was ignored and from the second set onwards was used.

The other error of data corruption that occurred when the HSDC was set up to send instantaneous values and conventional power was when the HSDC sends both instantaneous values and conventional power, the logger has a limit of 500 Hz. This limit is shown by the fact that the data are continuously being read out of one buffer. This is explained by the way that the DMA and interrupts for it work, as explained below.

The program checks to see which buffer is in use. If the DMA is writing to HSDCBuffer0 then it will use the data in HSDCBuffer1. When the calculations are finished, it waits for the next interrupt and repeats the process. If the calculation time was so long that it was still processing the data from the HSDCBuffer1 when HSDCBuffer0 has finished being written to, then the program will go to the waiting interrupt when the calculations are finished, but the available buffer will once again be HSDCBuffer1 instead of HSDCBuffer0. This is also the reason that the values from HSDCBufferx are first copied to TempHSDCBufferx before being manipulated. If they were not copied then the data would be written over while being manipulated and this type of corrupt data would be very difficult to debug.

7.2 Recalculation of required RAM

The buffer sizes used initially, when the HSDC was set to record instantaneous values only and the ADE had two recording buffers, are shown below:

```
#define BUFFER_SIZE_IN_WORDS 11 200

int8_t HSDCBuffer0[BUFFER_SIZE_IN_WORDS*4];
int8_t HSDCBuffer1[BUFFER_SIZE_IN_WORDS*4];

double TempHSDCBuffer0[((BUFFER_SIZE_IN_WORDS)  / 8) + 70];
double TempHSDCBuffer1[((BUFFER_SIZE_IN_WORDS)  / 8) + 70];
```

Buffer used by DMA, in bytes

Buffer used by program for calculations, in doubles (8 bytes)

Thus RAM used for the main buffers is:

((11 200*4)*2) + ((11 200/8)*2+70*2)*8 = 113 120 = 113.12 kB

The RAM available to the STM32F4 is stated as 192 kB; however it is split into 128 kB SRAM and 64 kB CCM. CCM is core-coupled-memory and is actually not accessible to the DMA, however it

does increase performance and decrease latency [55]. Thus the actual available RAM for buffers in this project is 128 kB. The reason that in certain configurations the frequencies greater than 1 000 Hz will not compile is that they exceeded this limit.

The results showed that the lower limit of BUFFER_SIZE_IN_WORDS is 10 kB due to the time needed for calculations.

Another restriction is the maximum size for the DMA buffer, which is 65 535 bytes. This dictates that BUFFER_SIZE_IN_WORDS can be a maximum of 65 535/4 = 16.3 kB. However for this purpose it cannot reach this due to the RAM limitation.

The BUFFER_SIZE_IN_WORDS was chosen as 11 200 because it was close to the limit of RAM, 12.69 kB, as well as a convenient number of cycles. The limit was calculated as follows:

```
BUFFER_SIZE_IN_WORDS = B
B*4*2 + (B/8+70)*2*8 = 128 kB
B ≤ 12.69 kB
```

However one possible way to increase this limit is by having only one TempHSDCBuffer. The reason two were used was convenience of debugging. Using only one buffer would mean that the BUFFER_SIZE_IN_WORDS could be as much as 14.16 kB.

Thus for a frequency of 1 600 Hz:

```
B*4*2 + (B/5 + 70)*2*8 = 128 kB
B ≤ 11.37 kB (one buffer: 13.28 kB)
```

And for a frequency of 2000 Hz:

```
B*4*2 + (B/4 + 70)*2*8 = 128 kB
B ≤ 10.62 kB (one buffer: 12.74 kB)
```

Thus it is possible, if needed for a particular application, to increase the frequency to 2 000 Hz.

7.3 RAM

The calculations in the previous section assume that all that is required of the microcontroller is to collect the data, manipulate it and store it to an SD card. A data logger will need communication such as 3G and a way for the user to request preferences. These will require processing power.

Two possible methods to increase the processing power if needed are:

- Use a more powerful processor.
- Instead of saving the data to a SD card, use that processor time to send the calculated data via USB or I2C to another microcontroller whose purpose is to interface with the user. This second microcontroller will not need to perform any more data calculations so that will allow it enough time to interface with the user.

7.4 Important Requirements

7.4.1 Measurement Frequency

The ADE7878 has a fixed measurement frequency of 8 kHz which means that the HSDC protocol is also fixed at sending sets of data every 8 kHz. If the measurement frequency could be adjusted this would give the microprocessor more time to perform calculations and, in further developments, more time to interface with the user.

7.4.2 Speed of Memory

An SD card typically takes 8 ms to perform a write cycle, and even longer when writing bigger files. Thus the time taken to write to memory is an important aspect. The current speed could be improved by modifying the SD library and subsequently the speed category of the SD card.

7.4.3 DMA

DMA is a very necessary requirement because it frees up the processor for other processes while still recording data. This could also be accomplished by multiple cores and an operating system. Both of these require very complex systems in which the exact process of the program will be unknown so difficult to ensure no corruption occurs when processing real time. However, with good tests it could be written. Also, the DMA buffer size could be increased along with the RAM size.

7.5 Limitations on Accuracy

The accuracy of the data is affected by the accuracy of each part of the system. The system used here will have multiplicative errors because each section uses the information produced by the preceding section.

When the system is accurately calibrated it will still have the errors of certain sections:

- The CTs have a limited accuracy. The ones used in this system are Class 1 accuracy and therefore have 1% ratio accuracy [59].
- The conversion factors themselves will be limited to the accuracy of the instrument or source that they were calibrated to as well as rounding errors when using the factors.

- The calculations using doubles will be subject to rounding errors though these are usually as small as 10^{-12} units.
- The results of the calculations may vary depending on the number of cycles the data is averaged over and the rate of sampling.
- Each component will have accuracies that depend on environmental factors such as temperature.

If the instrument needed to be more accurate these could be improved by using a higher accuracy CT, choosing the calibration equipment with the required accuracy and testing to investigate the most accurate range of cycles to average over. The latter, however, would be subject to a specific standard if the logger were to be produced commercially.

8 Conclusions

This dissertation has shown that a metering concept that is based on equipment cheaper and smaller than a laptop is able to meet the requirements of power measurements, such as those for non-active power, that need relatively high frequency, instantaneous sampling.

A meter based on a measurement chip, ADE7878, and a microcontroller, STM32F4, is able to perform the non-active power calculations in real time and store them to an SD card. The sampling rate is relatively high at 1 000 Hz, however this can be increased to as much as 2 000 Hz. Thus it is possible to create a metering concept that is fast enough for calculations such as the non-active power calculations.

In order to expand it to a full data logger there are various important specifications to consider when choosing the technologies. The aspects of importance are microcontroller speed, available RAM, speed of the memory and the measurement frequency flexibility. The accuracy is mostly dependant on the accuracy of the calibration; however there are some aspects that will remain the same such as the accuracy class of the CTs and the rounding errors.

One of the main limitations of using an embedded system is that extra RAM and flash cannot easily be added for improvements on the system. Thus all aspects requiring processing power need to be designed before the main system is chosen.

9 Recommendations

Further work in this field is to investigate a method to incorporate accurate, real-time frequency measurement. The frequency will need to be measured real time and so each data set will need the previous cycle in order to calculate the frequency to use in the calculation. Another option is to use a separate module to measure the frequency which feed into this system real time.

A communication system was not investigated here. It is recommended that GSM or 3G is used for long range communications. The time required for communications will need to be calculated and then the system adjusted so that there is sufficient processing time. This could be achieved easily if the instantaneous values are not needed by the user and only the results of the communications. Two other methods to increase the processing power if needed are to use a microcontroller with more RAM, a faster microcontroller or by using a second microcontroller system for the communications and have a series of shared SD cards that work in a similar manner to the double buffer system.

The first data set is discarded. This could be investigated with the purpose of achieving accurate data from the first set.

A full data logging system including power source, communications and user interface can be developed using both this dissertation and Stowe's dissertation.

List of references

[1] C. T. Gaunt and M. Malengret, "True power factor metering for m-wire systems with distortion, unbalance and direct current components," *Electric Power Systems Research*, vol. 95, no. 1, pp. 140–147, Feb. 2013.

[2] M. Malengret and C. T. Gaunt, "General theory of average power for multi-phase systems with distortion, unbalance and direct current components," *Electric Power Systems Research*, vol. 84, no. 1, pp. 224–230, Mar. 2012.

[3] M. Malengret and C. T. Gaunt, "General theory of instantaneous power for multi-phase systems with distortion, unbalance and direct current components," *Electric Power Systems Research*, vol. 81, no. 10, pp. 1897–1904, Oct. 2011.

[4] G. Stowe, "Design of an Energy Profiler for Domestic Load Research Analysis," M.S. Thesis, Dept. Elec. Eng., Univ. of Cape Town, Cape Town, South Africa, 2012.

[5] "IEEE Standard Definitions for the Measurement of Electric Power Quantities Under Sinusoidal, Nonsinusoidal, Balanced, or Unbalanced Conditions." pp. 1–50, 2010.

[6] C. T. G. M. Malengret, "Definition of Apparent Power in poly-phase systems." [Online]. Available: http://web.uct.ac.za/staff/gaunt/saupec2005malengret.pdf. [Accessed: 21-Feb-2013].

[7] C. T. Gaunt and M. Malengret, "Why we use the term non-active power, and how it can be measured under non-ideal power supply conditions," in *IEEE Power and Energy Society Conf. and Expo. in Africa: Intelligent Grid Integration of Renewable Energy Resources (PowerAfrica), Johannesburg, South Africa*, 2012, pp. 1–4.

[8] SmartGrid, "Time-Based Rate Programs." [Online]. Available: http://www.smartgrid.gov/recovery_act/deployment_status/time_based_rate_programs. [Accessed: 01-Nov-2013].

[9] "Smart meter policies around the world | eMeter." [Online]. Available: http://www.emeter.com/smart-grid-watch/2011/smart-meter-policies-around-the-world/. [Accessed: 04-Mar-2013].

[10] S. Kaufmann, K. Künzel, and M. Loock, "Customer value of smart metering: Explorative evidence from a choice-based conjoint study in Switzerland," *Energy Policy*, vol. 53, no. 1, pp. 229–239, Feb. 2013.

[11] S. Borenstein, M. Jaske, and A. Rosenfeld, "Dynamic Pricing, Advanced Metering, and Demand Response in Electricity Markets," California, Oct. 2002.

[12] Itron, “Itron Awarded Largest Smart Meter Contract in South Africa with City of Johannesburg.” [Online]. Available: https://www.itron.com/newsAndEvents/Pages/Itron-Awarded-Largest-Smart-Meter-Contract-in-South-Africa-with-City-of-Johannesburg.aspx. [Accessed: 07-Mar-2013].

[13] P. Bilik and D. Kaminsky, “Modular and flexible power network analyzer,” in *2010 International Conference on Applied Electronics (AE), Pilsen*, 2010, pp. 1–4.

[14] G. McKinstry, S. Galloway, and I. Kockar, “An initial assessment of the potential impact of smart metering on a decentralised energy network,” in *Universities Power Engineering Conference (UPEC), 2010 45th International. IEEE, Cardiff, Wales*, 2010, pp. 1–4.

[15] “News « Stop Smart Meters! (UK).” [Online]. Available: http://stopsmartmeters.org.uk/news/. [Accessed: 04-Mar-2013].

[16] C. Efthymiou and G. Kalogridis, “Smart Grid Privacy via Anonymization of Smart Metering Data,” in *2010 First IEEE International Conference on Smart Grid Communications, Gaithersburg, MD*, 2010, pp. 238–243.

[17] C. King, “California PUC adopts consumer data access and privacy rules for smart meters | eMeter.” [Online]. Available: http://www.emeter.com/smart-grid-watch/2011/california-puc-adopts-consumer-data-access-and-privacy-rules-for-smart-meters/. [Accessed: 04-Mar-2013].

[18] M. Weiss, A. Helfenstein, F. Mattern, and T. Staake, “Leveraging smart meter data to recognize home appliances,” in *2012 IEEE International Conference on Pervasive Computing and Communications, Lugano*, 2012, pp. 190–197.

[19] N. Cochrane, “Auditor-General slams Victorian smart meters - Networking - Technology - News - iTnews.com.au.” [Online]. Available: http://www.itnews.com.au/News/160398,auditor-general-slams-victorian-smart-meters.aspx. [Accessed: 04-Mar-2013].

[20] EurActiv, “Doubts cast over consumer benefits of smart meters.” [Online]. Available: http://www.euractiv.com/specialreport-access-energy/consumers-weak-smart-meter-roll-news-513445. [Accessed: 05-Mar-2013].

[21] M. Hickman, “Smart meters help to eliminate electricity-related theft, violence in Brazil.” [Online]. Available: http://www.mnn.com/your-home/at-home/blogs/smart-meters-help-to-eliminate-electricity-related-theft-violence-in-brazil. [Accessed: 05-Mar-2013].

[22] S. S. S. R. Depuru, L. Wang, and V. Devabhaktuni, “Electricity theft: Overview, issues, prevention and a smart meter based approach to control theft,” *Energy Policy*, vol. 39, no. 2, pp. 1007–1015, 2011.

[23] M. Anas, N. Javaid, A. Mahmood, S. M. Raza, U. Qasim, and Z. A. Khan, “Minimizing Electricity Theft using Smart Meters in AMI,” in *2012 Seventh Int. Conf.*

on P2P, Parallel, Grid, Cloud and Internet Computing, Victoria, BC, 2012, pp. 176–182.

[24] H. Khurana, M. Hadley, and D. A. Frincke, "Smart-grid security issues," *IEEE Security & Privacy Magazine*, vol. 8, no. 1, pp. 81–85, Jan. 2010.

[25] A. Molina-Markham, P. Shenoy, K. Fu, E. Cecchet, and D. Irwin, "Private memoirs of a smart meter," in *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building - BuildSys '10*, 2010, p. 61.

[26] "NIST Smart Grid Collaboration Wiki." [Online]. Available: http://collaborate.nist.gov/twiki-sggrid/bin/view/SmartGrid/WebHome. [Accessed: 04-Mar-2013].

[27] K. Samarakoon and J. Ekan, "Smart metering and self-healing of distribution networks," in *2010 IEEE International Conference on Sustainable Energy Technologies (ICSET), Kandy*, 2010, pp. 1–5.

[28] B. Reid, "Oncor Electric Delivery Smart Grid initiative," in *2009 62nd Annual Conference for Protective Relay Engineers, Austin, TX*, 2009, pp. 8–15.

[29] "European Smart Grids Technology Platform." [Online]. Available: ftp://ftp.cordis.europa.eu/pub/fp7/energy/docs/smartgrids_en.pdf. [Accessed: 04-Mar-2013].

[30] N. US Department of Commerce, "Smart Grid Collaboration Wiki." [Online]. Available: http://www.nist.gov/smartgrid/. [Accessed: 05-Mar-2013].

[31] B. News, "UK energy smart meter roll-out is outlined." [Online]. Available: http://news.bbc.co.uk/2/hi/business/8389880.stm. [Accessed: 04-Mar-2013].

[32] R. Yao and K. Steemers, "A method of formulating energy load profile for domestic buildings in the UK," *Energy and Buildings*, vol. 37, no. 6, pp. 663–671, Jun. 2005.

[33] C. Duarte, P. Delmar, K. W. Goossen, K. Barner, and E. Gomez-Luna, "Non-intrusive load monitoring based on switching voltage transients and wavelet transforms," in *2012 Future of Instrumentation International Workshop (FIIW) Proceedings, Gatlinburg, TN*, 2012, pp. 1–4.

[34] K. Suzuki, S. Inagaki, T. Suzuki, H. Nakamura, and K. Ito, "Nonintrusive appliance load monitoring based on integer programming," in *2008 SICE Annual Conference, Tokyo*, 2008, pp. 2742–2747.

[35] M. Zeifman and K. Roth, "Nonintrusive appliance load monitoring: Review and outlook," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 1, pp. 76–84, Feb. 2011.

[36] H. Y. Lam, G. S. K. Fung, and W. K. Lee, "A Novel Method to Construct Taxonomy Electrical Appliances Based on Load Signaturesof," *IEEE transactions on consumer electronics*, vol. 53 (2), p. 653, 2007.

[37] K. Samarakoon, J. Wu, J. Ekanayake, and N. Jenkins, "Use of delayed smart meter measurements for distribution state estimation," in *2011 IEEE Power and Energy Society General Meeting, San Diego, CA*, 2011, pp. 1–6.

[38] K. Samarakoon, J. Ekanayake, and N. Jenkins, "Investigation of Domestic Load Control to Provide Primary Frequency Response Using Smart Meters," *IEEE Transactions on Smart Grid*, vol. 3, no. 1, pp. 282–292, Mar. 2012.

[39] K. Samarakoon and J. Ekanayake, "Demand side primary frequency response support through smart meter control," in *Universities Power Eng. Conf. (UPEC), 2009 Proc. of the 44th Int., Glasgow*, 2009, pp. 1–5.

[40] V. Maurya, R. Jalan, and D. Pal, "A brief summery on voltage Sag & effects of Voltage Sag on Industrial Power Plant equipment," *J. of Environmental Sci., Comput. Sci. and Eng. and Technology*, vol. 2, no. 1, pp. 234–240, 2012.

[41] R. Naidoo and P. Pillay, "A New Method of Voltage Sag and Swell Detection," *IEEE Trans. on Power Delivery*, vol. 22, no. 2, pp. 1056–1063, Apr. 2007.

