CAPE PENINSULA UNIVERSITY OF TECHNOLOGY

Design and development of a battery cell voltage monitoring system.

Master's Thesis

Nick Prinsloo

4/1/2011

A Thesis for The Department of Electrical Engineering in fulfilment of the requirements for the Magister Technologiae degree in electrical engineering, at the Cape Peninsula University of Technology.

Thesis

Design and development of a battery cell voltage monitoring system.

Nick Prinsloo

Supervisor: Dr Ian De Vries

A Thesis

for

The Department of Electrical Engineering in fulfilment of the requirements for the

Magister Technologiae degree in electrical engineering,

at the Cape Peninsula University of Technology.

Friday, 1 April 2011

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

Declaration

I, Niklaas Antonie Prinsloo, declare that the contents of this thesis represent my own unaided work, and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology or any of the bursary providers.

Signed

Date

Abstract

The purpose of this thesis was to design and develop a measurement system that would allow accurate measurement of individual cell voltages in a series cell stack.

The system was initially proposed to be used in conjunction with an active cell balancer. This would allow for the efficient equalising of cells as well as provide detailed information on the cell stack and how the stack operates over time. Having a system that measures voltages accurately, with which the active cell balancer can be controlled would allow for peak cell lifetime and performance.

Current battery management systems are large, complex and inefficient and a new way of battery management had to be investigated.

To accurately measure individual cells in a series stack, the high common mode voltage must be negated. Different techniques that are currently used to create galvanic isolation were reviewed; circuits were designed and were simulated to find the most suitable design.

The traditional methods used to create galvanic isolation did not provide adequate results. The methods were too inefficient and not accurate enough to be used. The methods that had the required accuracy were too complicated to connect in a useable system.

This led to the investigation of integrated circuits created to measure voltages in large cell stacks. An integrated circuit from Linear Technology was chosen and a system was built. A system was thus designed that fulfilled the most desirable design specifications while delivering excellent results.

The system allowed accurate, individual voltages to be measured in the presence of high common mode voltages. Accuracies and measurement time were well below the required system specification. Power consumption was high, but different component choice will lower power consumption to within specification. Excellent results were obtained overall with most, although not all results well below the design specifications.

By including current measurements, as well as other technologies such as wireless communication, USB connectivity and a better data processor, this system will be at the forefront of current battery management technology.

Acknowledgements

I wish to thank:

My parents who have supported me throughout the duration of my studies. Without their help and support I would not have been able to achieve this personal goal.

Dr Ian de Vries for his guidance, direction and wisdom. His supply of ideas helped me broaden the scope of my thinking and put me on the right track.

Cape Peninsula University of Technology who provided a clean and safe studying environment.

Johan Wagener and Daniel O'Connell who were always available for discussions and input.

The financial assistance of the National Research Foundation and Eskom towards this research is acknowledged. Opinions expressed in this thesis and the conclusions derived at are those of the author and do not necessarily represent that of the National Research Foundation or Eskom.

List of Figures

Figure 1: Series cell stack that show different voltage levels with respect to different reference levels. 12
Figure 2: Measurement device connected in parallel with an individual cell results in different ground potentials for the device..... 13
Figure 3: Block diagram of a complete battery management system (Bergveld, 2002). 17
Figure 4: Classification of batteries (Aponte, Andujar & Schroder, 1996). 18
Figure 5: Ni-Cd and Ni-MH cell discharge characteristics (Williams, 2005). 19
Figure 6: Li-Ion cell discharge characteristics (Williams, 2005). 20
Figure 7: Energy densities of various cells (Aponte, Andujar & Schroder, 1996). 21
Figure 8: Block diagram of the ISO120 isolation amplifier from Burr Brown (Burr Brown, 1992). 24
Figure 9: A system that would allow multiple isolation amplifiers to be connected to a single controller through multiplexing (Van Putten, 1996). 25
Figure 10: Optocoupler circuit with transresistance and transconductance amplifiers (Band & Unguris, 1997). 26
Figure 11: Dual optocoupler in photovoltaic mode with compensation circuitry to linearise measurements (Band & Unguris, 1997). 27
Figure 12: Redesigned single photoconductive mode optocoupler circuit used for tests. 27
Figure 13: Optocoupler circuit with operational amplifier and gain compensation through resistors. 28
Figure 14: Simulated results of the single optocoupler. 28
Figure 15: Simulated output voltage subtracted from actual output voltage. 29
Figure 16: Measured and actual results of the single optocoupler. 29
Figure 17: Measured voltage subtracted from actual voltage of the single optocoupler. 30
Figure 18: Simulated results of the single optocoupler. 30
Figure 19: Simulated voltage subtracted from actual voltage of the single optocoupler. 31
Figure 20: Optocoupler with dual integrated phototransistors. 32
Figure 21: Dual optocoupler design 33
Figure 22: Dual optocoupler circuit used to increase accuracy over the single optocoupler circuit. ... 33
Figure 23: Simulated results of the dual optocoupler 34
Figure 24: Simulated voltage subtracted from actual voltage of the dual optocoupler. 34
Figure 25: Measured and actual results of the dual optocoupler. 35
Figure 26: Measured voltage subtracted from actual voltage of the dual optocoupler. 36
Figure 27: Transformer used to isolate the cell from ground while still allowing the cell to be measured (Williams & Thoren, 2008)..... 37
Figure 28: Isolation transformer circuit being supplied by a 1 kHz current pulse to measure a 2V DC source. 38
Figure 29: Waveform showing the transformers primary voltage plus the diode voltage drop and the error added by the transformer over a 0V to 5V input. 38
Figure 30: Simulated voltage subtracted from actual voltage of the isolation transformer. 39
Figure 31: DC-to-DC converter which can be used to measure individual cells (Wagener, 2009)..... 40
Figure 32: Circuit used for simulating the DC-to-DC converter (Wagener, 2009). 41
Figure 33: Simulated results of DC-to-DC converter. 42
Figure 34: Simulated voltage subtracted from actual voltage of DC-to-DC converter. 42
Figure 35: Measured and actual results of the DC-to-DC converter circuit. 43
Figure 36: Measured voltage subtracted from actual voltage of DC-to-DC converter. 43