[42] N. S. Tunaboylu, E. R. Collins, and P. R. Chaney, "Voltage disturbance evaluation using the missing voltage technique," in *8th Int. Conf. on Harmonics and Quality of Power Proc.*, 1998, vol. 1, pp. 577–582.

[43] M. Ringwelski, C. Renner, A. Reinhardt, A. Weigel, and V. Turau, "The Hitchhiker's guide to choosing the compression algorithm for your smart meter data," in *2012 IEEE Int. Energy Conf. and Exhibition, Florence*, 2012, pp. 935–940.

[44] "DIN EN 62056-21:2003 Electricity metering," 2003.

[45] N. A. Kulatunga, S. Navaratne, J. Dole, C. Liyanagedera, and T. Martin, "Hardware development for Smart Meter based innovations," in *IEEE PES Innovative Smart Grid Technologies, Tianjin*, 2012, pp. 1–5.

[46] S.-W. Luan, J.-H. Teng, S.-Y. Chan, and L.-C. Hwang, "Development of a smart power meter for AMI based on ZigBee communication," in *2009 Int. Conf. on Power Electron. and Drive Systems, Taipei*, 2009, pp. 661–665.

[47] Telit, "GSM/GPRS," 04-Mar-2013. [Online]. Available: http://www.telit.com/en/products/gsm-gprs.php?p_ac=show&p=12. [Accessed: 06-Mar-2013].

[48] "IEC 61000-4-30," 2003.

[49] "NRS 048-3," 2003.

[50] Analog Devices, "ADE7854/ADE7858/ADE7868/ADE7878 Data Sheet Rev. F."

[51] Analog Devices, "ADE7878 Evaluation Board Users Guide UG-146 Rev. 0."

[52] Delvee, “First look at STM32F4 Discovery development kit.” [Online]. Available: http://www.delvee.com/articles/first-look-at-stm32f4-discovery-development-kit/. [Accessed: 29-Jul-2013].

[53] SD Association, “SD Memory Card Choices - SD Association.” [Online]. Available: https://www.sdcard.org/consumers/choices/. [Accessed: 06-Mar-2013].

[54] Sparkfun, “Breakout Board for microSD Transflash.” [Online]. Available: http://www.hobbytronics.co.uk/microsd-transflash-breakout. [Accessed: 29-Jul-2013].

[55] STMicroelectronics, “STM32 Reference Manual RM0090,” 2011.

[56] University of Cape Town. A. Patel, “SD Card Library Modification for STM32.” 2012.

[57] Chan and M. Thomas, “SD Card Library for STM32.” 2010.

[58] P. Minciunescu, “Application Note, Calibrating an ADE7878-based, 3-phase Energy Meter AN-1076 Rev. 0.”

[59] “IEC 60044-1.” 2003.

Appendix

A. Programs Used

The following programs were used for this project:

Atollic TrueSTUDIO for ARM Lite, http://www.atollic.com, v4.1.0

Atollic TrueSTUDIO for ARM Pro, http://www.atollic.com, v4.1.0

Microsoft Visual Studio Express 2012 for Windows Desktop C#,
http://www.microsoft.com/visualstudio/eng/downloads, 2012

All code and results in .csv and .dat format are included on the attached CD.

B. Microcontroller code

Explain the libraries needed to make this work. Standard set up for the discovery board but include according to peripherals used if using not discovery.

B.1 Full System Set Up

i. ADE7878_register.h

```
/*
 * ADE7878_registers.h
 * Taken from EVAL-ADE7878 sample code
 *
 *  Created on: Apr 24, 2013
 *      Author: RLRCHR002
 */

#ifndef ADE7878_REGISTERS_H_
#define ADE7878_REGISTERS_H_

#define   AIGAIN            0x4380
#define   AVGAIN            0x4381
#define   BIGAIN            0x4382
#define   BVGAIN            0x4383
#define   CIGAIN            0x4384
#define   CVGAIN            0x4385
#define   NIGAIN            0x4386
```

```
#define   AIRMSOS           0x4387
#define   AVRMSOS           0x4388
#define   BIRMSOS           0x4389
#define   BVRMSOS           0x438A
#define   CIRMSOS           0x438B
#define   CVRMSOS           0x438C
#define   NIRMSOS           0x438D
#define   AWATTOS           0x4392
#define   BWATTOS           0x4394
#define   CWATTOS           0x4396
#define   AVAROS            0x4398
#define   BVAROS            0x439A
#define   CVAROS            0x439C
#define   AFWATTOS          0x439E
#define   BFWATTOS          0x43A0
#define   CFWATTOS          0x43A2
#define   AFVAROS           0x43A4
#define   BFVAROS           0x43A6
#define   CFVAROS           0x43A8
#define   VATHR1            0x43A9
#define   VATHR0            0x43AA
#define   WTHR1             0x43AB
#define   WTHR0             0x43AC
#define   VARTHR1           0x43AD
#define   VARTHR0           0x43AE
#define   VANOLOAD          0x43B0
#define   APNOLOAD          0x43B1
#define   VARNOLOAD         0x43B2
#define   VLEVEL            0x43B3
#define   ISUMLVL           0x43B8
#define   ISUM              0x43BF
#define   AIRMS             0x43C0
#define   AVRMS             0x43C1
#define   BIRMS             0x43C2
#define   BVRMS             0x43C3
#define   CIRMS             0x43C4
#define   CVRMS             0x43C5
#define   NIRMS             0x43C6
#define   STOP              0xE203
#define   RUN               0xE228
#define   AWATTHR           0xE400
#define   BWATTHR           0xE401
#define   CWATTHR           0xE402
#define   AFWATTHR          0xE403
#define   BFWATTHR          0xE404
#define   CFWATTHR          0xE405
#define   AVARHR            0xE406
#define   BVARHR            0xE407
#define   CVARHR            0xE408
#define   AFVARHR           0xE409
#define   BFVARHR           0xE40A
#define   CFVARHR           0xE40B
#define   AVAHR             0xE40C
#define   BVAHR             0xE40D
#define   CVAHR             0xE40E
#define   IPEAK             0xE500
#define   VPEAK             0xE501
#define   STATUS0           0xE502
#define   STATUS1           0xE503
#define   AIRMS2            0xE504
#define   BIRMS2            0xE505
#define   CIRMS2            0xE506
```

```
#define   OILVL             0xE507
#define   OVLVL             0xE508
#define   SAGLVL            0xE509
#define   MASK0             0xE50A
#define   MASK1             0xE50B
#define   IAWV              0xE50C
#define   IBWV              0xE50D
#define   ICWV              0xE50E
#define   INWV              0xE50F
#define   VAWV              0xE510
#define   VBWV              0xE511
#define   VCWV              0xE512
#define   AWATT             0xE513
#define   BWATT             0xE514
#define   CWATT             0xE515
#define   AVAR              0xE516
#define   BVAR              0xE517
#define   CVAR              0xE518
#define   AVA               0xE519
#define   BVA               0xE51A
#define   CVA               0xE51B
#define   CHECKSUM          0xE51F
#define   VNOM              0xE520
#define   PHSTATUS          0xE600
#define   ANGLE0            0xE601
#define   ANGLE1            0xE602
#define   ANGLE2            0xE603
#define   PERIOD            0xE607
#define   PHNOLOAD          0xE608
#define   LINECYC           0xE60C
#define   ZXTOUT            0xE60D
#define   COMPMODE          0xE60E
#define   GAIN              0xE60F
#define   CFMODE            0xE610
#define   CF1DEN            0xE611
#define   CF2DEN            0xE612
#define   CF3DEN            0xE613
#define   APHCAL            0xE614
#define   BPHCAL            0xE615
#define   CPHCAL            0xE616
#define   PHSIGN            0xE617
#define   CONFIG            0xE618
#define   MMODE             0xE700
#define   ACCMODE           0xE701
#define   LCYCMODE          0xE702
#define   PEAKCYC           0xE703
#define   SAGCYC            0xE704
#define   CFCYC             0xE705
#define   HSDC_CFG          0xE706
#define   LPOILVL           0xEC00
#define   CONFIG2           0xEC01

#endif /* ADE7878_REGISTERS_H_ */
```

ii. Hsdc.h

```
/*
 * hsdc.h
 *
 * This function sets up the DMA and the SPI, the SPI is used as the STM side for the
HSDC protocol used
 *
 *  Created on: Apr 23, 2013
```

```
 *      Author: RLRCHR002
 */

#ifndef HSDC_H_
#define HSDC_H_

#define SPI_PORT                                        SPI1
#define SPI_PORT_CLOCK                                  RCC_APB2Periph_SPI1
#define SPI_PORT_CLOCK_INIT                             RCC_APB2PeriphClockCmd
#define SPI_SCK_PIN                                     GPIO_Pin_5
#define SPI_GPIO_PORT                                   GPIOA
#define SPI_GPIO_CLK                                    RCC_AHB1Periph_GPIOA
#define SPI_SCK_SOURCE                                  GPIO_PinSource5
#define SPI_SCK_AF                                      GPIO_AF_SPI1
#define SPI_MOSI_PIN                                    GPIO_Pin_7
#define SPI_MOSI_SOURCE                                 GPIO_PinSource7
#define SPI_MOSI_AF                                     GPIO_AF_SPI1
#define SPI_NSS_PIN                                     GPIO_Pin_4
#define SPI_NSS_SOURCE                                  GPIO_PinSource4
#define SPI_NSS_AF                                      GPIO_AF_SPI1

/* Definition for DMAx resources */
#define SPI_PORT_DMAx_CLK                               RCC_AHB1Periph_DMA2
#define SPI_PORT_RX_DMA_STREAM                          DMA2_Stream2
#define SPI_PORT_RX_DMA_CHANNEL                         DMA_Channel_3

#define SPI_PORT_DMA_RX_IRQn                            DMA2_Stream2_IRQn
#define SPI_PORT_DMA_RX_IRQHandler                      DMA2_Stream2_IRQHandler


void init_SPI1(int8_t* address0, int8_t* address1, int32_t buffer_size);
uint8_t SPI1_send(uint8_t data);
uint8_t SPI1_receive();

#endif /* HSDC_H_ */
```

iii. hsdc.c

```
/*
 * hsdc.c
 *
 * This function sets up the DMA and the SPI, the SPI is used as the STM side for the
 * HSDC protocol used
 *
 *  Created on: Apr 22, 2013
 *      Author: RLRCHR002
 */

#include <stm32f4xx.h>
#include <stm32f4xx_spi.h>
#include <hsdc.h>
#include <stm32f4xx_dma.h>

// this function initializes the SPI1 peripheral and the DMA
void init_SPI1(int8_t* address0, int8_t* address1, int32_t buffer_size) {

	uint32_t i;
	// set buffers to zero
	for (i = 0; i < buffer_size*4; i++) {
		address0[i] = 0;
		address1[i] = 0;
	}
```

```
GPIO_InitTypeDef GPIO_InitStruct;
SPI_InitTypeDef SPI_InitStruct;
DMA_InitTypeDef DMA_InitStructure;
NVIC_InitTypeDef NVIC_InitStructure;

// enable clock for used IO pins
RCC_AHB1PeriphClockCmd(SPI_GPIO_CLK, ENABLE);

//enable dma clk
RCC_AHB1PeriphClockCmd(SPI_PORT_DMAx_CLK, ENABLE);

// enable peripheral clock for SPI
SPI_PORT_CLOCK_INIT(SPI_PORT_CLOCK, ENABLE);

/* configure pins used by SPI1
 * PA4 = NSS
 * PA5 = SCK
 * PA7 = MOSI
 * - not needed so not used PA7 = MOSI
 */
GPIO_InitStruct.GPIO_Pin = SPI_MOSI_PIN | SPI_SCK_PIN | SPI_NSS_PIN;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(SPI_GPIO_PORT, &GPIO_InitStruct);

// connect SPI1 pins to SPI alternate function
GPIO_PinAFConfig(SPI_GPIO_PORT, SPI_NSS_SOURCE, SPI_NSS_AF );
GPIO_PinAFConfig(SPI_GPIO_PORT, SPI_SCK_SOURCE, SPI_SCK_AF );
GPIO_PinAFConfig(SPI_GPIO_PORT, SPI_MOSI_SOURCE, SPI_MOSI_AF );

//deinit spi first
SPI_I2S_DeInit(SPI_PORT );
//set spi to default settings
SPI_StructInit(&SPI_InitStruct);
//change those settings that are not the default
SPI_InitStruct.SPI_Direction = SPI_Direction_2Lines_RxOnly;
// 1 line receive only (or two lines receive only?)
SPI_InitStruct.SPI_Mode = SPI_Mode_Slave;     // transmit in slave mode
SPI_InitStruct.SPI_DataSize = SPI_DataSize_8b;
// one packet of data is 8 bits wide, will need two to receive 32 bits
SPI_InitStruct.SPI_CPOL = SPI_CPOL_High;        // clock is high when idle
SPI_InitStruct.SPI_CPHA = SPI_CPHA_2Edge;     // data sampled at second edge
SPI_InitStruct.SPI_NSS = SPI_NSS_Soft; // | SPI_NSSInternalSoft_Set;
// ? set the NSS management to internal and pull internal NSS high
SPI_InitStruct.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_16;
// ? SPI frequency is APB2 frequency / 64 which is? about 4MHz
SPI_InitStruct.SPI_FirstBit = SPI_FirstBit_MSB;
// data is transmitted MSB first
SPI_Init(SPI_PORT, &SPI_InitStruct);

//DMA double buffer mode

// start with a blank DMA configuration just to be sure
DMA_DeInit(SPI_PORT_RX_DMA_STREAM );
/*
 * Check if the DMA Stream is disabled before enabling it.
 * Note that this step is useful when the same Stream is used multiple times:
 * enabled, then disabled then re-enabled... In this case, the DMA Stream
 * disable
 * will be effective only at the end of the ongoing data transfer and it will
```

```
         * not be possible to re-configure it before making sure that the Enable bit
         * has been cleared by hardware. If the Stream is used only once, this step
         * might
         * be bypassed.
         */
        while (DMA_GetCmdStatus(SPI_PORT_RX_DMA_STREAM ) != DISABLE)
              ;
        // Configure DMA controller to manage RX DMA requests
        // first make sure we are using the default values
        DMA_StructInit(&DMA_InitStructure);
        // then set the ones that are not default values
        DMA_InitStructure.DMA_Channel = SPI_PORT_RX_DMA_CHANNEL;
        DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t) (SPI1_BASE + 0x0C);
        DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t) address0;
        DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;
        DMA_InitStructure.DMA_BufferSize = (buffer_size*4);    //NDT
        DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
        DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
        DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
        DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;      //word
        DMA_InitStructure.DMA_Mode = DMA_Mode_Circular; //circular
        DMA_InitStructure.DMA_Priority = DMA_Priority_High;
        DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Enable;  //enable
        DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_Full; //full
        DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_INC4;     //4
        DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;

        //set up double bufefr
        DMA_DoubleBufferModeConfig(SPI_PORT_RX_DMA_STREAM, address1, DMA_Memory_0);
        //enable double buffer
        DMA_DoubleBufferModeCmd(SPI_PORT_RX_DMA_STREAM, ENABLE);

        /* Enable the DMA Stream IRQ Channel */
        NVIC_InitStructure.NVIC_IRQChannel = SPI_PORT_DMA_RX_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStructure);

        DMA_Init(SPI_PORT_RX_DMA_STREAM, &DMA_InitStructure);

        SPI_I2S_DMACmd(SPI_PORT, SPI_I2S_DMAReq_Rx, ENABLE);
        // enable the receive buffer DMA

        DMA_ClearITPendingBit(SPI_PORT_RX_DMA_STREAM, DMA_IT_TCIF2 );

        /* Enable DMA Stream Transfer Complete interrupt */
        DMA_ITConfig(SPI_PORT_RX_DMA_STREAM, DMA_IT_TC, ENABLE);

        SPI_Cmd(SPI_PORT, ENABLE); // enable SPI1
        DMA_Cmd(SPI_PORT_RX_DMA_STREAM, ENABLE);

}


//these two functions were used in testing SPI but are not currently used

/* This funtion is used to transmit and receive data
 * with SPI1
 *                  data --> data to be transmitted
 *                  returns received value
 */
```

```
uint8_t SPI1_send(uint8_t data) {

	SPI1 ->DR = data; // write data to be transmitted to the SPI data register
	while (!(SPI1 ->SR & SPI_I2S_FLAG_TXE ))
		; // wait until transmit complete
	while (!(SPI1 ->SR & SPI_I2S_FLAG_RXNE ))
		; // wait until receive complete
	//while( SPI1->SR & SPI_I2S_FLAG_BSY ); // wait until SPI is not busy anymore
	return SPI1 ->DR; // return received data from SPI data register
}

uint8_t SPI1_receive() {

	while (!(SPI1 ->SR & SPI_I2S_FLAG_RXNE ))
		; // wait until receive complete
	//while( !(SPI1->SR & SPI_I2S_FLAG_BSY) ) {return;} // ensure SPI still busy
	return SPI1 ->DR; // return received data from SPI data register
}
```