Figure 37: Internal block diagram of the MAX11068 IC by Maxim (Maxim, 2008). 46
Figure 38: Internal block diagram of the ATA6870 IC by Atmel (Atmel, 2009). 47
Figure 39: Delta sigma ADC sampling at 1M samples per second with a throughput of 1k samples per second (Kultgen, 2009). 48
Figure 40: SAR ADC sampling at 1M samples per second with an effective throughput of 1k samples per second (Kultgen, 2009). 48
Figure 41: A single pole RC and a finite input response (FIR) filters attenuation shown and marked at 10 kHz (Kultgen, 2009). 49
Figure 42: Internal block diagram of the LTC6802 IC by Linear Technology Corporation (Linear Technology Corporation, 2009). 51
Figure 43: Master connected to three slaves with communication through the SPI protocol (Philips, 2003). 58
Figure 44: Master and a slave connected on an I^2C bus (Philips, 2003). 59
Figure 45: Bottom three cells with filters and simulated internal switches of the LTC6802. 63
Figure 46: Top cell's (V1) discharge switch activated. 63
Figure 47: Middle cell's (V2) discharge switch activated. 64
Figure 48: Cell 2s (V2) voltage disconnected. 65
Figure 49: Schematic of the development board for ATmega16. 66
Figure 50: Development board used for tests. 67
Figure 51: LTC integration circuit with microcontroller, indication LEDs, power circuit and RS232 IC. 70
Figure 52: Bottom LTC IC with connector and filter section. 71
Figure 53: Top LTC IC with connector and filter section. 72
Figure 54: Cell simulator schematic used to simulate different cell type chemistries. 73
Figure 55: Prototype board without cell simulators 74
Figure 56: Prototype board with two twelve cell simulators connected. 74
Figure 57: Full simplified flowchart of C program. 75
Figure 58: USART transmit function that transmits data from the data register via the USART peripheral. 76
Figure 59: SPI transmit function that transmits and receives data simultaneously via the onboard SPI peripheral. 76
Figure 60: EEPROM write function that writes data to a specific address in the EEPROM. 77
Figure 61: EEPROM read function that reads a value from a certain address and returns the value to the caller. 78
Figure 62: Process that reconstructs even cell voltages and then compare them to new values to see if they should be written to the EEPROM. 79
Figure 63: Process that reconstructs odd cell voltages and then compare them to new values to see if they should be written to the EEPROM. 80
Figure 64: Process that checks if discharge switch needs to be enabled and if the switch can be enabled. 81
Figure 65: Serial Communicator showing values of cell 1 through 12 (with 6 cell values visible). 92
Figure 66: Difference in measured voltage minus actual voltage for cell 1. 93
Figure 67: Measured voltage variation with stable input voltage for cell 1. 93
Figure 68: Difference in measured voltage minus actual voltage for cell 5. 94
Figure 69: Measured voltage variation with stable input voltage for cell 5. 94
Figure 70: Difference in measured voltage minus actual voltage for cell 11. 95
Figure 71: Measured voltage variation with stable input voltage for cell 11. 95

Figure 72: 48 Measurements of arbitrary voltages created with the cell simulator and logged with the bottom most IC. 96
Figure 73: Measured voltage variation with stable input voltage for cell 1. 97
Figure 74: Measured voltage variation with stable input voltage for cell 1. 98
Figure 75: Voltage of cell 5 with an increase in chip temperature from 37°C to 80°C. 99
Figure 76: Screenshot of the GUI created to view cell voltages, chip parameters and additional information. 100
Figure 77: Actual voltage of 24 individual lead acid cells. 101
Figure 78: 24 Lead acid cells voltages displayed between 2V and 2.6V. 102
Figure 79: Voltage variation from ideal of 2.25V for a 24 cell stack. 102
Figure 80: 2D PCB of development board topside. 118
Figure 81: 2D PCB of development board bottom. 118
Figure 82: 2D Top view of PCB 119
Figure 83: 2D Bottom view of PCB 119
Figure 84: Cell simulator board used to simulate twelve cells of different chemistries. 120
Figure 85: Prototype board with the cell simulators placed in appropriate places as it would be used when testing is done. 120