iv. I2c.h

```
/*
 * i2c.h
 *
 * For I2C communication between the STM32F4 and the ADE7878
 * It is a standardised protocol explained in detail in the data sheet
 *
 *  Created on: Apr 19, 2013
 *      Author: RLRCHR002
 */

#ifndef I2C_H_
#define I2C_H_

#define SLAVE_ADDRESS 0x70 // the slave address, the ADE7878-EVAL
/* Private function prototypes */

// The following three functions are specific to the ADE7878
void write_ADE78_I2C(I2C_TypeDef* I2Cx, uint16_t register_address, uint32_t data,
		uint8_t communication_bit_length);
void write_last_ADE78_I2C(I2C_TypeDef* I2Cx, uint16_t register_address, uint32_t data,
		uint8_t communication_bit_length);
void read_ADE78_I2C(I2C_TypeDef* I2Cx, uint16_t register_address, uint32_t*
received_data,
		uint8_t communication_bit_length);

// The following functions are specific to the STM and follow the standard protocol
void init_I2C1(void);
void I2C_start(I2C_TypeDef* I2Cx, uint8_t address, uint8_t direction);
void I2C_start_read(I2C_TypeDef* I2Cx, uint8_t address, uint8_t direction);
void I2C_write(I2C_TypeDef* I2Cx, uint8_t data);
uint8_t I2C_read_ack(I2C_TypeDef* I2Cx);
uint8_t I2C_read_nack(I2C_TypeDef* I2Cx);
void I2C_stop(I2C_TypeDef* I2Cx);

#endif /* I2C_H_ */
```

v. i2c.h

```
/*
 * i2c.c
 *
```

```
 * this function has the I2C protocol in it and is used to set up the i2c and to read
and write with it
 * It is a standardised protocol explained in detail in the data sheet
 *
 *  Created on: Apr 19, 2013
 *      Author: RLRCHR002
 */

#include <stm32f4xx_i2c.h>
#include "stm32f4xx.h"
#include <i2c.h>


// To write to a register/address using the I2C
void write_ADE78_I2C(I2C_TypeDef* I2Cx, uint16_t register_address, uint32_t data,
              uint8_t communication_bit_length) {
       I2C_start(I2Cx, SLAVE_ADDRESS, I2C_Direction_Transmitter );
       // start a transmission in Master transmitter mode

       uint8_t address_msb = register_address >> 8;
       uint8_t address_lsb = register_address;

       I2C_write(I2Cx, address_msb);
       I2C_write(I2Cx, address_lsb);

       uint8_t count = communication_bit_length / 8;

       int i;

       for (i = (count - 1); i >-1; i--) {
              uint8_t temp = data >> (8 * i);
              I2C_write(I2Cx, temp);
       }

       I2C_stop(I2C1 ); // stop the transmission
}

// to write to the final register when using an ADE and performing multiple writes at
// once see data sheet about the two part pipeline register buffer
void write_last_ADE78_I2C(I2C_TypeDef* I2Cx, uint16_t register_address, uint32_t data,
              uint8_t communication_bit_length) {

       I2C_start(I2Cx, SLAVE_ADDRESS, I2C_Direction_Transmitter );
       // start a transmission in Master transmitter mode

       uint8_t address_lsb = register_address;
       uint8_t address_msb = register_address >> 8;

       I2C_write(I2Cx, address_msb);
       I2C_write(I2Cx, address_lsb);

       uint8_t count = communication_bit_length / 8;
       int i;

       for (i = (count - 1); i >-1; i--) {
              uint8_t temp = data >> (8 * i);
              I2C_write(I2Cx, temp);
       }

       I2C_write(I2Cx, 0x00);
       I2C_write(I2Cx, 0x00);
```

```
	//This means that when a single register needs to be initialized, two more
	// writes are required to ensure the
	//value has been written into RAM, and if two or more registers need to be
	// initialized, the last register
	//must be written two more times to ensure the value has been written into RAM.

	I2C_stop(I2Cx ); // stop the transmission
}

// read a register from ADE7878
void read_ADE78_I2C(I2C_TypeDef* I2Cx, uint16_t register_address, uint32_t*
received_data,
		uint8_t communication_bit_length) {

	uint8_t address_lsb, address_msb, temp[4];
	int i;

	for (i = 0; i <4; i++) {
			temp[i] = 0;
		}

	address_lsb = register_address;
	address_msb = register_address >> 8;

	I2C_start(I2Cx, SLAVE_ADDRESS, I2C_Direction_Transmitter );
	// start a transmission in Master transmitter mode

	I2C_write(I2Cx, address_msb);
	I2C_write(I2Cx, address_lsb);

	I2C_start_read(I2Cx, SLAVE_ADDRESS, I2C_Direction_Receiver );
	// start a transmission in Master receiver mode

	uint8_t count = communication_bit_length / 8;

	for (i = (count - 1); i > 0; i--) {
		temp[i] = I2C_read_ack(I2C1 );
	}

	temp[0] = I2C_read_nack(I2C1 );

	I2C_stop(I2Cx ); // stop the transmission

	*received_data = ( (temp[3] << 24) +  (temp[2] << 16) +  (temp[1] << 8) +
	(temp[0]) );

}

// Initialise the I2C communications
void init_I2C1(void) {

	GPIO_InitTypeDef GPIO_InitStruct;
	I2C_InitTypeDef I2C_InitStruct;

	// enable APB1 peripheral clock for I2C1
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
	// enable clock for SCL and SDA pins
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

	/* setup SCL and SDA pins
	 * You can connect I2C1 to two different
	 * pairs of pins:
```

```
	 * 1. SCL on PB6 and SDA on PB7
	 * 2. SCL on PB8 and SDA on PB9
	 */
	GPIO_InitStruct.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
	// we are going to use PB6 and PB7
	GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
	// set pins to alternate function
	GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;		// set GPIO speed
	GPIO_InitStruct.GPIO_OType = GPIO_OType_OD;		// set output to open
	// drain --> the line has to be only pulled low, not driven high
	GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
	// enable pull up resistors
	GPIO_Init(GPIOB, &GPIO_InitStruct);
	// init GPIOB

	// Connect I2C1 pins to AF
	GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_I2C1 );	// SCL
	GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_I2C1 );	// SDA

	// configure I2C1
	I2C_InitStruct.I2C_ClockSpeed = 100000;
		// 100kHz
	I2C_InitStruct.I2C_Mode = I2C_Mode_I2C;
			// I2C mode
	I2C_InitStruct.I2C_DutyCycle = I2C_DutyCycle_2;
		// 50% duty cycle --> standard
	I2C_InitStruct.I2C_OwnAddress1 = 0x00;
			// own address, not relevant in master mode
	I2C_InitStruct.I2C_Ack = I2C_Ack_Disable;
		// disable acknowledge when reading (can be changed later on)
	I2C_InitStruct.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
	// set address length to 7 bit addresses
	I2C_Init(I2C1, &I2C_InitStruct);
			// init I2C1

	// enable I2C1
	I2C_Cmd(I2C1, ENABLE);
}

/* This function issues a start condition and
 * transmits the slave address + R/W bit
 *
 * Parameters:
 *		I2Cx --> the I2C peripheral e.g. I2C1
 *		address --> the 7 bit slave address
 *		direction --> the tranmission direction can be:
 *				I2C_Direction_Tranmitter for Master transmitter mode
 *					I2C_Direction_Receiver for Master receiver
 */
void I2C_start(I2C_TypeDef* I2Cx, uint8_t address, uint8_t direction) {
	// wait until I2C1 is not busy anymore
	while (I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY ))
		;

	// Send I2C1 START condition
	I2C_GenerateSTART(I2Cx, ENABLE);

	// wait for I2C1 EV5 --> Slave has acknowledged start condition
	while (!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT ))
		;

	// Send slave Address for write
```

```
	I2C_Send7bitAddress(I2Cx, address, direction);

	/* wait for I2C1 EV6, check if
	 * either Slave has acknowledged Master transmitter or
	 * Master receiver mode, depending on the transmission
	 * direction
	 */
	if (direction == I2C_Direction_Transmitter ) {
		while (!I2C_CheckEvent(I2Cx,
				I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED ))
			;
	} else if (direction == I2C_Direction_Receiver ) {
		while (!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED ))
			;
	}
}

void I2C_start_read(I2C_TypeDef* I2Cx, uint8_t address, uint8_t direction) {
	// wait until I2C1 is not busy anymore
	//while(I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY));

	// Send I2C1 START condition
	I2C_GenerateSTART(I2Cx, ENABLE);

	// wait for I2C1 EV5 --> Slave has acknowledged start condition
	while (!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT ))
		;

	// Send slave Address for write
	I2C_Send7bitAddress(I2Cx, address, direction);

	/* wait for I2C1 EV6, check if
	 * either Slave has acknowledged Master transmitter or
	 * Master receiver mode, depending on the transmission
	 * direction
	 */
	/*if(direction == I2C_Direction_Transmitter){
	 while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
	 }
	 else if(direction == I2C_Direction_Receiver){
	 while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
	 }*/
}

/* This function transmits one byte to the slave device
 * Parameters:
 *		I2Cx --> the I2C peripheral e.g. I2C1
 *		data --> the data byte to be transmitted
 */
void I2C_write(I2C_TypeDef* I2Cx, uint8_t data) {
	I2C_SendData(I2Cx, data);
	// wait for I2C1 EV8_2 --> byte has been transmitted
	while (!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED ))
		;
}

/* This function reads one byte from the slave device
 * and acknowledges the byte (requests another byte)
 */
uint8_t I2C_read_ack(I2C_TypeDef* I2Cx) {
	// enable acknowledge of recieved data
	I2C_AcknowledgeConfig(I2Cx, ENABLE);
```

```
	// wait until one byte has been received
	while (!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_RECEIVED ))
		;
	// read data from I2C data register and return data byte
	uint8_t data = I2C_ReceiveData(I2Cx);
	return data;
}

/* This function reads one byte from the slave device
 * and doesn't acknowledge the recieved data
 */
uint8_t I2C_read_nack(I2C_TypeDef* I2Cx) {
	// disable acknowledge of received data
	I2C_AcknowledgeConfig(I2Cx, DISABLE);
	// wait until one byte has been received
	while (!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_RECEIVED ))
		;
	// read data from I2C data register and return data byte
	uint8_t data = I2C_ReceiveData(I2Cx);
	return data;
}

/* This funtion issues a stop condition and therefore
 * releases the bus
 */
void I2C_stop(I2C_TypeDef* I2Cx) {
	// Send I2C1 STOP Condition
	I2C_GenerateSTOP(I2Cx, ENABLE);
}
```

vi. Main.h

```
/*
 *	main.h
 *
 *	For all arrays and definitions used by more than one .c file
 *
 *  Created on: May 22, 2013
 *      Author: Christine Martindale
 *      RLRCHR002
 *      For Masters dissertation
 */

#ifndef MAIN_H_
#define MAIN_H_

#define frequency 2000
// in Hz
#define HSDCdata 7
// 7 = instant only, 16 = instant and conventional power
#define noOfCycles 10
// this is the number of cycles to store in each .dat file

#define numberOfDataPointsPerCycle (frequency / 50)		//40
#define numberOfLinesSkipped (8000 / frequency)
	// 4 lines skipped when storing and using data from HSDC

		// 8000 because the EVAL records data at 8kHz

#define BUFFER_SIZE_IN_WORDS (noOfCycles * numberOfDataPointsPerCycle *
numberOfLinesSkipped * HSDCdata) //11200
// The line below is added when wanting to visually check that each file follows on //
from the last with no data loss - it is 10.5 cycles instead of 10
```

```
            // + numberOfLinesSkipped * HSDCdata *numberOfDataPointsPerCycle/ 2)
                  // ensure that above is within ram

int8_t HSDCBuffer0[BUFFER_SIZE_IN_WORDS * 4];
// Buffers used by the DMA
int8_t HSDCBuffer1[BUFFER_SIZE_IN_WORDS * 4];
int8_t TempWordBuffer0[4];
// Array used to change little endian to big endian byte order
int8_t TempWordBuffer1[4];

double TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS) / numberOfLinesSkipped)
            + ((noOfCycles / 2) * HSDCdata * 2)];
// Array used to stored data to be written to a file on the SD card - instantaneous //
and results of power calculations
                  //5 rms rows and 5 power rows
// TempHSDCBuffer1 and TempHSDCBuffer0 used before but using just one TempHSDCBuffer
// uses less RAM
//double TempHSDCBuffer1[((BUFFER_SIZE_IN_WORDS)  / numberOfLinesSkipped )+
//((noOfCycles / 2) * HSDCdata * 2)];

#endif /* MAIN_H_ */
```

vii. main.c

```
/**
 ******************************************************************************
 **
 **  File        : main.c
 **
 **  Description: This sets up the communications and begins the recording of data,
 **           the recording of data is only stopped from within the interupt routine
 **
 **  Hardware : STM32F4-discovery evaluation board (STMicroelectronics)
 **           ADE7878-EVAL Energy measurement chip evaluation board (analog components)
 **
 **  Environment : Atollic TrueSTUDIO(R)
 **                STMicroelectronics STM32F4xx Standard Peripherals Library
 **
 **    Author        : Christine Martindale 10.04.2013
 **
 ******************************************************************************
 */

/* Includes */
#include "stm32f4xx.h"
#include "stm32f4_discovery.h"
#include "i2c.h"
#include "hsdc.h"
#include "ADE7878_registers.h"
#include <stdio.h>
#include "main.h"
#include "ff.h"

// Set up for the GPIO pins that are used
GPIO_InitTypeDef GPIO_InitStructure;

#define GPIO_PIN_RESETB                  GPIO_Pin_9
#define GPIO_PIN_PSM0                    GPIO_Pin_13
#define GPIO_PIN_PSM1                    GPIO_Pin_14
#define GPIO_PORT_PSM_RESET_IRQ GPIOD
#define GPIO_PIN_IRQ0                    GPIO_Pin_11
#define GPIO_PIN_IRQ1                    GPIO_Pin_10
```

```
int main(void) {

	// Initialise the GPIO ports

	/* GPIOD Periph clock enable */
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

	/* Configure psm and reset */
	GPIO_InitStructure.GPIO_Pin = GPIO_PIN_PSM0 | GPIO_PIN_PSM1
			| GPIO_PIN_RESETB;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_Init(GPIO_PORT_PSM_RESET_IRQ, &GPIO_InitStructure);

	// set IRQ pins with pull up resistors
	GPIO_InitStructure.GPIO_Pin = GPIO_PIN_IRQ0 | GPIO_PIN_IRQ1;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
	GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; // push pull or open drain
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP; // pull up
	GPIO_Init(GPIO_PORT_PSM_RESET_IRQ, &GPIO_InitStructure);

	// to store data for testing purposes
	uint32_t enable, HSDC_CFG_register_value, locked, original,
			run, temp, data_test_status1;
	//uint32_t CHECKSUM_register_value, AVAROS_register_value_original,
	//AVAROS_register_value_changed,
	//MASK0_register_value, MASK1_register_value,
	//BIGAIN_register_value, HSDC_CFG_register_value,enable, enable2, run, temp;
	//uint8_t data0, data1, data2;
	//data0 = 1; data1 = 1; data2 = 1;

	// set reset high, psm0 high and psm1 low
	GPIO_WriteBit(GPIO_PORT_PSM_RESET_IRQ, GPIO_PIN_RESETB, Bit_SET);
	GPIO_WriteBit(GPIO_PORT_PSM_RESET_IRQ, GPIO_PIN_PSM0, Bit_SET);
	GPIO_WriteBit(GPIO_PORT_PSM_RESET_IRQ, GPIO_PIN_PSM1, Bit_RESET);

	uint32_t i;
	for (i = 0; i <( BUFFER_SIZE_IN_WORDS/8 + 70); i++) {
			TempHSDCBuffer[i] = 0;
			TempHSDCBuffer[i] = 0;

			}

	init_I2C1(); // initialize I2C peripheral

	// break point here for debugging

	uint32_t data = 0;

	// wait for interrupt to say the normal power mode is set up correctly:
	uint8_t IRQ1B = GPIO_ReadInputDataBit(GPIO_PORT_PSM_RESET_IRQ,
			GPIO_PIN_IRQ1 );
	//uint8_t IRQ0B = GPIO_ReadInputDataBit(GPIO_PORT_PSM_RESET_IRQ,
	//GPIO_PIN_IRQ0);
	while (IRQ1B == 1) {
		IRQ1B = GPIO_ReadInputDataBit(GPIO_PORT_PSM_RESET_IRQ, GPIO_PIN_IRQ1 );
	} // only move on when irq1 has been pulled low

	//wait for interrupt - then read status1 register:
```

```
read_ADE78_I2C(I2C1, STATUS1, &data, 32);

//TODO:  test this section that is commented out
//Wait until the reset is done and PSM0 has finished initializing by waiting
//until RSDONE is high
// RSDONE is bit 15 of the status1 register
//uint16_t t = 1 << (15); //test line to ensure bit and with but 15
//uint32_t a = (data) & (1 << (15));
if (((data) & (1 << (15))) == 0x8000) {
 write_ADE78_I2C(I2C1, STATUS1, data, 32);// write back same to clear flag
 // for debug purposes read the STATUS1 again and irq1 again:
 read_ADE78_I2C(I2C1, STATUS1, &data_test_status1, 32);
 IRQ1B = GPIO_ReadInputDataBit(GPIO_PORT_PSM_RESET_IRQ, GPIO_PIN_IRQ1);
 }

//If I2C is the active serial port, Bit 1 (I2C_LOCK) of the CONFIG2 register
//must be set to 1 to lock it in
//CONFIG2 Register (Address 0xEC01) bit 1 (I2C_LOCK)
// set (I2C_LOCK) to 1 in order to lock into I2C mode
read_ADE78_I2C(I2C1, CONFIG2, &original, 8);
//to lock
write_ADE78_I2C(I2C1, CONFIG2, 0x02, 8);
// check locked/ read status reg - for debugging:
read_ADE78_I2C(I2C1, CONFIG2, &locked, 8);

// ***** NB ***** if problems reading and writing using i2c, then use the
//write_last_ADE78_I2C function
//Assumed only needs to be done once in the set up of the ADE7878

// Initialise the SPI - found in hsdc.c
init_SPI1(&HSDCBuffer0[0], &HSDCBuffer1[0], BUFFER_SIZE_IN_WORDS);

// let the dsp run begin
// the DSP is however not really used at the moment because it is set up to
//read only instantaneous values, however when set up to also read conventional
//power then this is needed
write_ADE78_I2C(I2C1, RUN, 0x0001, 16);
read_ADE78_I2C(I2C1, RUN, &run, 16);
//to check i2c is working
// Read check sum register - a default value
// Expecting 0x33666787
// address of checksum 0xE51F
//read_ADE78_I2C(I2C1, 0xE51F, &CHECKSUM_register_value, 32);
uint32_t AVRMS_reg =0, BVRMS_reg =0, CVRMS_reg =0;
read_ADE78_I2C(I2C1, AVRMS, &AVRMS_reg, 32);
read_ADE78_I2C(I2C1, BVRMS, &BVRMS_reg, 32);
read_ADE78_I2C(I2C1, CVRMS, &CVRMS_reg, 32);
//write_ADE78_I2C(I2C1, 0x4398, 0x00f31145, 32);
//read_ADE78_I2C(I2C1, 0x4398, &AVAROS_register_value_changed, 32);

/***********
 -DMA must be set as as a slave and as below: SCLK=8MHz, SCLK active low
 (CPOL=1, CPHA=1)
 -HSDC must be set for:
 -HCLK =0 8MHz clk
 -HSIZE=1: HSDC transmits in 8 bit packages - n/a
 -HGAP=0: no gap
 -HXFER=01: HSDC transmits only instant therefore 7
 -HSAPOL=0: HSACTIVE is active low
 -This means HSDC_CFG=0x0B
 *************/
```

```
	write_ADE78_I2C(I2C1, HSDC_CFG, 0x0B, 8);	// write to set up hsdc
	//check reg value correct
	read_ADE78_I2C(I2C1, HSDC_CFG, &HSDC_CFG_register_value, 8);

	// for testing/debugging:
	//read mask0 and mask1
	//read_ADE78_I2C(I2C1, MASK0, &MASK0_register_value, 32);
	//read_ADE78_I2C(I2C1, MASK1, &MASK1_register_value, 32);

	// read current config value, add the enable bit and then enable hsdc
	read_ADE78_I2C(I2C1, CONFIG, &enable, 16);
	temp = 0x0040 | enable;
	write_ADE78_I2C(I2C1, CONFIG, temp, 16);
	//read_ADE78_I2C(I2C1, CONFIG, &enable2, 16);

	// read the data being sent

	while (1) {
	}

	// need to show when ended and end hsdc

	//write stop for hsdc - need to also stop DMA first!
	//write_ADE78_I2C(I2C1, CONFIG, 0x0000, 16);
	//read_ADE78_I2C(I2C1, CONFIG, &run, 16);

	//return 0;

}



/*
 * Callback used by stm32f4_discovery_audio_codec.c.
 * Refer to stm32f4_discovery_audio_codec.h for more info.
 */
void EVAL_AUDIO_TransferComplete_CallBack(uint32_t pBuffer, uint32_t Size) {
	/* TODO, implement your code here */
	return;
}

/*
 * Callback used by stm324xg_eval_audio_codec.c.
 * Refer to stm324xg_eval_audio_codec.h for more info.
 */
uint16_t EVAL_AUDIO_GetSampleCallBack(void) {
	/* TODO, implement your code here */
	return -1;
}
```

viii. Stm32f4xx_it.c

```
/**
 ******************************************************************************
 * @file    Project/STM32F4xx_StdPeriph_Template/stm32f4xx_it.c
 * @author  MCD Application Team
 * @version V1.1.0
 * @date    18-January-2013
 * @brief   Main Interrupt Service Routines.
 *          This file provides template for all exceptions handler and
 *          peripherals interrupt service routine.
 ******************************************************************************
 * @attention
```

```
 *
 * <h2><center>&copy; COPYRIGHT 2013 STMicroelectronics</center></h2>
 *
 * Licensed under MCD-ST Liberty SW License Agreement V2, (the "License");
 * You may not use this file except in compliance with the License.
 * You may obtain a copy of the License at:
 *
 *        http://www.st.com/software_license_agreement_liberty_v2
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 ******************************************************************************
 ******************************************************************************
 ******************************************************************************
 ******************************************************************************
 */

/*
 * Adaptations to this file done by:
 * Chrstine Martindale
 * RLRCHR002
 */

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_it.h"
#include "hsdc.h"
#include "main.h"
#include "ff.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// Initialise counters:
uint32_t j, z;
uint32_t tempcounter = 0, rowcounter;
uint8_t condition, endRead;
uint32_t counter = 0;
int SDcounter = 0;

// Initialise what is needed for the SD card write
FATFS fs[1]; /* Work area (file system object) for logical drives */
FIL fdat; /* file objects */
FRESULT res; /* FatFs function common result code */
UINT bw = 0, br = 0; /* File write count */
char filename[12];

// Initialise conversion factors as well as data arrays
double accumulatorSquaredV1 = 0, accumulatorSquaredV2 = 0,
              accumulatorSquaredV3 = 0, accumulatorSquaredI1 = 0,
              accumulatorSquaredI2 = 0, accumulatorSquaredI3 = 0,
              accumulatorSquaredIN = 0, accumulatorNewP = 0, accumulatorNewS = 0,
              accumulatorNewPF = 0;

double conversionFactorBinaryToVolt = 0.95520 / 10000;
double conversionFactorBinaryToAmp = 0.33804 / 100000;
double conversionFactorBinaryToP = 0.95520 / 10000 * 0.33804 / 100000;
double conversionFactorBinaryToS = 0.95520 / 10000 * 0.33804 / 100000;
double conversionFactorBinaryToQ = 0.95520 / 10000 * 0.33804 / 100000;
```

```
double conversionFactorBinaryToPF = 1;

double conversionFactorV1 = 1;
double conversionFactorV2 = 0.99841;
double conversionFactorV3 = 0.97979;

double conversionFactorI1 = 1;
double conversionFactorI2 = 0.98906;
double conversionFactorI3 = 0.96789;
double conversionFactorIN = 1;
float neutralWireResistance = 2,

       phaseWireResistance = 4;

// Only needed if using ADE7878 convenstional power calcs:

/*double conversionFactorAVA = 1;
double conversionFactorBVA = 0.98532;
double conversionFactorCVA = 0.96491;

double conversionFactorAWATT = 1;
double conversionFactorBWATT = 0.98351;
double conversionFactorCWATT = 0.95852;

double conversionFactorAVAR = 1;
double conversionFactorBVAR = 1;
double conversionFactorCVAR = 1; */

double instantIA = 0, instantIB = 0, instantIC = 0, instantIN = 0,
              instantVA = 0, instantVB = 0, instantVC = 0;

int accumulatorCounterNewP = 0, accumulatorCounterV1 = 0, accumulatorCounterV2 =
              0, accumulatorCounterV3 = 0, accumulatorCounterVN = 0,
              accumulatorCounterI1 = 0, accumulatorCounterI2 = 0,
              accumulatorCounterI3 = 0, accumulatorCounterIN = 0;

double averageNewP = 0, averageNewS = 0, averageNewQ = 0, averageNewPF = 0;
int firstrow = 0;


// General standard interupts:

/** @addtogroup Template_Project
 * @{
 */

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/

/******************************************************************************/
/*            Cortex-M4 Processor Exceptions Handlers                         */
/******************************************************************************/

/**
 * @brief   This function handles NMI exception.
 * @param  None
 * @retval None
 */
```

```
void NMI_Handler(void) {
}

/**
 * @brief  This function handles Hard Fault exception.
 * @param  None
 * @retval None
 */
void HardFault_Handler(void) {
	/* Go to infinite loop when Hard Fault exception occurs */
	while (1) {
	}
}

/**
 * @brief  This function handles Memory Manage exception.
 * @param  None
 * @retval None
 */
void MemManage_Handler(void) {
	/* Go to infinite loop when Memory Manage exception occurs */
	while (1) {
	}
}

/**
 * @brief  This function handles Bus Fault exception.
 * @param  None
 * @retval None
 */
void BusFault_Handler(void) {
	/* Go to infinite loop when Bus Fault exception occurs */
	while (1) {
	}
}

/**
 * @brief  This function handles Usage Fault exception.
 * @param  None
 * @retval None
 */
void UsageFault_Handler(void) {
	/* Go to infinite loop when Usage Fault exception occurs */
	while (1) {
	}
}

/**
 * @brief  This function handles SVCall exception.
 * @param  None
 * @retval None
 */
void SVC_Handler(void) {
}

/**
 * @brief  This function handles Debug Monitor exception.
 * @param  None
 * @retval None
 */
void DebugMon_Handler(void) {
}
```

```
/**
 * @brief  This function handles PendSVC exception.
 * @param  None
 * @retval None
 */
void PendSV_Handler(void) {
}

/**
 * @brief  This function handles SysTick Handler.
 * @param  None
 * @retval None
 */
void SysTick_Handler(void) {
      /*  TimingDelay_Decrement(); */
}

/******************************************************************************/
/*                 STM32F4xx Peripherals Interrupt Handlers                   */
/*  Add here the Interrupt Handler for the used peripheral(s) (PPP), for the  */
/*  available peripheral interrupt handler's name please refer to the startup */
/*  file (startup_stm32f40xx.s/startup_stm32f427x.s).                         */
/******************************************************************************/

/**
 * @brief  This function handles PPP interrupt request.
 * @param  None
 * @retval None
 */
/*void PPP_IRQHandler(void)
 {
 }*/

/**
 * @}
 */

// Peripheral interrupts written by Christine Martindale

void SPI_PORT_DMA_RX_IRQHandler(void) {

  //this triggers when one buffer is full - so when copying out ensure reading
  //from buffer that is not in use by DMA


  /* Test on DMA Stream Transfer Complete interrupt */
  if (DMA_GetITStatus(SPI_PORT_RX_DMA_STREAM, DMA_IT_TCIF2 )) {

    if (endRead == 0) {// if not already collected the needed amount of data

       //if in one buffer then copy from other buffer to a new array
       if (!DMA_GetCurrentMemoryTarget(SPI_PORT_RX_DMA_STREAM )) {

             // initialise counters to zero
             tempcounter = 0;
             rowcounter = 0;
             // loop through the number of bytes present in the buffer:
             for (j = 0; j < (BUFFER_SIZE_IN_WORDS * 4); j++) {
             // only take the 5th set of data - to reduce down
             //from 8kHz to 2 k Hz:
             if (((j) % (numberOfLinesSkipped * 4 * HSDCdata)) == 0) {
```

```
                // Loop through each set of data consisting of 7 words and then
                //once more
                // The final loop os to perform the new power calculations:
                for (z = 0; z < (HSDCdata + 1); z++) {

if (z < HSDCdata) {// if dealing with one of the 7 words
   //then convert it to the correct byte order and save as a
   //double in the buffer to be written to SD card
        char* byte0address = (char*) (void*) &HSDCBuffer1[j + z];
        /*address of first byte*/
        char* byte1address = (char*) (void*) &HSDCBuffer1[j + z + 1];
        char* byte2address = (char*) (void*) &HSDCBuffer1[j + z + 2];
        char* byte3address = (char*) (void*) &HSDCBuffer1[j + z + 3];
        /*address of last byte*/

        TempWordBuffer0[0] = *byte3address; //LSB
        TempWordBuffer0[1] = *byte2address;
        TempWordBuffer0[2] = *byte1address;
        TempWordBuffer0[3] = *byte0address; //MSB

        TempHSDCBuffer[tempcounter + z] =
              (double) *(int32_t *) &TempWordBuffer0[0];
}

j += 3;// part of the byte swap

switch (z) {

        case 0: {                               // IA

              instantIA = TempHSDCBuffer[tempcounter + z];
              // save the instant value to the buffer to be //written to SD card

              if (accumulatorCounterI1< ((numberOfDataPointsPerCycle * 2) - 1))
              // Loop through two cycles and accumulate the values
              {
                    accumulatorSquaredI1 = accumulatorSquaredI1
                          + pow( TempHSDCBuffer[tempcounter + z], 2);
                    accumulatorCounterI1++;

              } else {
                    accumulatorSquaredI1 = accumulatorSquaredI1
                          + pow( TempHSDCBuffer[tempcounter  + z], 2);
                    // Once accumulated the instant values for two cycles then
                    //add the conversion factors, average and square root

                    double answer =(sqrt(      accumulatorSquaredI1
                          / (numberOfDataPointsPerCycle * 2)))
                          * conversionFactorI1
                          * conversionFactorBinaryToAmp;

                    //Save the RMS value at the end of the temp //array

                    TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)
                          /numberOfLinesSkipped) + z
                          + rowcounter * HSDCdata] = answer;
                    // reset the counters to zero
                    accumulatorSquaredI1 = 0;
                    accumulatorCounterI1 = 0;
              }
              break;
```

```
}

case 1:                                     //VA
{
       instantVA = TempHSDCBuffer[tempcounter + z];

       if (accumulatorCounterV1 < ((numberOfDataPointsPerCycle * 2) - 1))
       {
              accumulatorSquaredV1 = accumulatorSquaredV1
                     + pow( TempHSDCBuffer[tempcounter + z],2);
              accumulatorCounterV1++;

       } else {
              accumulatorSquaredV1 = accumulatorSquaredV1
                     + pow(TempHSDCBuffer[tempcounter+z], 2);

              double answer =(sqrt(       accumulatorSquaredV1
                     / (numberOfDataPointsPerCycle* 2)))
                     * conversionFactorV1 * conversionFactorBinaryToVolt;
              TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)
                     / numberOfLinesSkipped) + z
                     + rowcounter * HSDCdata] = answer;

              accumulatorSquaredV1 = 0;
              accumulatorCounterV1 = 0;
       }

       break;
}

case 2:                                     //IB
{
       instantIB = TempHSDCBuffer[tempcounter + z];

       if (accumulatorCounterI2 < ((numberOfDataPointsPerCycle * 2) - 1))
       {
              accumulatorSquaredI2 = accumulatorSquaredI2
                     + pow( TempHSDCBuffer[tempcounter + z],2);
              accumulatorCounterI2++;

       } else {
              accumulatorSquaredI2 = accumulatorSquaredI2
                     + pow( TempHSDCBuffer[tempcounter  + z], 2);

              double answer =(sqrt(accumulatorSquaredI2
                     / (numberOfDataPointsPerCycle* 2)))
                     * conversionFactorI2
                     * conversionFactorBinaryToAmp;


              TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)
                     / numberOfLinesSkipped) + z
                     + rowcounter * HSDCdata] = answer;

              accumulatorSquaredI2 = 0;
              accumulatorCounterI2 = 0;
       }
       break;
}

case 3:                             //VB
{
```

```
		instantVB = TempHSDCBuffer[tempcounter + z];

		if (accumulatorCounterV2< ((numberOfDataPointsPerCycle * 2) - 1))
		{
			accumulatorSquaredV2 = accumulatorSquaredV2
				+ pow( TempHSDCBuffer[tempcounter + z],2);
			accumulatorCounterV2++;
		} else {
			accumulatorSquaredV2 = accumulatorSquaredV2
				+ pow( TempHSDCBuffer[tempcounter + z], 2);

			double answer =(sqrt(accumulatorSquaredV2
				/ (numberOfDataPointsPerCycle* 2)))
				* conversionFactorV2
				* conversionFactorBinaryToVolt;
			TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)
				/ numberOfLinesSkipped) + z
				+ rowcounter * HSDCdata] = answer;

			accumulatorSquaredV2 = 0;
			accumulatorCounterV2 = 0;
		}
		break;
	}
	case 4:                                    // IC
	{
		instantIC = TempHSDCBuffer[tempcounter + z];

		if (accumulatorCounterI3
			< ((numberOfDataPointsPerCycle * 2) - 1))
		{
			accumulatorSquaredI3 = accumulatorSquaredI3
				+ pow( TempHSDCBuffer[tempcounter + z], 2);
			accumulatorCounterI3++;

		} else {
			accumulatorSquaredI3 = accumulatorSquaredI3
				+ pow( TempHSDCBuffer[tempcounter  + z], 2);

			double answer =(sqrt(accumulatorSquaredI3
				/ (numberOfDataPointsPerCycle* 2)))
				* conversionFactorI3
				* conversionFactorBinaryToAmp;

			TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)
				/ numberOfLinesSkipped) + z
				+ rowcounter * HSDCdata] = answer;

			accumulatorSquaredI3 = 0;
			accumulatorCounterI3 = 0;
		}
		break;
	}
	case 5:                              //VC
	{
		instantVC = TempHSDCBuffer[tempcounter + z];

		if (accumulatorCounterV3< ((numberOfDataPointsPerCycle * 2) - 1))
		{
			accumulatorSquaredV3 = accumulatorSquaredV3
				+ pow( TempHSDCBuffer[tempcounter + z],2);
			accumulatorCounterV3++;
```

```
			} else {
				accumulatorSquaredV3 = accumulatorSquaredV3
					+ pow( TempHSDCBuffer[tempcounter  + z], 2);
				double answer =(sqrt(accumulatorSquaredV3
					/ (numberOfDataPointsPerCycle* 2)))
					* conversionFactorV3
					* conversionFactorBinaryToVolt;

				TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)
					/ numberOfLinesSkipped) + z
					+ rowcounter * HSDCdata] = answer;

				accumulatorSquaredV3 = 0;
				accumulatorCounterV3 = 0;
			}
			break;
		}
		case 6:// IN
		{
			// Although calculated later, this is used to compare the
			//calculated value to the read value for error checking
			instantIN = TempHSDCBuffer[tempcounter + z];

			if (accumulatorCounterIN< ((numberOfDataPointsPerCycle * 2) - 1))
			{
				accumulatorSquaredIN = accumulatorSquaredIN
					+ pow( TempHSDCBuffer[tempcounter + z],2);
				accumulatorCounterIN++;

			} else {
				accumulatorSquaredIN = accumulatorSquaredIN
					+ pow( TempHSDCBuffer[tempcounter+ z], 2);

				double answer =(sqrt(accumulatorSquaredIN
					/ (numberOfDataPointsPerCycle* 2)))
					* conversionFactorIN
					* conversionFactorBinaryToAmp;


				TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)
					/ numberOfLinesSkipped) + z
					+ rowcounter * HSDCdata] = answer;

				accumulatorSquaredIN = 0;
				accumulatorCounterIN = 0;
			}
			break;
		}

		default: {
				break;
		}
	}

	/*
	* Order of calculations shown below:
	* sum i
	* summ i^2
	* sqrt ((sum i^2)*(wire res))
	* assume e ref 0
	* sum(v-eref)^2 + (0-eref)^2
	* sqrt the above
```

```
* above ans / sqrt wire res
* sum (v - ref)*i
*/
if (z > 6) {
// Once the 7 words have been stored, then perform the new power calculations

float x = neutralWireResistance / phaseWireResistance;
double neutralCurrent = -(instantIA + instantIB + instantIC);
double sqrSumCurrent = ((instantIA * instantIA) + (instantIB * instantIB)
       + (instantIC * instantIC)+ x * (neutralCurrent * neutralCurrent));
double Ibar = sqrt(  sqrSumCurrent * phaseWireResistance);
double voltageRef = (instantVA + instantVB+ instantVC) / (3 + (1 / x));
double sumE_Eref_sq = pow((instantVA - voltageRef), 2)
       + pow((instantVB - voltageRef), 2) + pow((instantVC - voltageRef), 2)
       + pow((-voltageRef / sqrt(x)), 2);