List of Tables

Table 1. Desired system specifications. 14
Table 2: Gravimetric and volumetric density of Ni-MH, Ni-Cd and Li-Ion cells (Aponte, Andujar & Schroder, 1996) (Bergveld, 2002). 21
Table 3: Advantages and disadvantages of Ni-Cd, Ni-MH and Li-Ion cells (Microchip Technology Incorporated, 2007). 22
Table 4: PIC16F87XA series specifications (Microchip, 2003). 54
Table 5: ATmegaX series specifications (Atmel, 2002). 54
Table 6: MSP430X22X4 series specifications (Texas Instruments, 2009) 55
Table 7: Three different architectures compared with respect to code size and execution time. 56
Table 8: Current draw specification and pricing for the PIC, Atmel and MSP (Atmel, 2002) (Texas Instruments, 2009) (Microchip, 2003) 56
Table 9: Available designed protocols with their specifications (Philips, 2003) (Mathivanan, 2007) ... 60
Table 10: Actual specification for a 12 cell measurement system, taken for worst case scenarios. .. 104
Table 11: Table showing desired specification against actual specifications of the complete system. 104
Table 12. Table of comparison between the LTC, MAX and ATA BMS ICs (Andrea, 2009). 115

Glossary

Ah: [Ampere Hour]A specification of how much current the cell or battery can deliver to a load for an hour before the voltage drops to 0% SOC.

Battery: A certain amount of cells connected either in series, to form a higher stack potential, or parallel to increase current supply capability.

BMS: [Battery Management System] A system that does all the necessary battery management (charging, monitoring and balancing) and either displays or logs the results.

C: The C rate is numerically equal to the Ah rate. Discharge and charge currents are expressed as fractions of the C rate.

Cell: An electro-chemical device capable of supplying energy to a load as a result of internal chemical reactions

Cell Reversal: If a cell in a series stack is over discharged and current is forced through the cell the polarities of the cell is reversed.

ESR: [Equivalent Series Resistance] The internal resistance of any cell that limits the peak current into and out of the cell, as well as the element that generates heat when current flows into or out of the cell.

Float Charge: A voltage applied to the cell stack at the end of the charging cycle to negate the effects of the cells characteristic self-discharge.

Galvanic Isolation: Isolation that allows data to be transferred across an isolation barrier while preventing current and power flow.

GED: [Gravimetric Energy Density] The measure of how much energy a cell contains with respect to its weight.

SOC: [State of Charge] The measure of charge in the cell, based on the known potential difference across the terminals of a cell.

VED: [Volumetric Energy Density] The measure of how much energy a cell has with respect to its volume.

Table of Contents

Declaration iii

Abstract iv

Acknowledgements v

List of Figures 1

List of Tables 4

Glossary 5

Table of Contents 6

1 Chapter One: Introduction 11

1.1 Statement of Research Problem 11

1.2 Background to the Research Problem 12

1.3 Assumptions 13

1.4 Specifications 14

1.5 Objectives of Research 14

1.6 Delineation of the Research 15

1.7 Expected Outcomes, Results and Contributions of the Research 15

2 Chapter Two: Literature Review 16

2.1 Battery Management Systems 16

2.1.1 Definition of a Battery Management System 16

2.1.2 A General Battery Management System 16

2.1.3 The Need for a New BMS 17

2.2 Cell Types: Overview and Considerations 18

2.2.1 Lead Acid 19

2.2.2 Ni-Cd (Nickel Cadmium) 19

2.2.3 Ni-MH (Nickel Metal Hydride) 20

2.2.4 Li-Ion (Lithium Ion) 20

2.2.5 Gravimetric and Volumetric Density 21

2.2.6 Summary 22

3 Chapter Three: Methods Commonly Used for Galvanic Isolation. 23

3.1 Isolation Amplifier: 23

3.1.1 Theory 23

3.1.2 Circuit Design 24

3.1.3 Simulation 25

3.1.4 Summary 25
3.2 Single Optocoupler 26
3.2.1 Theory 26
3.2.2 Circuit Design 27
3.2.3 Simulation 28
3.2.4 Results 28
3.2.5 Summary 31
3.3 Dual Optocoupler 32
3.3.1 Theory 32
3.3.2 Circuit Design 33
3.3.3 Simulation 33
3.3.4 Results 34
3.3.5 Summary 36
3.4 Isolation Transformer 37
3.4.1 Theory 37
3.4.2 Circuit Design 37
3.4.3 Simulation 38
3.4.4 Results 38
3.4.5 Summary 39
3.5 DC to DC Converter 40
3.5.1 Theory 40
3.5.2 Circuit Design 40
3.5.3 Simulation 41
3.5.4 Results 42
3.5.5 Summary 44
3.6 Conclusion 44
4 Chapter Four: Stackable Battery Management ICs 45
4.1 Introduction 45
4.2 Maxim (MAX11068) 45
4.3 Atmel (ATA6870) 46
4.4 Linear Technology Corporation (LTC6802-1) 48
4.4.1 Delta Sigma ADC 48
4.4.2 High Voltage Multiplexer 49
4.4.3 Discharge Switches 49
4.4.4 Communication Protocol 50

4.4.5 Watchdog Timer 50
4.4.6 Internal Temperature Measurement 50
4.5 Conclusion 51
5 Chapter Five: System Considerations 52
5.1 Microcontrollers 53
5.1.1 PIC 16F877A 53
5.1.2 AVR ATmega16 54
5.1.3 TI MSP430F2254 54
5.1.4 Summary 56
5.2 Communication Protocols 57
5.2.1 SPI 57
5.2.2 I^2C 58
5.2.3 USART 59
5.2.4 USB 59
5.2.5 Summary 60
5.3 Programming Language 61
5.3.1 Assembler 61
5.3.2 C 61
5.4 Programming Interface 61
5.4.1 AVR Studio 4.17 61
5.4.2 Win AVR 20090313 61
5.5 Programming of the ATmega16 62
5.6 Cell Simulator 62
5.7 Simulation of Discharge Switch Tests 63
5.8 Development Board 65
5.9 Data Logging History 67
5.9.1 Instantaneous data 67
5.9.2 Graphical User Interface 68
5.9.3 Saved data 68
5.10 Conclusion 68
6 Chapter Six: Design 69
7 Chapter Seven: Programming 75
7.1 Flowcharts 75
7.2 Programming Code 82
7.2.1 Hex Code 82

7.2.2 C Code 82
7.2.3 Visual Basic Code 89
8 Chapter Eight: Results 92
8.1 Cell Measurement Results with Cell Simulator 92
8.1.1 Testing Criteria 92
8.1.2 Measurement and supply devices 92
8.1.3 Testing Procedure 92
8.1.4 Test Results for Cell 1 93
8.1.5 Test Results for Cell 5 94
8.1.6 Test Results for Cell 11 95
8.1.7 Conclusion 95
8.1.8 Test Results for all 12 Cells 96
8.2 Cell Measurement Results with 24 Cell Lead Acid Stack 97
8.2.1 Test Results for Cell 1 97
8.3 Cell Measurement Results while Charging 98
8.3.1 Test Results for Cell 1 98
8.4 Heat Test 99
8.4.1 Heat Tests on Bottom LTC IC 99
9 Chapter Nine: Graphical User Interface 100
9.1 GUI 100
9.2 GUI Results and Explanations 101
9.3 Conclusion 103
10 Chapter Ten: Conclusion 104
10.1 Discussion of Problems Encountered 105
10.1.1 Isolation amplifier 105
10.1.2 Optocoupler 105
10.1.3 DC to DC Converter 105
10.1.4 Transformer Isolation 105
10.1.5 Battery management ICs 105
10.1.6 LTC6802-1 105
10.1.7 System Design 106
10.2 Future Design 107
10.2.1 Microcontroller 107
10.2.2 Microcontroller or FPGA 108
10.2.3 Programmable Components 108

10.2.4 Isolation 108
10.2.5 Wireless Communication 108
10.2.6 Integration 109
10.2.7 Power Measurement 109
10.3 Conclusion and Closing Remarks 110
10.3.1 Objectives of Research 110
10.3.2 Delineation of the Research 111
11 References 112
12 Appendices 115
12.1 Appendix A: BMS Chip Comparisons 115
12.2 Appendix B: Development Board 118
12.3 Appendix C: Prototype Board 119
12.4 Appendix D: Programming Code 121
12.4.1 C Code 121
12.4.2 Visual Basic Code 150

1 Chapter One: Introduction

Batteries are used daily to provide power, mobility and backup to equipment. The ability of cells to perform these tasks depend on their charged status. Cells are manufactured, according to their chemistry, to produce a certain amount of ampere hours (Ah) at a rated voltage. The closer cells are to their peak charged levels the better they will perform the required work, and for increased periods.

Measurement of cells in a battery stack is important for the following reasons:

1. The present state of the stack can be monitored.
2. When charging the stack, the voltage has to be known so that overcharging does not occur and when discharging the stack so that over discharging does not occur.
3. Accurate voltage measurements of individual cells will allow the use of equalisation techniques to balance cells.
4. By measuring and logging data the overall health of the stack can be calculated and future calculations can be made.

When measuring a cell in a high voltage stack a reference is created specific to the measured cell. This reference point is at a different voltage than the system ground. This is caused by the common mode voltage which rises as the stacked cell count is increased as well as the position of the measured cell in the stack.

The output has to be isolated from the input to protect the measurement device and the user from high voltages and potential shock risk (Van Overschee, 1998).

Battery management systems aid this process. They are designed to charge, measure and display or store the cell information. To measure the cells they incorporate galvanic isolation techniques and numerous methods to do this exist. This can be done either through transformer or optical based methods. These BMSs are expensive, complex and bulky. A smaller and more economical method is necessary to measure individual cells in a large series stack.

1.1 Statement of Research Problem

Measurement of individual cells in a high voltage DC series connected cell stack is complicated by the common mode stacked voltage with reference to ground. It is necessary to negate the common mode voltage through isolation which will allow the individual cell voltages to be accurately measured with regard to system ground.

1.2 Background to the Research Problem

High voltage batteries are used in the telecommunications, aviation and motor industries. These batteries provide power or backup power to the different systems connected to them.

Measurement of a low value DC voltage is easily measured using specific integrated circuits (ICs). When an individual cell voltage in a high cell count battery stack needs to be measured it cannot be done with these ICs due to the high stacked common mode voltage with respect to the referenced ground of the system (Ibanez & Dixon, 2004) (Williams & Thoren, 2008).