double V2Bar = sqrt(sumE_Eref_sq)  / sqrt(phaseWireResistance);
double RealPinstant = (instantVA - voltageRef)* instantIA
       + (instantVB - voltageRef) * instantIB
       + (instantVC - voltageRef) * instantIC
       + (0 - voltageRef) * neutralCurrent;
double ApparentPinstant = V2Bar * Ibar;
double powerFactorinstant = RealPinstant  / ApparentPinstant;
// then average over two cycles

if (accumulatorCounterNewP  < ((numberOfDataPointsPerCycle * 2) - 1))
{
       accumulatorNewP += RealPinstant;
       accumulatorNewS += ApparentPinstant;
       accumulatorNewPF += powerFactorinstant;

       accumulatorCounterNewP++;

} else {
       accumulatorNewP += RealPinstant;
       accumulatorNewS += ApparentPinstant;
       accumulatorNewPF += powerFactorinstant;
       averageNewP = (accumulatorNewP/ (numberOfDataPointsPerCycle * 2));
       averageNewS = (accumulatorNewS/ (numberOfDataPointsPerCycle * 2));
       averageNewPF = (accumulatorNewPF/ (numberOfDataPointsPerCycle * 2));
       averageNewQ = sqrt(  pow(averageNewS, 2)  - pow(averageNewP, 2));
       accumulatorNewP = 0;
       accumulatorNewS = 0;
       accumulatorNewPF = 0;
       accumulatorCounterNewP = 0;

       // Store calculated data at the end of the file

       TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)/ numberOfLinesSkipped) + 1
              + rowcounter * HSDCdata+ (noOfCycles / 2) * HSDCdata] =
              averageNewP   * conversionFactorBinaryToP;

       TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)/ numberOfLinesSkipped) + 2
              + rowcounter * HSDCdata+ (noOfCycles / 2) * HSDCdata] =
              averageNewS* conversionFactorBinaryToS;

       TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)/ numberOfLinesSkipped) + 3
              + rowcounter * HSDCdata+ (noOfCycles / 2) * HSDCdata] =
              averageNewQ* conversionFactorBinaryToQ;

       TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)/ numberOfLinesSkipped) + 4
              + rowcounter * HSDCdata+ (noOfCycles / 2) * HSDCdata] =
              averageNewPF  * conversionFactorBinaryToPF;
```

```
                                rowcounter++;

                        }
                }


        }
        j = j + (HSDCdata + 1) * 0.75;
        // To make the looping through of the skipped lines faster
        tempcounter = tempcounter + 7;

}

}

/* Register work area for each volume (Always succeeds regardless of disk
status) */
f_mount(0, &fs[0]);

sprintf(filename, "New%i.dat", SDcounter);// ensure name not too long otherwise
can't //cope - name can only be 8 characters long (filename is a 12 sized array
and .dat is //3 of them

/* Create destination file on the drive 0 */
res = f_open(&fdat, filename, FA_CREATE_ALWAYS | FA_WRITE);
if (res)
        exit(res);

// write buffer to SD card as a file
res = f_write(&fdat, TempHSDCBuffer,((BUFFER_SIZE_IN_WORDS /
numberOfLinesSkipped * 8)
        + ((noOfCycles / 2) * HSDCdata * 2 * 8)), &bw);
if (res)
        exit(res);

/* Close open file */
f_close(&fdat);

/* Unregister work area prior to discard it */
f_mount(0, NULL );
// increment SD file counter and the number of written files counter - they are
//related to ech other
SDcounter++;

counter++;

} else {
        // same comments as first section apply
        // It is repeated for the other buffer

        tempcounter = 0;
        rowcounter = 0;
        for (j = 0; j < (BUFFER_SIZE_IN_WORDS * 4); j++) {

                if (((j) % (numberOfLinesSkipped * 4 * HSDCdata)) == 0) {
                        for (z = 0; z < (HSDCdata + 1); z++) {

                                if ((z) < HSDCdata) {

                                        char* byte0address =
                                                (char*) (void*) &HSDCBuffer0[j + z];
                                        /*address of first byte*/
```

```
        char* byte1address = (char*) (void*) &HSDCBuffer0[j + z + 1];
        char* byte2address = (char*) (void*) &HSDCBuffer0[j + z + 2];
        char* byte3address = (char*) (void*) &HSDCBuffer0[j + z + 3];
        /*address of last byte*///lsb
        TempWordBuffer1[0] = *byte3address;             //LSB
        TempWordBuffer1[1] = *byte2address;
        TempWordBuffer1[2] = *byte1address;
        TempWordBuffer1[3] = *byte0address;

        TempHSDCBuffer[tempcounter + z] =  (double) *(int32_t *) &TempWordBuffer1[0];
}

j += 3;

switch (z) {

        case 0: {                                               // IA
                instantIA = TempHSDCBuffer[tempcounter + z];

                if (accumulatorCounterI1< ((numberOfDataPointsPerCycle * 2) - 1))
                {
                        accumulatorSquaredI1 = accumulatorSquaredI1
                                + pow( TempHSDCBuffer[tempcounter + z],   2);
                        accumulatorCounterI1++;
                } else {
                        accumulatorSquaredI1 = accumulatorSquaredI1
                                + pow( TempHSDCBuffer[tempcounter  + z], 2);

                        double answer =(sqrt(accumulatorSquaredI1
                                / (numberOfDataPointsPerCycle* 2)))
                                * conversionFactorI1 * conversionFactorBinaryToAmp;

                        TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)/ numberOfLinesSkipped)
                                + z    + rowcounter * HSDCdata] = answer;

                        accumulatorSquaredI1 = 0;
                        accumulatorCounterI1 = 0;
                }
                break;
        }
        case 1:                                                 //VA
        {
                instantVA = TempHSDCBuffer[tempcounter + z];

                if (accumulatorCounterV1    < ((numberOfDataPointsPerCycle * 2) - 1))
                {
                        accumulatorSquaredV1 = accumulatorSquaredV1
                                + pow( TempHSDCBuffer[tempcounter + z],   2);
                        accumulatorCounterV1++;

                } else {
                        accumulatorSquaredV1 = accumulatorSquaredV1
                                + pow( TempHSDCBuffer[tempcounter  + z], 2);

                        double answer =(sqrt(accumulatorSquaredV1
                                / (numberOfDataPointsPerCycle* 2)))
                                * conversionFactorV1 * conversionFactorBinaryToVolt;
                        TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)    / numberOfLinesSkipped)
                                + z + rowcounter * HSDCdata] = answer;
                        accumulatorSquaredV1 = 0;
                        accumulatorCounterV1 = 0;
                }
```

```
        break;
}
case 2:                                 //IB
{
        instantIB = TempHSDCBuffer[tempcounter + z];

        if (accumulatorCounterI2< ((numberOfDataPointsPerCycle * 2) - 1))
        {
                accumulatorSquaredI2 = accumulatorSquaredI2
                        + pow( TempHSDCBuffer[tempcounter + z],   2);
                accumulatorCounterI2++;
        } else {
                accumulatorSquaredI2 = accumulatorSquaredI2
                        + pow( TempHSDCBuffer[tempcounter  + z], 2);

                double answer =(sqrt(accumulatorSquaredI2
                        / (numberOfDataPointsPerCycle* 2)))
                        * conversionFactorI2 * conversionFactorBinaryToAmp;
                TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)    / numberOfLinesSkipped)
                        + z + rowcounter * HSDCdata] = answer;

                accumulatorSquaredI2 = 0;
                accumulatorCounterI2 = 0;
        }
        break;
}
case 3:                                                 //VB
{
        instantVB = TempHSDCBuffer[tempcounter + z];

        if (accumulatorCounterV2 < ((numberOfDataPointsPerCycle * 2) - 1))
        {
                accumulatorSquaredV2 = accumulatorSquaredV2
                        + pow( TempHSDCBuffer[tempcounter + z],   2);
                accumulatorCounterV2++;

        } else {
                accumulatorSquaredV2 = accumulatorSquaredV2
                        + pow( TempHSDCBuffer[tempcounter + z], 2);
                double answer = (sqrt( accumulatorSquaredV2
                        / (numberOfDataPointsPerCycle* 2)))
                        * conversionFactorV2 * conversionFactorBinaryToVolt;
                TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)    / numberOfLinesSkipped)
                        + z    + rowcounter * HSDCdata] = answer;

                accumulatorSquaredV2 = 0;
                accumulatorCounterV2 = 0;
        }
        break;
}
case 4:                                             // IC
{
        instantIC = TempHSDCBuffer[tempcounter + z];

        if (accumulatorCounterI3    < ((numberOfDataPointsPerCycle * 2) - 1))
        {
                accumulatorSquaredI3 = accumulatorSquaredI3
                        + pow( TempHSDCBuffer[tempcounter + z],2);
                accumulatorCounterI3++;
        } else {
                accumulatorSquaredI3 = accumulatorSquaredI3
```

```
                        + pow( TempHSDCBuffer[tempcounter  + z], 2);

                double answer = (sqrt(accumulatorSquaredI3
                        / (numberOfDataPointsPerCycle* 2)))
                        * conversionFactorI3 * conversionFactorBinaryToAmp;
                TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS) numberOfLinesSkipped) + z
                        + rowcounter * HSDCdata] = answer;
                accumulatorSquaredI3 = 0;
                accumulatorCounterI3 = 0;
        }
        break;
}
case 5:                     //VC
{
        instantVC = TempHSDCBuffer[tempcounter + z];

        if (accumulatorCounterV3 < ((numberOfDataPointsPerCycle * 2) - 1))
        {
                accumulatorSquaredV3 = accumulatorSquaredV3
                        + pow( TempHSDCBuffer[tempcounter + z],   2);
                accumulatorCounterV3++;

        } else {
                accumulatorSquaredV3 = accumulatorSquaredV3
                        + pow( TempHSDCBuffer[tempcounter + z], 2);

                double answer = (sqrt(accumulatorSquaredV3
                        / (numberOfDataPointsPerCycle * 2)))
                        * conversionFactorV3 * conversionFactorBinaryToVolt;

                TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)/ numberOfLinesSkipped) + z
                        + rowcounter * HSDCdata] = answer;

                accumulatorSquaredV3 = 0;
                accumulatorCounterV3 = 0;
        }
        break;
}
case 6:                                     // IN
{
        instantIN = TempHSDCBuffer[tempcounter + z];

        if (accumulatorCounterIN    < ((numberOfDataPointsPerCycle * 2) - 1))
        {
                accumulatorSquaredIN = accumulatorSquaredIN
                        + pow( TempHSDCBuffer[tempcounter + z],   2);
                accumulatorCounterIN++;
        } else {
                accumulatorSquaredIN = accumulatorSquaredIN
                        + pow( TempHSDCBuffer[tempcounter  + z], 2);

                double answer =(sqrt(accumulatorSquaredIN
                        / (numberOfDataPointsPerCycle* 2)))
                        * conversionFactorIN * conversionFactorBinaryToAmp;

                TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)/ numberOfLinesSkipped) + z
                        + rowcounter * HSDCdata] = answer;
                accumulatorSquaredIN = 0;
                accumulatorCounterIN = 0;
        }
        break;
}
```

```
        default: {
        break;
        }
}

if (z > 6) {
        // these values were taken as example
        float x = neutralWireResistance/ phaseWireResistance;
        double neutralCurrent = -(instantIA + instantIB + instantIC);
        double sqrSumCurrent = ((instantIA * instantIA) + (instantIB * instantIB)
                + (instantIC * instantIC)   + x * (neutralCurrent * neutralCurrent));
        double Ibar = sqrt(  sqrSumCurrent * phaseWireResistance);
        double voltageRef = (instantVA + instantVB + instantVC) / (3 + (1 / x));
        double sumE_Eref_sq = pow(  (instantVA - voltageRef), 2)
                + pow((instantVB - voltageRef), 2) + pow((instantVC - voltageRef), 2)
                + pow((-voltageRef / sqrt(x)), 2);
        double V2Bar = sqrt(sumE_Eref_sq)  / sqrt(phaseWireResistance);
        double RealPinstant = (instantVA - voltageRef)   * instantIA
                + (instantVB - voltageRef) * instantIB+ (instantVC - voltageRef) *
                instantIC+ (0 - voltageRef) * neutralCurrent;
        double ApparentPinstant = V2Bar * Ibar;
        double powerFactorinstant = 0;
        powerFactorinstant = RealPinstant  / ApparentPinstant;

        if (accumulatorCounterNewP  < ((numberOfDataPointsPerCycle * 2) - 1))
        {
                accumulatorNewP += RealPinstant;
                accumulatorNewS += ApparentPinstant;
                accumulatorNewPF += powerFactorinstant;
                accumulatorCounterNewP++;

        } else {
                accumulatorNewP += RealPinstant;
                accumulatorNewS += ApparentPinstant;
                accumulatorNewPF += powerFactorinstant;
                averageNewP = (accumulatorNewP/ (numberOfDataPointsPerCycle * 2));
                averageNewS = (accumulatorNewS/ (numberOfDataPointsPerCycle * 2));
                averageNewPF = (accumulatorNewPF/ (numberOfDataPointsPerCycle * 2));
                averageNewQ = sqrt(pow(averageNewS, 2)    - pow(averageNewP, 2));
                accumulatorNewP = 0;
                accumulatorNewS = 0;
                accumulatorNewPF = 0;
                accumulatorCounterNewP = 0;


                TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)/ numberOfLinesSkipped) + 1
                        + rowcounter * HSDCdata+ (noOfCycles / 2) * HSDCdata] =
                        averageNewP   * conversionFactorBinaryToP;

                TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)/ numberOfLinesSkipped) + 2
                        + rowcounter * HSDCdata     + (noOfCycles / 2) * HSDCdata] =
                        averageNewS   * conversionFactorBinaryToS;

                TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)/ numberOfLinesSkipped) + 3
                        + rowcounter * HSDCdata     + (noOfCycles / 2) * HSDCdata] =
                        averageNewQ * conversionFactorBinaryToQ;

                TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS)    / numberOfLinesSkipped) + 4
                        + rowcounter * HSDCdata     + (noOfCycles / 2) * HSDCdata] =
                        averageNewPF  * conversionFactorBinaryToPF;
                rowcounter++;
```

```
                }
        }

}
j = j + (HSDCdata + 1) * 0.75;
tempcounter = tempcounter + 7;

}
}

/* Register work area for each volume (Always succeeds regardless of disk status) */
f_mount(0, &fs[0]);

sprintf(filename, "New%i.dat", SDcounter);
/* Create destination file on the drive 0 */
res = f_open(&fdat, filename, FA_CREATE_ALWAYS | FA_WRITE);
if (res)
        exit(res);

res = f_write(&fdat, TempHSDCBuffer,((BUFFER_SIZE_IN_WORDS / numberOfLinesSkipped * 8)
        + ((noOfCycles / 2) * HSDCdata * 2 * 8)), &bw);
if (res)
        exit(res);

// if wished to read the file:
//res = f_read(&fdat, array_buffer, (8000 * 4), &br);
//if (res)
//     exit(res);

/* Close open file */
f_close(&fdat);

/* Unregister work area prior to discard it */
f_mount(0, NULL );
SDcounter++;
counter++;
}

if (counter > 10) {
        // when enough data has been stored end.
        DMA_Cmd(SPI_PORT_RX_DMA_STREAM, DISABLE);
        endRead = 1;
}
} else
;
/* Clear DMA Stream Transfer Complete interrupt pending bit */
DMA_ClearITPendingBit(SPI_PORT_RX_DMA_STREAM, DMA_IT_TCIF2 );

        }
}

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

ix. stm32f4xx_it.h