What is more, the system is further complicated due to the fact that the initial prototype and the final design will require the measured information to be in digital form. This information must be integrated into an electronic system for analysis, comparisons and data acquisition.

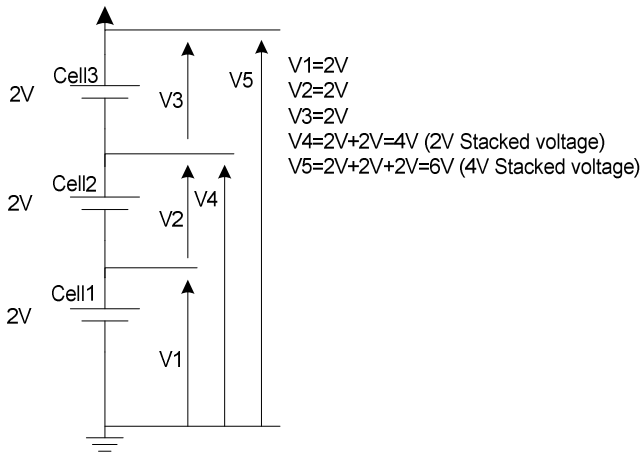

Figure 1: Series cell stack that show different voltage levels with respect to different reference levels.

In the series stack (figure 1) it is illustrated how the voltage is increased when more cells are added to the stack.

The second cell in the stack has a potential of 4V at the cell's positive terminal. When the second cell is measured, the bottom of the cell, the negative terminal, is taken as the ground reference point, but the voltage is at a potential of 2V with regard to the systems ground.

Even though the cell has to be measured with respect to this reference point, the system ground is at a lower potential. When a measurement is taken between the cell ground and system ground a voltage difference exist and as a result, 2V will be measured (Morrison, 1997).

Figure 2 illustrates the common mode stacked voltage that is measured between the reference ground and system ground. When the measured cell is allocated higher in the stack this common mode stacked voltage increase.

The ability of a system to resist this common mode voltage is called the systems common mode rejection ratio (CMRR). Operation near or beyond these limits will cause inaccurate measurements or damage to the measurement device (Van Overschee, 1998).

By isolating the measurement device from the measured cell the CMRR can be increased.

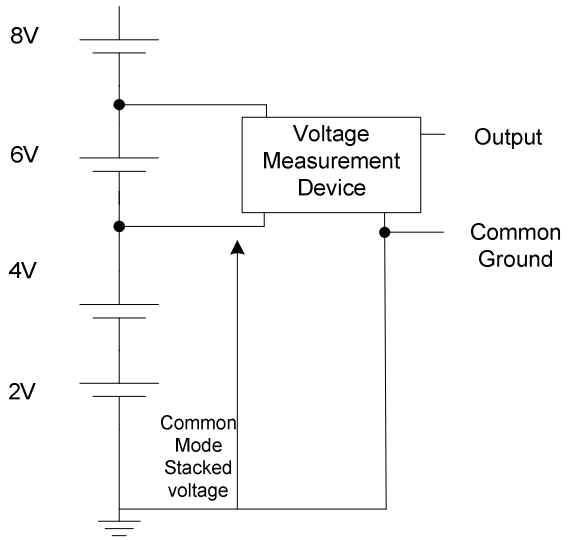

Figure 2: Measurement device connected in parallel with an individual cell results in different ground potentials for the device.

Isolation is also needed where ground loops exists. If the circuit has multiple ground points and their potentials are not at the same voltage, a current flows from the higher potential to the lower potential. This leads to the common mode voltage being added to the output of the measurement device.

Ground loops are a source of noise in measurement systems and can contain possible damaging and lethal voltages (Van Overschee, 1998).

1.3 Assumptions

The following assumptions were made during this research:

1. The system is designed to work with an active cell balancer. However, complete operation of the prototype system needs to be demonstrated as a stand-alone system before integration to the cell balancer can be done.

2. Due to the equalisation technique used by the active cell balancer, all cell voltages will be equal unless the equalisation device is forced into current limiting (De Vries, 2008).
3. To form a complete BMS, the system is required to charge, measure and equalise cells. A charging circuit can be designed or done by a third party product and is not discussed here. Equalising is done by means of the passive balancing in the stand alone system. Individual cell measurement, communications, passive balancing and display system will be discussed in this thesis.

1.4 Specifications

The following desired specifications are based on set parameters after careful considerations of what features are required from a battery management system (BMS). The specification was determined from current battery management systems (LEM, 2010) (Power Shield, 2010) (Network Service Group LLC, 2010) (On-Line Monitoring Inc., 2010).

Table 1. Desired system specifications.

Parameter	Specified
Current Consumption.	≤ 1mA
Voltage Measurement Range.	0V to 5V
Individual Cell Overvoltage Range.	7V
Overall Voltage Measurement Accuracy.	≤ 0.25%
Main Isolation Barrier, if Isolated.	≥ 2kV
Measurement Time for all Cells.	< 100msec
Measurement Accuracy per Cell.	≤ 10mV
Cells in stack	≥ 24 cells
Desired final cost	≤ R2500

1.5 Objectives of Research

- Identify a suitable method to measure individual cell voltages in a string of twenty four cells.
- Analyse, simulate, build and test possible circuits to determine suitability as possible voltage measurement solution.
- Build a scalable prototype of final circuit and do tests on the chosen circuit design.
- Initial tests will be done on a twenty four cell stack, but should be suitable for any cell count.
- Integrate to a PC and log data through a graphical user interface (GUI).
- Investigate wireless capabilities to increase portability of the system.

1.6 Delineation of the Research

- Measure, display and log the individual voltages of a twenty four series cell stack.

1.7 Expected Outcomes, Results and Contributions of the Research

The research prototype will allow the user to measure the voltage of individual cells in a stack of 24 cells connected in series. The prototype will be small, easily integrable and energy efficient. The system will be of particular interest to industries named in section 1.2. Although numerous BMSs exist, the addition of the active cell balancer and the miniaturisation of the individual cell measuring devices will ensure premium stack operation.

The system will ensure that battery stack problems are detected early through preventative maintenance and can thus minimize overall downtime as well as increase system runtime while being highly energy efficient (Mulder et al, n.d.).

2 Chapter Two: Literature Review

2.1 Battery Management Systems

Due to the vast and extensive nature of current BMSs, a general overview will be favoured.

2.1.1 Definition of a Battery Management System

According to Bergveld (2002) the functions of a BMS include battery -, power - and energy management.

Battery management includes all functions that are required to ensure proper battery operation and use.

Power management includes functions that reduce power consumption by the system parts with respect to active hardware and software design changes e.g. lowering or powering components and signals down when not in use.

Energy management includes functions that ensure energy conversions in the system are at its lowest levels e.g. zero-voltage and zero-current switching in the battery charger.

The correct definition of a BMS is then:

A battery management system ensures that optimum use is made of the energy inside the battery powering the equipment while the risk of damage inflicted upon the battery is minimized. This is achieved by monitoring and controlling the battery's charging and discharging process.

2.1.2 A General Battery Management System

According to Bergveld (2002) the following needs to be taken into account when buying a BMS:

- The cost/additional cost of the BMS cannot outweigh the load the battery is powering. The load cost, which is the type of equipment being powered, as well as the reliability needs of the load will dominate the allowable price of the BMS.
- The type of load that will be driven by the battery dominates the type of BMS that will be used. Where space is minimal, a smaller or less complex BMS will be used. If however the battery is a room stack then the BMS's size is insignificant.
- If the battery stack is used regularly, then the type and quality of battery will be better than the battery of a load that will only be used as a backup.

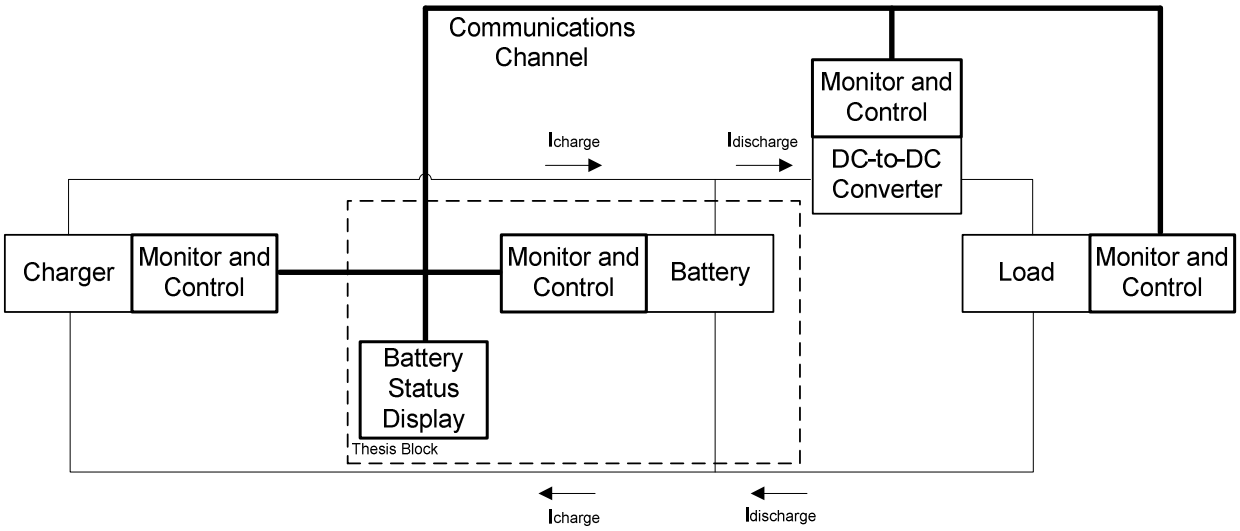

Figure 3: Block diagram of a complete battery management system (Bergveld, 2002).

Figure 3 shows the different parts of a BMS. The dashed outline section is what will be analysed and discussed in this thesis. Batteries will be discussed in section 2.2. Note that the term battery is used in figure 3 to signify a series stack of cells. The term battery stack or series cell stack will be used henceforth. Monitor and control will be discussed in sections 4.4 and 5.1 respectively and battery status display will be discussed in chapter 9.

2.1.3 The Need for a New BMS

Problems with current commercial BMS are the following:

1. Price, even simple units can be costly.
2. Size and complexity do not allow them to be installed in a small space.
3. Designed for specific cell stack size and cell chemistry.
4. Equalisation through passive balancing.