```
/**
******************************************************************************
**
**  File        : stm32f4xx_it.h
**
**  Abstract    : Main Interrupt Service Routines.
**                This file provides template for all exceptions handler and
**                peripherals interrupt service routine.
```

```
**
**  Environment : Atollic TrueSTUDIO(R)
**                STMicroelectronics STM32F4xx Standard Peripherals Library
**
**  Distribution: The file is distributed “as is,” without any warranty
**                of any kind.
**
**  (c)Copyright Atollic AB.
**  You may use this file as-is or modify it according to the needs of your
**  project. This file may only be built (assembled or compiled and linked)
**  using the Atollic TrueSTUDIO(R) product. The use of this file together
**  with other tools than Atollic TrueSTUDIO(R) is not permitted.
**
******************************************************************************
*/

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __STM32F4xx_IT_H
#define __STM32F4xx_IT_H

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx.h"

/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */

void NMI_Handler(void);
void HardFault_Handler(void);
void MemManage_Handler(void);
void BusFault_Handler(void);
void UsageFault_Handler(void);
void SVC_Handler(void);
void DebugMon_Handler(void);
void PendSV_Handler(void);
void SysTick_Handler(void);

void SPI_PORT_DMA_RX_IRQHandler(void);

#ifdef __cplusplus
}
#endif

#endif /* __STM32F4xx_IT_H */
```

B.2 Read HSDC

The code in section 10.2.2 and 10.2.1 except for main.c and main.h

i. Main.h

```
/*
 *     main.h
 *
 *     For all arrays and definitions used by more than one .c file
 *
 *  Created on: May 22, 2013
```

```
 *      Author: Christine Martindale
 *      RLRCHR002
 *      For Masters dissertation
 */

#ifndef MAIN_H_
#define MAIN_H_

#define frequency 2000                                                  // in Hz
#define HSDCdata 7          // 7 = instant only, 16 = instant and conventional power
#define noOfCycles 10       // this is the number of cycles to store in each .dat file

#define numberOfDataPointsPerCycle (frequency / 50)           //40
#define numberOfLinesSkipped (8000 / frequency)
       // 4 lines skipped when storing and using data from HSDC

       // 8000 because the EVAL records data at 8kHz

#define BUFFER_SIZE_IN_WORDS (noOfCycles * numberOfDataPointsPerCycle *
       numberOfLinesSkipped * HSDCdata) //11200
       // The line below is added when wanting to visually check that each file
       // follows on from the last with no data loss - it is 10.5 cycles instead of 10
       // + numberOfLinesSkipped * HSDCdata *numberOfDataPointsPerCycle/ 2)
       // ensure that above is within ram

int8_t HSDCBuffer0[BUFFER_SIZE_IN_WORDS * 4];          // Buffers used by the DMA
int8_t HSDCBuffer1[BUFFER_SIZE_IN_WORDS * 4];
int8_t TempWordBuffer0[4];
// Array used to change little endian to big endian byte order
int8_t TempWordBuffer1[4];

double TempHSDCBuffer[((BUFFER_SIZE_IN_WORDS) / numberOfLinesSkipped)
             + ((noOfCycles / 2) * HSDCdata * 2)];
// Array used to stored data to be written to a file on the SD card - instantaneous
//and results of power calculations
//5 rms rows and 5 power rows
// TempHSDCBuffer1 and TempHSDCBuffer0 used before but using just one TempHSDCBuffer
// uses less RAM
//double TempHSDCBuffer1[((BUFFER_SIZE_IN_WORDS)  / numberOfLinesSkipped )+
//((noOfCycles / 2) * HSDCdata * 2)];

#define ARRAY_SIZE (BUFFER_SIZE_IN_WORDS*2)

void read();
void SD_write(uint8_t* array_buffer);

#endif /* MAIN_H_ */
```

ii. Main.c

```
/**
 ******************************************************************************
 **
 **  File        : main.c
 **
 **  Abstract    :This function is the only thing that is different in this program
       from the data logger program
 **  This function allows the STM to read from the hsdc protocol made by another STM
       and not the EVAL
 **
 **  Functions   : main
 **
 **  Environment : Atollic TrueSTUDIO(R)
```

```
 **                STMicroelectronics STM32F4xx Standard Peripherals Library
 **
 **  Author          :      Christine Martindale
 **
 **  Description     : The beginnign of the read hsdc project
 **
 ******************************************************************************
 */

/* Includes */
#include "stm32f4xx.h"
#include <stdio.h>
#include "stm32f4_discovery.h"
#include "hsdc.h"
#include "ADE7878_registers.h"
#include "main.h"
#include "ff.h"

// below is used in read and SD write for testing/debugging

/*typedef enum {
       SD_CARD_OK = 1 << 13, LOGGING_ON = 1 << 12
} STATUS_BITS;

#define COMMS_TX_BUFFER_SIZE 256
volatile uint16_t global_statusBytes;

volatile uint8_t LoggingTask_logging_enabled;

#define pdPASS                                                          ( 1 )
#define pdFAIL                                                          ( 0 )*/


int main(void) {

       //uint32_t i;
       //for (i = 0; i < ARRAY_SIZE; i++) {
       //     array[i] = 0;
       //}

       init_SPI1(&HSDCBuffer0[0], &HSDCBuffer1[0], BUFFER_SIZE_IN_WORDS);

       /*//test variable
        uint32_t countBSY = 0;
        uint32_t countNTBSY = 0;*/

       //initialise the condition
       //condition = 0;
       //endRead = 0;

       /* Infinite loop */
       while (1) {



              // test code
              /*while ((SPI1 ->SR & SPI_I2S_FLAG_BSY )>> 7){
               //gets data while the hsa is low
               if (n > 31)
               n = 0;

               HSDCBuffer[n] = SPI1_receive();
```

```
                n++;
                countBSY++;
                }

                uint8_t i;
                for (i = 0; i < 32; i++) {
                HSDCBuffer[i] = 0;
                }

                while (!(SPI1 ->SR & SPI_I2S_FLAG_BSY )>> 7){
                countNTBSY++;
                }*/
        }
}

void read() {

        /*uint32_t i = 0;
        if (endRead == 0) {

               //if in one buffer then copy from other buffer to a new array
               if (!DMA_GetCurrentMemoryTarget(SPI_PORT_RX_DMA_STREAM )) {

                      SD_write(HSDCBuffer0);
                      for (i = 0; i < 8000; i++) {
                       array[counter + i] = HSDCBuffer0[i];
                       }
                       counter = counter + 8000;
                      condition = 1;
                      counter++;

               } else {
                      if (condition != 0) //don't write if first one!
                                    {
                             SD_write(HSDCBuffer1);
                             for (i = 0; i < 8000; i++) {
                              array[counter + i] = HSDCBuffer1[i];
                              }
                              counter = counter + 8000;
                             condition = 1;
                             counter++;
                      }
               }
               if (counter > 9) {
                      DMA_Cmd(SPI_PORT_RX_DMA_STREAM, DISABLE);
                      endRead = 1;
               }
        } else
               ;*/
// only get meaningful data if read in same packets that it is sent

}

void SD_write(uint8_t* array_buffer) {

        // Register work area for each volume (Always succeeds regardless of disk
               //status)
        /*/f_mount(0, &fs[0]);

        sprintf(filename, "New%i.dat", SDcounter);

        // Create destination file on the drive 0
```

```
	//res = f_open(&fdat, filename, FA_CREATE_ALWAYS | FA_WRITE);
	// ensure name not too long testagain.dat
	if (res)
		exit(res);

	res = f_write(&fdat, array_buffer, (BUFFER_SIZE_IN_WORDS * 4), &bw);
	if (res)
		exit(res);

	//res = f_read(&fdat, array_buffer, (8000 * 4), &br);
	//if (res)
	//	exit(res);
	//uint32_t temp = array1[5];

	// Close open file
	f_close(&fdat);

	// Unregister work area prior to discard it
	f_mount(0, NULL );
	SDcounter++;*/

}

/*
 * Callback used by stm32f4_discovery_audio_codec.c.
 * Refer to stm32f4_discovery_audio_codec.h for more info.
 */
void EVAL_AUDIO_TransferComplete_CallBack(uint32_t pBuffer, uint32_t Size) {
	/* TODO, implement your code here */
	return;
}

/*
 * Callback used by stm324xg_eval_audio_codec.c.
 * Refer to stm324xg_eval_audio_codec.h for more info.
 */
uint16_t EVAL_AUDIO_GetSampleCallBack(void) {
	/* TODO, implement your code here */
	return -1;
}
```

B.3 Produce HSDC

i. Main.c

```
/**
 ******************************************************************************
 **
 **  File        : main.c
 **
 **  Abstract    : This file sends 7 words of data at 8kHz with the HSDC protocol
 **
 **  Environment : Atollic TrueSTUDIO(R)
 **                STMicroelectronics STM32F4xx Standard Peripherals Library
 **
 ******************************************************************************
 */

/* Includes */
#include "stm32f4xx.h"
#include "stm32f4_discovery.h"
#include <stm32f4xx_spi.h>
```

```
#include <stm32f4xx_dma.h>

void EXTILine0_Config(void);

// Set up the timer and SPI
SPI_InitTypeDef SPI_InitStructure;
GPIO_InitTypeDef GPIO_InitStructure;
NVIC_InitTypeDef NVIC_InitStructure;
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_OCInitTypeDef TIM_OCInitStructure;
__IO uint16_t CCR1_Val = 749;
uint16_t PrescalerValue = 13; // to get 125us

#define SPI_PORT                                        SPI2
#define SPI_PORT_CLOCK                                  RCC_APB1Periph_SPI2
#define SPI_PORT_CLOCK_INIT                             RCC_APB1PeriphClockCmd
#define SPI_SCK_PIN                                     GPIO_Pin_13
#define SPI_SCK_GPIO_PORT                               GPIOB
#define SPI_SCK_GPIO_CLK                                RCC_AHB1Periph_GPIOB
#define SPI_SCK_SOURCE                                  GPIO_PinSource13
#define SPI_SCK_AF                                      GPIO_AF_SPI2
#define SPI_MOSI_PIN                                    GPIO_Pin_15
#define SPI_MOSI_GPIO_PORT                              GPIOB
#define SPI_MOSI_GPIO_CLK                               RCC_AHB1Periph_GPIOB
#define SPI_MOSI_SOURCE                                 GPIO_PinSource15
#define SPI_MOSI_AF                                     GPIO_AF_SPI2
#define SPI_NSS_PIN                                     GPIO_Pin_12
#define SPI_NSS_GPIO_PORT                               GPIOB
#define SPI_NSS_GPIO_CLK                                RCC_AHB1Periph_GPIOB
#define SPI_NSS_SOURCE                                  GPIO_PinSource12
#define SPI_NSS_AF                                      GPIO_AF_SPI2

/* Definition for DMAx resources */
#define SPI_PORT_DMA                                    DMA1
#define SPI_PORT_DMAx_CLK                               RCC_AHB1Periph_DMA1
#define SPI_PORT_TX_DMA_CHANNEL                 DMA_Channel_0
#define SPI_PORT_TX_DMA_STREAM                          DMA1_Stream4
#define SPI_PORT_TX_DMA_FLAG_FEIF               DMA_FLAG_FEIF4
#define SPI_PORT_TX_DMA_FLAG_DMEIF                      DMA_FLAG_DMEIF4
#define SPI_PORT_TX_DMA_FLAG_TEIF               DMA_FLAG_TEIF4
#define SPI_PORT_TX_DMA_FLAG_HTIF               DMA_FLAG_HTIF4
#define SPI_PORT_TX_DMA_FLAG_TCIF               DMA_FLAG_TCIF4
#define SPI_PORT_DMA_TX_IRQn                            DMA1_Stream4_IRQn
#define SPI_PORT_DMA_TX_IRQHandler                      DMA1_Stream4_IRQHandler

/**
 **===========================================================================
 **
 **  Abstract: main program
 **
 **===========================================================================
 */
int main(void) {

        /* Configure EXTI Line15 (connected to PG15 pin) in interrupt mode */
        EXTILine0_Config();
        // this is used for starting the HSDC when the button is pushed

        //initialise LED, LED will be used to show the interrupt timer is correct
        STM_EVAL_LEDInit(LED3);
        STM_EVAL_LEDOn(LED3);
```

```
// enable the SPI peripheral clock
SPI_PORT_CLOCK_INIT(SPI_PORT_CLOCK, ENABLE);

// enable the peripheral GPIO port clocks
RCC_AHB1PeriphClockCmd(
              SPI_SCK_GPIO_CLK | SPI_MOSI_GPIO_CLK | SPI_NSS_GPIO_CLK, ENABLE);

// enable the DMA peripheral clock
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);

/* TIM3 clock enable */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

/* Configure PA4 pin as input floating */
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
GPIO_Init(GPIOA, &GPIO_InitStructure);

GPIO_WriteBit(GPIOA, GPIO_Pin_4, Bit_SET);

// Connect SPI pins to AF5 - see section 3, Table 6 in the device datasheet
GPIO_PinAFConfig(SPI_SCK_GPIO_PORT, SPI_SCK_SOURCE, SPI_SCK_AF );
GPIO_PinAFConfig(SPI_MOSI_GPIO_PORT, SPI_MOSI_SOURCE, SPI_MOSI_AF );
GPIO_PinAFConfig(SPI_NSS_GPIO_PORT, SPI_NSS_SOURCE, SPI_NSS_AF );
// now configure the pins themselves
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;

GPIO_InitStructure.GPIO_Pin = SPI_SCK_PIN;
GPIO_Init(SPI_SCK_GPIO_PORT, &GPIO_InitStructure);

GPIO_InitStructure.GPIO_Pin = SPI_MOSI_PIN;
GPIO_Init(SPI_MOSI_GPIO_PORT, &GPIO_InitStructure);

GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Pin = SPI_NSS_PIN;
GPIO_Init(SPI_NSS_GPIO_PORT, &GPIO_InitStructure);
// now we can set up the SPI peripheral
// Assume the target is write only and we look after the chip select ourselves
// SPI clock rate will be system frequency/2/prescaler
// so here we will go for 84/2/8 = 5.25MHz so leave at the slower one, or
// 84/2/4 = 10.5Mhz
SPI_I2S_DeInit(SPI_PORT );
SPI_StructInit(&SPI_InitStructure);
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
SPI_InitStructure.SPI_Direction = SPI_Direction_1Line_Tx;
SPI_InitStructure.SPI_NSS = SPI_NSS_Hard;
SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_8;
SPI_Init(SPI_PORT, &SPI_InitStructure);

/* Compute the prescaler value */
PrescalerValue = (uint16_t) ((SystemCoreClock / 2) / 6000000) - 1;

/* Time base configuration */
TIM_TimeBaseStructure.TIM_Period = 749;
TIM_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
```

```
	TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
	TIM_TimeBaseStructure.TIM_RepetitionCounter = 3;
	TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

	/* Enable the TIM3 global Interrupt */
	NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
	NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
	NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;     //
	NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
	NVIC_Init(&NVIC_InitStructure);

	/* Enable the DMA Stream IRQ Channel */
	NVIC_InitStructure.NVIC_IRQChannel = DMA1_Stream4_IRQn;
	NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
	NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
	NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
	NVIC_Init(&NVIC_InitStructure);

	/* Prescaler configuration */
	TIM_PrescalerConfig(TIM3, PrescalerValue, TIM_PSCReloadMode_Immediate );

	/* TIM Interrupts enable */
	TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);

	/* TIM3 enable counter */
	TIM_Cmd(TIM3, ENABLE);

	/* Infinite loop */
	while (1) {

		//*******************************
		//test for SPI
		// make sure the transmit buffer is free
		/*while (SPI_I2S_GetFlagStatus(SPI_PORT, SPI_I2S_FLAG_TXE ) == RESET)
		 ;
		 uint8_t n;
		 for (n = 0; n < 32; n++) {
		 SPI_I2S_SendData(SPI_PORT, data[n]);
		 // we are not reading data so be sure that the character goes to the
		//shift register
		 while (SPI_I2S_GetFlagStatus(SPI_PORT, SPI_I2S_FLAG_TXE ) == RESET)
		 ;
		 }
		 // and then be sure it has been sent over the wire
		 while (SPI_I2S_GetFlagStatus(SPI_PORT, SPI_I2S_FLAG_BSY ) == SET)
		 ;*/
		//*************************************
	}
}

/**
 * @brief  Configures EXTI Line0 (connected to PA0 pin) in interrupt mode
 * @param  None
 * @retval None
 */
void EXTILine0_Config(void) {
	EXTI_InitTypeDef EXTI_InitStructure;
	GPIO_InitTypeDef GPIO_InitStructure;
	NVIC_InitTypeDef NVIC_InitStructure;

	/* Enable GPIOA clock */
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
```

```
	/* Enable SYSCFG clock */
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

	/* Configure PA0 pin as input floating */
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
	GPIO_Init(GPIOA, &GPIO_InitStructure);

	/* Connect EXTI Line0 to PA0 pin */
	SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0 );

	/* Configure EXTI Line0 */
	EXTI_InitStructure.EXTI_Line = EXTI_Line0;
	EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
	EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
	EXTI_InitStructure.EXTI_LineCmd = ENABLE;
	EXTI_Init(&EXTI_InitStructure);

	/* Enable and set EXTI Line0 Interrupt to the lowest priority */
	NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
	NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
	NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
	NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
	NVIC_Init(&NVIC_InitStructure);
}

/*
 * Callback used by stm32f4_discovery_audio_codec.c.
 * Refer to stm32f4_discovery_audio_codec.h for more info.
 */
void EVAL_AUDIO_TransferComplete_CallBack(uint32_t pBuffer, uint32_t Size) {
	/* TODO, implement your code here */
	return;
}