The design covered in this thesis will eliminate and minimise the shortcomings of current BMSs.

The first and most important part of a BMS, the cells, are discussed in the next section.

2.2 Cell Types: Overview and Considerations

The voltage measurement system was developed to ensure extended battery life and reliability by allowing the active cell balancer to be enabled and disabled and thus allowing proper equalisation of the battery stack without any user input. The cells and the effects of their different chemistries form an integral part of the voltage measurement system and of this thesis.

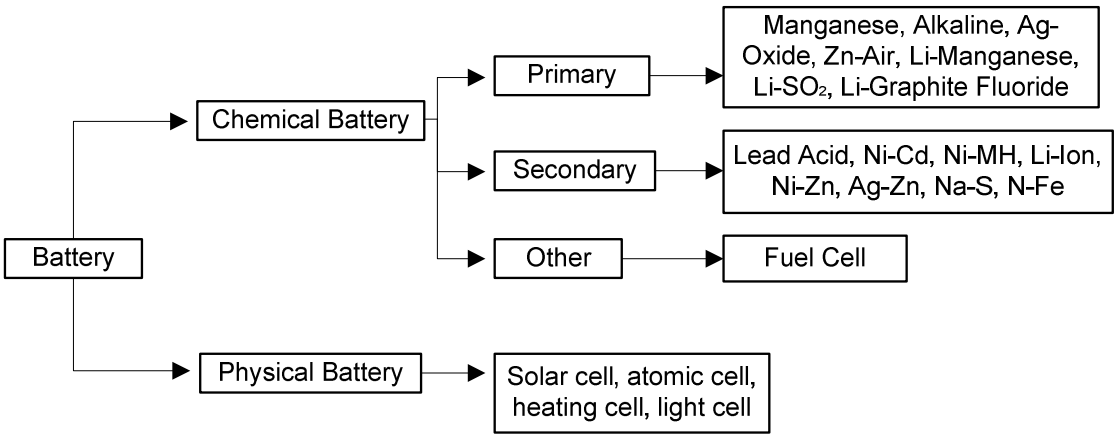

Figure 4: Classification of batteries (Aponte, Andujar & Schroder, 1996).

Batteries, comprised of series stacked cells, are divided into different categories as shown in figure 4. Of those in figure 4 only secondary cells will be focused on in this thesis.

Three types of rechargeable cells are mainly used in contemporary systems. They are Lead Acid, Nickel Metal Hydride (Ni-MH) and Lithium Ion (Li-Ion). Nickel Cadmium (Ni-Cd) used to be a popular choice but is now discontinued because of the poisonous Cadmium in the cells. A Ni-Cd cell will still be analysed as they were a popular choice for many systems and are still in use in installed systems.

Different types of cells are made of different chemical compositions and because of this the different cells have different voltage levels and different types of charging routines are needed. The state of charge (SOC) is reflected in the voltage and allows the charged state of the cell to be calculated (Perez, 1993) (Stevens & Corey, 1998) (De Vries, 2008) (Wagener, 2009).

Respective cell types have different discharge curves based on their chemical composition and the load that is being driven. Since the focus is only on the voltage measurement, and it is known that SOC is reflected in voltage levels (Perez, 1993) (Stevens & Corey, 1998), it will be shown how these cells reflect the SOC through their discharge curves. It must be noted that SOC is reflected in cell voltage, even if the cell is not loaded (Williams, 2005).

2.2.1 Lead Acid

As Lead Acid cells are common and have been well researched they will not be discussed. It must be noted that SOC measurement through voltage still applies to them and that heat and cold diminishes capacity as it does in all other cells. Slightly overcharging cells is called “gassing” and is used to balances the cells. This eliminates the need for special equalisation circuitry at the price of shortening the life of the cells through loss of internal chemicals.

2.2.2 Ni-Cd (Nickel Cadmium)

Nickel Cadmium cells have a fully charged terminal voltage of 1.2V and an open circuit voltage of 1.3V with a flat discharge curve as shown in figure 5. The discharge curve is generally assumed to be the same for a Ni-Cd and a Ni-MH cell and is shown as such in the figure.

Ni-CD cells are summarised as follows (Ratnakumar & Smart, 2007):

1 Having a flat discharge curve, these cells are ideally suitable to be used along with linear regulators.
2 Low equivalent series resistance (ESR) allow high peak and surge currents out of the cell without generating much heat inside the cell.
3 Self-discharge rate is high when compared to Li-Ion cells, but slightly lower than Ni-MH cells. The typical discharge rate is 15%-20% per month at 20°C.
4 Ni-Cd cells can be float charged at a rate of C/10 with no adverse effects on the cell.
5 Cells contain Cadmium which is poisonous and not environmentally friendly.
6 At high temperatures Ni-Cd cells lose capacity, which means that even though the cell is fully charged it will not deliver the charge it used to when it was new. Another drawback of Ni-Cd cells are that heat influences the cells’ ability to charge negatively. Cells are incapable of charging to their full capacity and charge efficiency is reduced. (Wright, 2006).