/*
 * Callback used by stm324xg_eval_audio_codec.c.
 * Refer to stm324xg_eval_audio_codec.h for more info.
 */
uint16_t EVAL_AUDIO_GetSampleCallBack(void) {
	/* TODO, implement your code here */
	return -1;
}
```

ii. Stm32f4xx_it.h

```
/**
******************************************************************************
**
**  File        : stm32f4xx_it.h
**
**  Abstract    : Main Interrupt Service Routines.
**                This file provides template for all exceptions handler and
**                peripherals interrupt service routine.
**
**  Environment : Atollic TrueSTUDIO(R)
**                STMicroelectronics STM32F4xx Standard Peripherals Library
**
**  Distribution: The file is distributed “as is,” without any warranty
**                of any kind.
**
**  (c)Copyright Atollic AB.
```

```
**  You may use this file as-is or modify it according to the needs of your
**  project. This file may only be built (assembled or compiled and linked)
**  using the Atollic TrueSTUDIO(R) product. The use of this file together
**  with other tools than Atollic TrueSTUDIO(R) is not permitted.
**
*****************************************************************************
*/

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __STM32F4xx_IT_H
#define __STM32F4xx_IT_H

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx.h"

/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */

void NMI_Handler(void);
void HardFault_Handler(void);
void MemManage_Handler(void);
void BusFault_Handler(void);
void UsageFault_Handler(void);
void SVC_Handler(void);
void DebugMon_Handler(void);
void PendSV_Handler(void);
void SysTick_Handler(void);

void TIM2_IRQHandler(void);
void DMA1_Stream4_IRQHandler(void);
void EXTI0_IRQHandler(void);

#ifdef __cplusplus
}
#endif

#endif /* __STM32F4xx_IT_H */
```

iii. stm32f4xx_it.c

```
/**
 *****************************************************************************
 * @file    Project/STM32F4xx_StdPeriph_Template/stm32f4xx_it.c
 * @author  MCD Application Team
 * @version V1.1.0
 * @date    18-January-2013
 * @brief   Main Interrupt Service Routines.
 *          This file provides template for all exceptions handler and
 *          peripherals interrupt service routine.
 *****************************************************************************
 * @attention
 *
 * <h2><center>&copy; COPYRIGHT 2013 STMicroelectronics</center></h2>
```

```
 *
 * Licensed under MCD-ST Liberty SW License Agreement V2, (the "License");
 * You may not use this file except in compliance with the License.
 * You may obtain a copy of the License at:
 *
 *        http://www.st.com/software_license_agreement_liberty_v2
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 ******************************************************************************
 */

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_it.h"
#include "stm32f4_discovery.h"

#define SPI_PORT_TX_DMA_STREAM                          DMA1_Stream4
#define SPI_PORT                                              SPI2
#define SPI_PORT_TX_DMA_CHANNEL                 DMA_Channel_0

/** @addtogroup Template_Project
 * @{
 */

uint16_t capture = 0;
uint16_t capture2 = 0;
extern __IO uint16_t CCR1_Val;
DMA_InitTypeDef DMA_InitStructure;
uint8_t data[64];
uint8_t PushButtonStart = 0;

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/

/******************************************************************************/
/*            Cortex-M4 Processor Exceptions Handlers                         */
/******************************************************************************/

/**
 * @brief   This function handles NMI exception.
 * @param  None
 * @retval None
 */
void NMI_Handler(void) {
}

/**
 * @brief  This function handles Hard Fault exception.
 * @param  None
 * @retval None
 */
void HardFault_Handler(void) {
       /* Go to infinite loop when Hard Fault exception occurs */
       while (1) {
```

```
	}
}

/**
 * @brief  This function handles Memory Manage exception.
 * @param  None
 * @retval None
 */
void MemManage_Handler(void) {
	/* Go to infinite loop when Memory Manage exception occurs */
	while (1) {
	}
}

/**
 * @brief  This function handles Bus Fault exception.
 * @param  None
 * @retval None
 */
void BusFault_Handler(void) {
	/* Go to infinite loop when Bus Fault exception occurs */
	while (1) {
	}
}

/**
 * @brief  This function handles Usage Fault exception.
 * @param  None
 * @retval None
 */
void UsageFault_Handler(void) {
	/* Go to infinite loop when Usage Fault exception occurs */
	while (1) {
	}
}

/**
 * @brief  This function handles SVCall exception.
 * @param  None
 * @retval None
 */
void SVC_Handler(void) {
}

/**
 * @brief  This function handles Debug Monitor exception.
 * @param  None
 * @retval None
 */
void DebugMon_Handler(void) {
}

/**
 * @brief  This function handles PendSVC exception.
 * @param  None
 * @retval None
 */
void PendSV_Handler(void) {
}

/**
 * @brief  This function handles SysTick Handler.
```

```
 * @param  None
 * @retval None
 */
void SysTick_Handler(void) {
	/*  TimingDelay_Decrement(); */
}

/******************************************************************************/
/*            STM32F4xx Peripherals Interrupt Handlers                        */
/******************************************************************************/

/**
 * @brief  This function handles TIM3 global interrupt request.
 * @param  None
 * @retval None
 */

/**
 * @brief  This function handles External line 0 interrupt request.
 * @param  None
 * @retval None
 */
void EXTI0_IRQHandler(void) {
	// when the push button is pressed ...
	if (EXTI_GetITStatus(EXTI_Line0 ) != RESET) {
		PushButtonStart = 1;

		/* Clear the EXTI line 0 pending bit */
		EXTI_ClearITPendingBit(EXTI_Line0 );
	}
}

void DMA1_Stream4_IRQHandler(void) {

	/* Test on DMA Stream Transfer Complete interrupt */
	if (DMA_GetITStatus(SPI_PORT_TX_DMA_STREAM, DMA_IT_TCIF4 )) {
		/* Clear DMA Stream Transfer Complete interrupt pending bit */
		DMA_ClearITPendingBit(SPI_PORT_TX_DMA_STREAM, DMA_IT_TCIF4 );

		GPIO_WriteBit(GPIOA, GPIO_Pin_4, Bit_SET);
		// make hsa line low here
	}
}

void TIM3_IRQHandler(void) {

	if (PushButtonStart) {

		// processing not normally in interrupt but timing of the start of this
	//essential. main program actually does nothing.
		if (TIM_GetITStatus(TIM3, TIM_IT_Update ) != RESET) {
			TIM_ClearITPendingBit(TIM3, TIM_IT_Update );

			STM_EVAL_LEDToggle(LED3);

			uint8_t i;
			//if (capture == 0)

			//{
			// Below is a method of producing data so that it counts up each
			//set - to ensure no data is lost
			for (i = 0; i < 28; i++) {
```

```
		if ((i + 1) % 4 == 0) {
			if (i == 3)
				data[i] = capture2;
			else
				data[i] = capture;
		} else
			data[i] = 0;
	}

	/*} else {
	 for (i = 0; i < 32; i++) {
	 if (i % 2 == 0)
	 data[i] = 0xAA;
	 else
	 data[i] = 0x00;
	 }*/
	//}
	if (capture < 255) {
		capture++;
	} else {
		capture = 0;
		capture2++;
	}

	/*for (i = 10; i < 20; i++) {
	 data[i] = 0;
	 }
	 for (i = 20; i < 30; i++) {
	 data[i] = 0xFF;
	 }*/

	// because in normal mode has to be reinitialised every time
	// start with a blank DMA configuration just to be sure
	DMA_DeInit(SPI_PORT_TX_DMA_STREAM );
	/*
	 * Check if the DMA Stream is disabled before enabling it.
	 * Note that this step is useful when the same Stream is used
	multiple times:
	 * enabled, then disabled then re-enabled... In this case, the DMA
	Stream disable
	 * will be effective only at the end of the ongoing data transfer
	and it will
	 * not be possible to re-configure it before making sure that the
	Enable bit
	 * has been cleared by hardware. If the Stream is used only once,
	this step might
	 * be bypassed.
	 */
	while (DMA_GetCmdStatus(SPI_PORT_TX_DMA_STREAM ) != DISABLE)
		;
	// Configure DMA controller to manage TX DMA requests
	// first make sure we are using the default values
	DMA_StructInit(&DMA_InitStructure);
	// these are the only parameters that change from the defaults
	DMA_InitStructure.DMA_PeripheralBaseAddr =
			(uint32_t) &(SPI_PORT ->DR);
	DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t) &data[0];
	DMA_InitStructure.DMA_Channel = SPI_PORT_TX_DMA_CHANNEL;
	DMA_InitStructure.DMA_DIR = DMA_DIR_MemoryToPeripheral;
	DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
	DMA_InitStructure.DMA_Priority = DMA_Priority_High;
	DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
```

```
                    DMA_InitStructure.DMA_BufferSize = 28;
                    DMA_Init(SPI_PORT_TX_DMA_STREAM, &DMA_InitStructure);

                    GPIO_WriteBit(GPIOA, GPIO_Pin_4, Bit_RESET);
                    // make hsa line low here

                    DMA_Cmd(SPI_PORT_TX_DMA_STREAM, ENABLE);
                    SPI_I2S_DMACmd(SPI_PORT, SPI_I2S_DMAReq_Tx, ENABLE);

                    /* Enable DMA Stream Transfer Complete interrupt */
                    DMA_ITConfig(SPI_PORT_TX_DMA_STREAM, DMA_IT_TC, ENABLE);

                    // Enable the SPI port=====================================
                    SPI_Cmd(SPI_PORT, ENABLE);


            }
        }

}

/*****************************************************************************/
/*                 STM32F4xx Peripherals Interrupt Handlers                  */
/*  Add here the Interrupt Handler for the used peripheral(s) (PPP), for the  */
/*  available peripheral interrupt handler's name please refer to the startup */
/*  file (startup_stm32f40xx.s/startup_stm32f427x.s).                         */
/*****************************************************************************/

/**
 * @brief  This function handles PPP interrupt request.
 * @param  None
 * @retval None
 */
/*void PPP_IRQHandler(void)
 {
 }*/

/**
 * @}
 */

/**
 * @brief  This function handles SPI interrupt request.
 * @param  None
 * @retval None
 */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

C. C# Code

i. Program.cs

```
/* Author: Christine Martindale
 * Name: PC-side C# code
 * For Masters dissertation
 * Description: To convert .dat files which are written in binary into .csv files
 *              Was also used for testing - this code is commented out
 * 01/03/13
```

```
 *
 */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Threading.Tasks;
using System.IO;

namespace DatConverter
{
    class Program
    {
        static void Main(string[] args)
        {

            /*
              * This section of code was to test the byte conversion from little endian
              to big endian using NetworkToHostOrder() method
                //  int networkByte = -1064655361; //in hex signed that is: C08AA5FF
                //  int hostByte;
             // Converts an integer value from network byte order to host byte order.
                //  hostByte = IPAddress.NetworkToHostOrder(networkByte);
              // Console.WriteLine("Network byte order to Host byte order of {0} is
              {1}", networkByte, hostByte);

            */

            //The code is laid out to convert only one .dat file at a time, but a for
            loop can be added to convert more than one file
            //The console is used as an easy debug tool to write out errors and check
            data

            //The location of the file to be converted is saved to filename:
            string fileName = @"C:\svn_datalogger\data\New folder\NEW3.dat";
                    //@"H:\NEW3.dat";
                    // Other commonly used sources:
                    //@"C:\svn_datalogger\data\3 ph resisitive\NEW2.dat";
                    // @"C:\svn_datalogger\DATA NI\test 1 177.54\NEW0.dat";
                      // The C code always saves the file as NEWx.dat where x is the
                     file number representing the order that the data was recorded in


            if (File.Exists(fileName)) // If the file is found:
            {
                 Console.WriteLine(fileName + " exists.");
                    // therefore correct source name


                // Create the file stream and the reader for data.
                FileStream fs = new FileStream(fileName, FileMode.Open,
                     FileAccess.Read);
                BinaryReader r = new BinaryReader(fs);

                double data; //where the data as a double is stored

                // variables for testing little to big endian conversion and
                     calculation accuracy

                /*int networkByte;
```

```
double accumulatorSquaredV1 = 0, accumulatorSquaredV2 = 0,
     accumulatorSquaredV3 = 0, accumulatorSquaredI1 = 0,
    accumulatorSquaredI2 = 0, accumulatorSquaredI3 = 0,
     accumulatorSquaredIN = 0, accumulatorNewP = 0,
    accumulatorNewS = 0, accumulatorNewPF = 0;

double numberOfDataPointsPerCycle = 20;
double conversionFactorBinaryToVolt = 0.95520 / 10000;
double conversionFactorBinaryToAmp = 0.33804 / 100000; ;//TODO get
     real value

double conversionFactorV1 = 1;
double conversionFactorV2 = 0.99578;
double conversionFactorV3 = 0.99701;

double conversionFactorI1 = 1;
double conversionFactorI2 = 0.98906;//TODO get real value
double conversionFactorI3 = 0.96789;//TODO get real value
double conversionFactorIN = 1;//TODO get real value

double conversionFactorAVA = 1;
double conversionFactorBVA = 0.98532;//CALC BY AVERAGE
double conversionFactorCVA = 0.96491;//CALC BY AVERAGE

double conversionFactorAWATT = 1;
double conversionFactorBWATT = 0.98351;//CALC BY AVERAGE
double conversionFactorCWATT = 0.95852;//CALC BY AVERAGE

double conversionFactorAVAR = 1;
double conversionFactorBVAR = 1;//TODO
double conversionFactorCVAR = 1;//TODO

double instantIA = 0, instantIB = 0, instantIC = 0, instantIN = 0,
     instantVA = 0, instantVB = 0, instantVC = 0;
double averageNewP = 0, averageNewS = 0, averageNewQ = 0, averageNewPF
     = 0;

int accumulatorCounterNewP = 0, accumulatorCounterV1 = 0,
     accumulatorCounterV2 = 0, accumulatorCounterV3 = 0,
    accumulatorCounterI1 = 0, accumulatorCounterI2 = 0,
    accumulatorCounterI3 = 0, accumulatorCounterIN = 0;
 * */

//create a stream writer for the new file as a .csv
using (StreamWriter sw = new StreamWriter("3ph resistvie 3 new
     folder.csv"))
{
     // loop through until the r stream which is the original .dat file
            is finished
     // the count counts up to the length of the .dat file divided by 8
            bytes for the double
    // divided by 7 for the words in the inner for loop (the number of
            columns
   for (int count = 0; count < (r.BaseStream.Length / 56); count++)
    {
        // all 7 bytes of data sent through - this would have to
            change if the setting of hsdc changed

        for (int i = 0; i < (7); i++)
        {
            data = r.ReadDouble();// to read binary data as a double
```

```
                    from the stream

                //Below is code used for testing at different stages,
                    mostly testing for correct data conversion

                    …. It can be found on the attachéd CD


                        sw.Write(averageNewP);
                        sw.Write(",");
                        sw.Write(averageNewS);
                        sw.Write(",");
                        sw.Write(averageNewQ);
                        sw.Write(",");
                        sw.Write(averageNewPF);
                        sw.Write(",");


                      }*/

                // write the value stored in data to the file stream and
                    add a comma
                sw.Write(data);

                sw.Write(",");// comma delimited



            }

           //at the end of each row add a new line symbol

            sw.Write("\n");//end of line after the 16 words

        }
    }


    r.Close();
    fs.Close();// close both new (CSV ) and old (DAT) files


         // }


    }
    else
    {
        // If the file is not found:
        Console.WriteLine("Specified file not found at " + fileName);

        // This code was to show how to create a new file
        // Create the new, empty data file.
        //  FileStream fs1 = new FileStream(fileName, FileMode.CreateNew);
        // Create the writer for data.
        //  BinaryWriter w = new BinaryWriter(fs1);
        // Write data to Test.data.
        //  for (int i = 0; i < 5; i++)
        //  {
        //      w.Write((int)i);
        //  }
```

```
                    //  w.Close();
                    //  fs1.Close();
                }

            //End program and close console
                Console.WriteLine("Press any key to continue.");// just to see that it
                     has been done - for testing
                Console.ReadKey();
        }
    }
}
```

D. Data Correction Justification

D.1 Problem Identification

The original data shown in this dissertation gave the following power calculation results:

Table 6.5 New Power Calculations

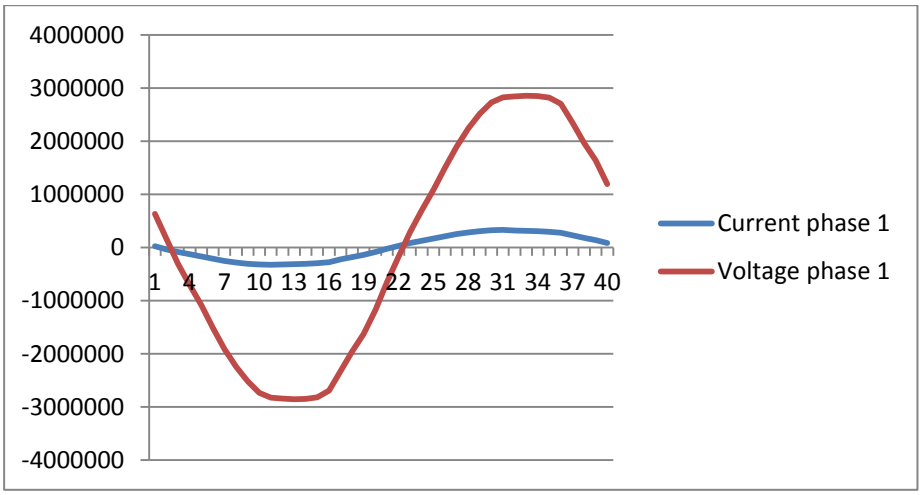
P (W)	S (VA)	Q (VAR)	PF
-602.735	1942.666	1846.798	-0.24473

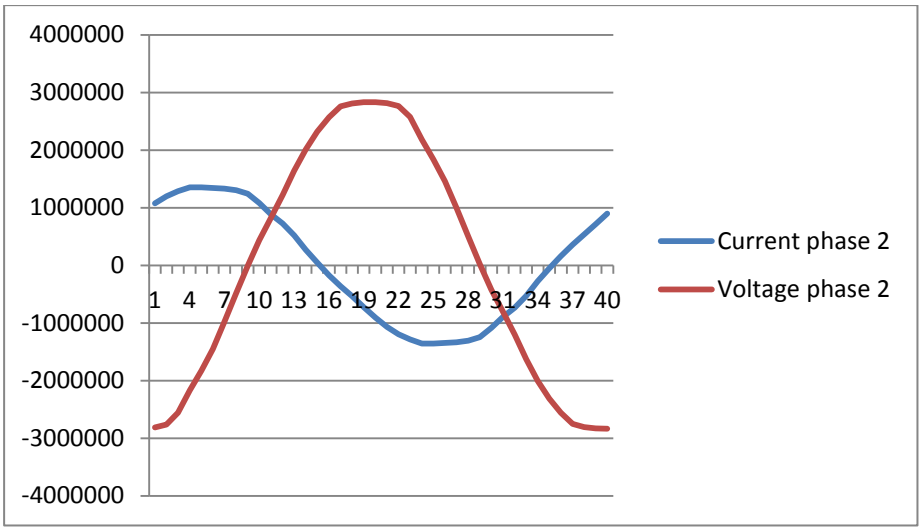
There are two problems here:

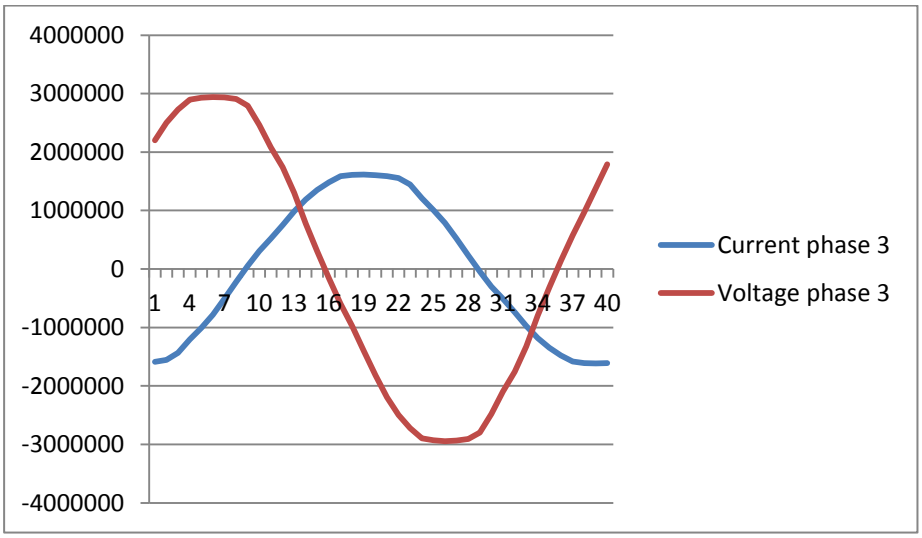
- The power values is negative implying that the logger is measuring a source (generator) rather than the resistive load described in sections 5 and 6.
- The PF shown is -0.24473. this is not the same as P/S = -602.735/1942.666 = 0.3103. this is explained in section 6.

To address the first problem the current and voltage values per phase were plotted in figures 1, 2 and 3 below.

The y-axis is the value stored in the.dat file obtained from the logger and is respectively proportional to voltage and current. The x-axis is the sample number.



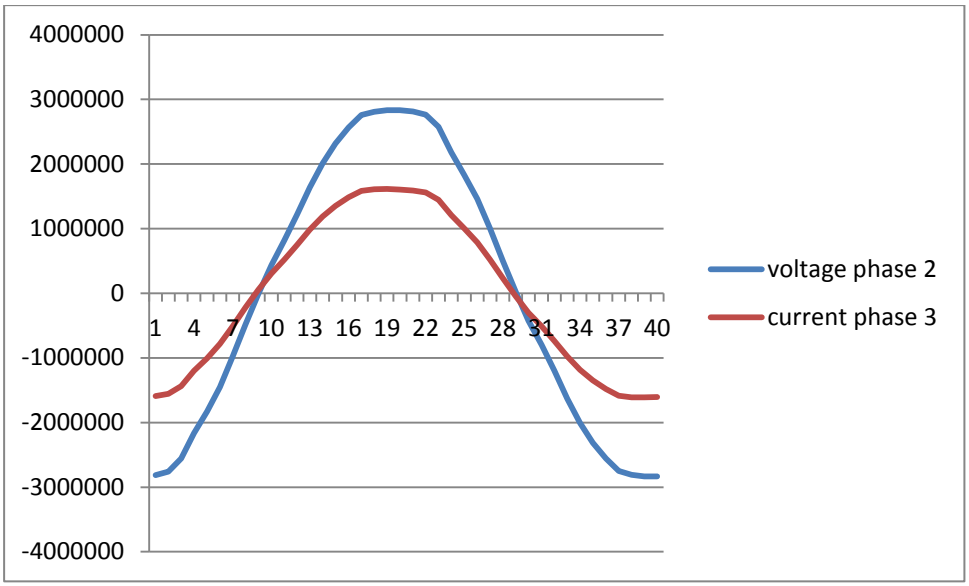


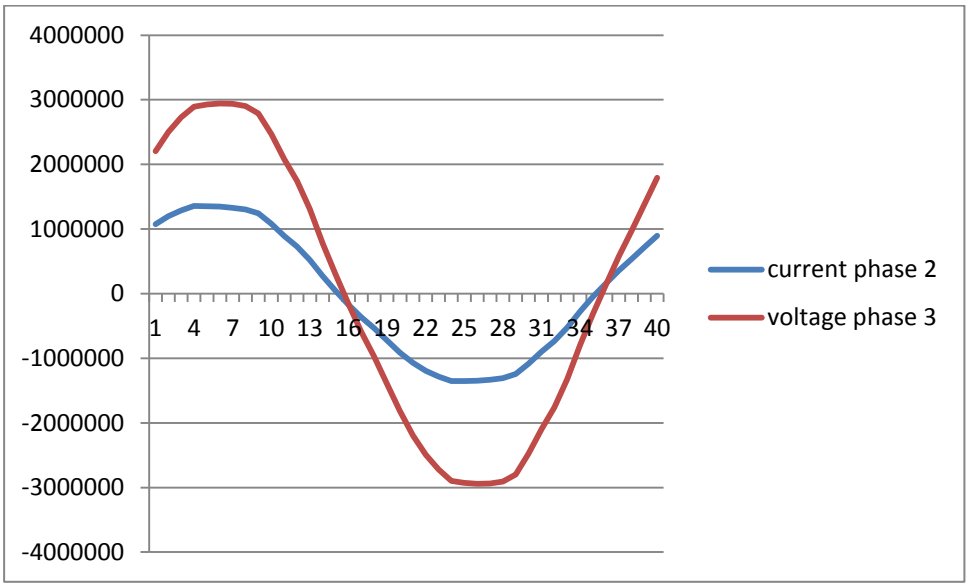


The above figures imply that only phase 1 has a resistive only load.

D.2 Possible Alternatives and Credibility Check

1. The phase and neutral resistances were assumed and not measured. This would only affect the Q, S and PF values and would have no impact on P because the power is calculated by averaging the instantaneous powers over two cycles. The calculation of instantaneous power does not include these resistances.
2. The data shown in the graphs was then multiplied by conversion factors. These conversion factors were all positive and so this would not affect the calculated power values.
3. The load may not have been resistive only. This is unlikely given the set-up of resistive loads as explained in section 5.
4. The current transformers or the voltage transformers for the phases were interchanged. If this was so then the graphs that would have been obtained for phase 2 and 3 are shown below.





D.3 Choice of Solution

From the graphs shown above it is likely that the transformers were interchanged because the resulting graphs show in phase voltage and current which is expected for a resistive only load.

If the respective data columns are then interchanged the following power calculations are obtained:

P	S	PF	Q
1627.785305	1942.389095	0.824964	1059.806773

As illustrated above, the power is now positive.

Therefore the results shown in section 6 were adjusted by interchanging voltage phase 2 and voltage phase 3. However, the original .dat files are on the CD submitted with this thesis and the original data for section 6 is shown at the end of this appendix.

D.4 Impact

The results that are sensitive to this correction are the graphs and tables shown in section 6.3. However it does not affect the following sections because the mistake was in the input currents to the logger, and would not have affected the loggers ability to perform the calculations.

D.5 Recommendation

Should this dissertation and the data logger described be used to make important decisions and recommendations then the aspect of this error must be checked.

As stated in the conclusion "The accuracy is mostly dependant on the accuracy of the calibration" as well as the accuracy of the setup.

D.6 Original Data (as originally labelled in Chapter 6)

Table 6.1 Excerpt from csv file of results. Lines 1 through 400 display unit-less values. Lines 401 through 405 current values in amperes and voltage values in volts. Lines 406 through 410 display real power in W, apparent power in VAR, non-active power in VA and power factor is unit-less. The conversion factors used for these calculations are displayed in the paragraph following this table.

Line number	Current phase 1	Voltage phase 1	Current phase 2	Voltage phase 2	Current phase 3	Voltage phase 3
1	21113	632906	1074042	-2811709	-1589701	2202792
2	-34187	159189	1199454	-2759970	-1554839	2504056
3	-85716	-309573	1289414	-2557867	-1434138	2729301
4	-127213	-711247	1354938	-2166410	-1201146	2895363
5	-167738	-1085114	1354461	-1829517	-1004123	2927984
6	-212133	-1511376	1345193	-1449201	-782283	2941639
7	-252066	-1910464	1329916	-977801	-508386	2935113
8	-282379	-2248996	1304849	-481117	-219077	2905196
9	-305503	-2519328	1242163	-7799	54254	2790305
10	-322175	-2733368	1078746	434674	305683	2466062
11	-326240	-2824710	889631	818208	521329	2079901
12	-319244	-2845068	725725	1220217	748537	1743511
13	-313638	-2853060	519152	1641179	984413	1306131
14	-305473	-2845642	270302	2011577	1187970	778967
15	-296155	-2816634	45604	2318487	1353372	302665
16	-275495	-2692246	-172723	2567598	1483788	-162009
17	-225983	-2323882	-364549	2757761	1587581	-594390
18	-180022	-1957635	-535935	2809358	1608905	-976462

19	-137501	-1627715	-725619	2831666	1613193	-1398095
20	-85012	-1174347	-909212	2831824	1606243	-1814825
Continued …						
497	259124	2575771	239540	-2638950	-1524514	314171
498	209468	2174517	422062	-2794693	-1607150	720734
499	167622	1854141	604248	-2822425	-1612262	1125816
400	121056	1485201	790691	-2835144	-1611927	1548585
Calculated per two cycles	Current RMS phase 1	Voltage RMS phase 1	Current RMS phase 2	Voltage RMS phase 2	Current RMS phase 3	Voltage RMS phase 3
401	0.79732	200.114	3.29135	197.7679	3.87166	200.152
402	0.797286	200.1242	3.29096	197.7763	3.871737	200.1405
403	0.797195	200.1099	3.291424	197.7551	3.871425	200.1595
404	0.797362	200.1004	3.291995	197.79	3.871092	200.223
405	0.796781	200.013	3.291928	197.8719	3.872917	200.2189
Calculated per two cycles	-	Real power	Apparent power	Non-active power	Power factor	-
406	0	-602.735	1942.666	1846.798	-0.24473	0
407	0	-602.657	1942.656	1846.813	-0.24486	0
408	0	-603.113	1942.511	1846.511	-0.2452	0
409	0	-603.597	1942.725	1846.578	-0.24538	0
410	0	-602.462	1943.72	1847.995	-0.24487	0

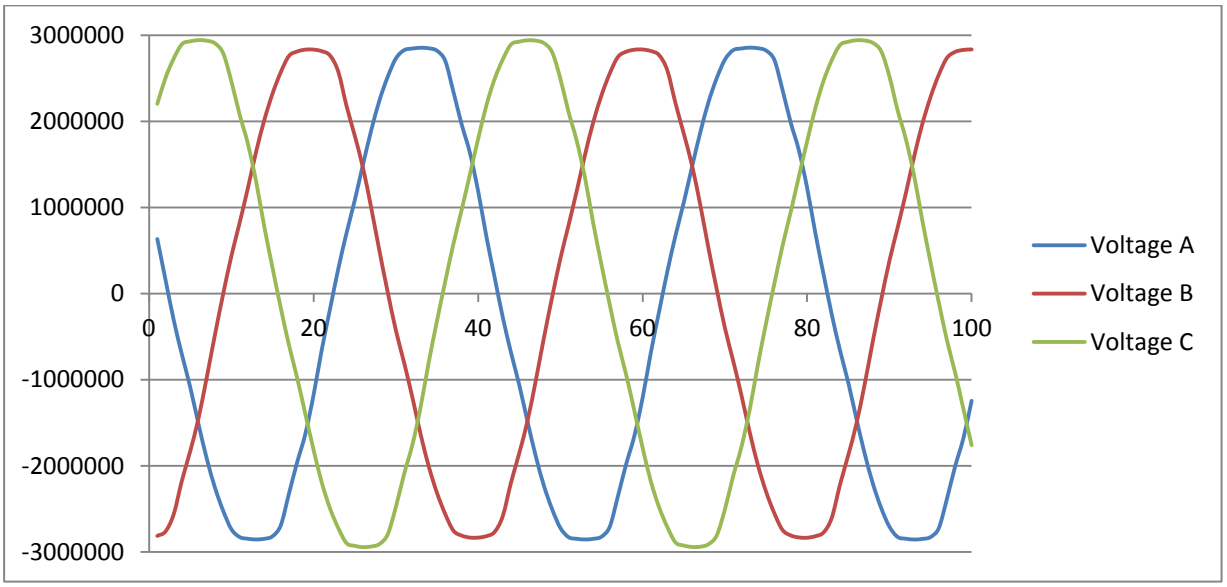


Figure 6.3 Three Phase Voltage Waveforms. This figure shows unit-less voltages relative to time. The time axis refers to sample number.

Table 6.2 RMS Voltage

	Voltage 1	Voltage 2	Voltage 3
Voltage in Volts RMS	200.1140	197.7679	200.1520

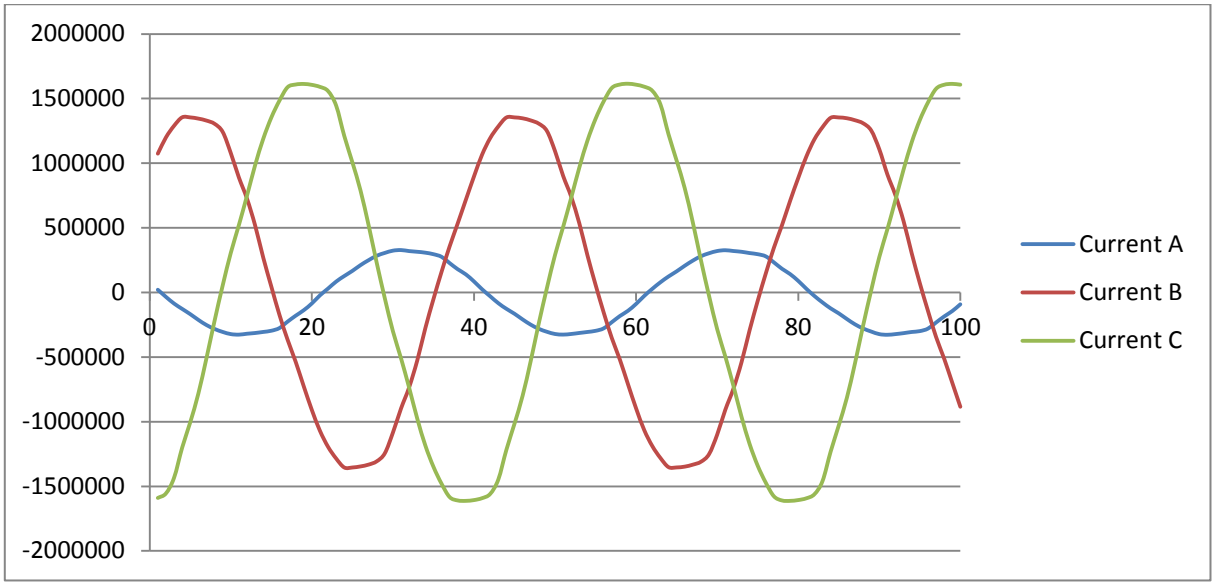


Figure 6.4 Three Phase Current Waveforms. This figure shows unit-less currents relative to time. The time axis refers to sample number.

Table 6.3 RMS Current

	Current 1	Current 2	Current 3
Current in Amperes RMS	0.79732	3.29135	3.87166

Table 6.4 Resistive Loads

	Phase 1	Phase 2	Phase 3
Resistance in ohms (Ω)	293	71	57

Table 6.5 New Power Calculations

P (W)	S (VA)	Q (VAR)	PF
-602.735	1942.666	1846.798	-0.24473