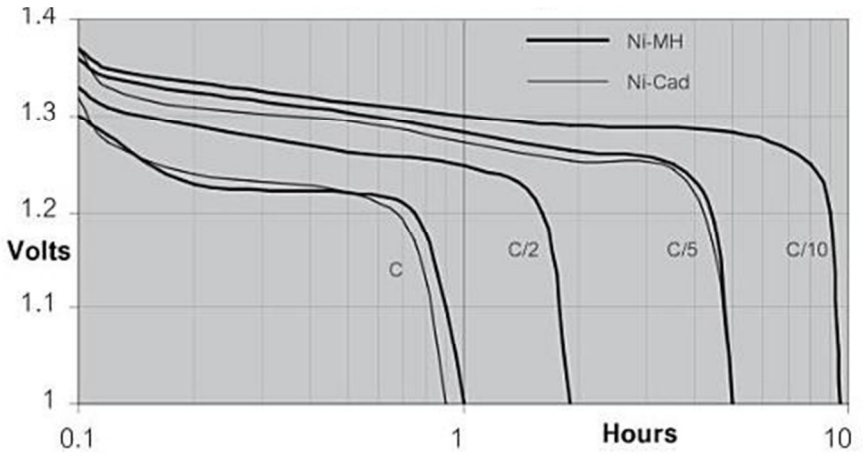

Figure 5: Ni-Cd and Ni-MH cell discharge characteristics (Williams, 2005).

2.2.3 Ni-MH (Nickel Metal Hydride)

Ni-MH cells are summarised as follows (Ratnakumar & Smart, 2007):

1. The fully charged terminal voltage of a Ni-MH cell is 1.2V, with an open circuit voltage of 1.3V and almost the same discharge curve as a Ni-Cd cell. The discharge curve can be seen in figure 5.
2. Ni-MH cells have the highest self-discharge rate of all the cells reviewed. Self-discharge rates of 20%-30% at 20°C per month can be expected.
3. Ni-MH cells are not as tolerant of constant charging as Ni-Cd cells are and have difficulty trickle charging, even at a charging rate of C/40.
4. Temperature influences Ni-MH cells in the same way as it does Ni-Cd cells (Wright, 2006).

2.2.4 Li-Ion (Lithium Ion)

Li-Ion cells are summarised as follows (Endo & Kim, 2003):

1 A fully charged Li-Ion cell has a terminal voltage of 3.6V and an open circuit voltage of 4.2V, which is the main advantage of Li-Ion cells. One Li-Ion cell has 3 times the voltage of a Ni-Cd or a Ni-MH cell and about 1.7 times more than a lead acid cell. The discharge curve can be seen in figure 6. The steeper discharge slope of the Li-Ion cell is advantageous due to the fact that it is easier to measure the cell's SOC with high accuracy. This is due to the large voltage difference from the fully charged to fully discharged state.

2 The ESR of a Li-Ion cell is not as low as a Ni-CD or Ni-MH cell, but do not impose any great current source or sink penalties.

3 Li-Ion cells have the lowest discharge rate of all the cells reviewed. Discharge rates of 5%-10% per month at 20°C can be expected.

4 Li-Ion cells cannot be trickle charged and can only be charged in constant voltage mode.

5 Li-Ion cells are not severely affected by temperature change, but at low temperatures cell capacity diminishes slightly (Wright, 2006) (Chen & Evans, 1995).

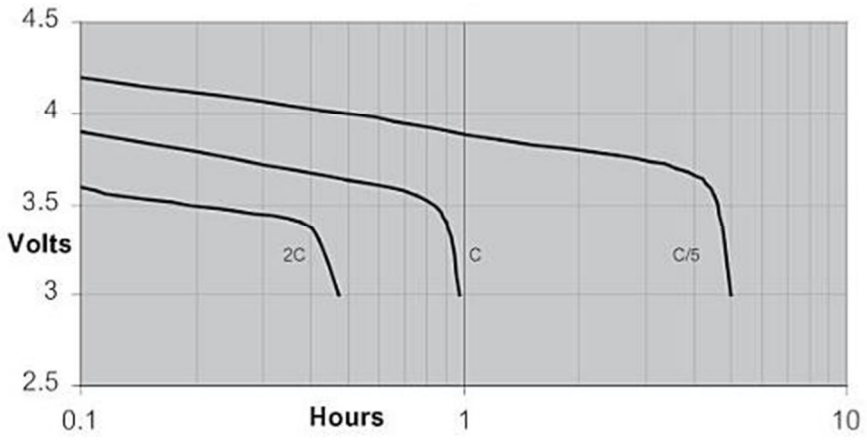

Figure 6: Li-Ion cell discharge characteristics (Williams, 2005).

2.2.5 Gravimetric and Volumetric Density

The last consideration is the power/weight and the power/volume ratios. Power/weight is normally expressed as watt hours per kilogram (Wh/kg) and power/volume in Watt hour per litre (Wh/l).

Table 2: Gravimetric and volumetric density of Ni-MH, Ni-Cd and Li-Ion cells (Aponte, Andujar & Schroder, 1996) (Bergveld, 2002).

Cell Type	Ni-MH	Ni-Cd	Li-Ion
Gravimetric Density(Wh/kg)	60-100	40-80	110-130
Volumetric Density(Wh/l)	180	140	210

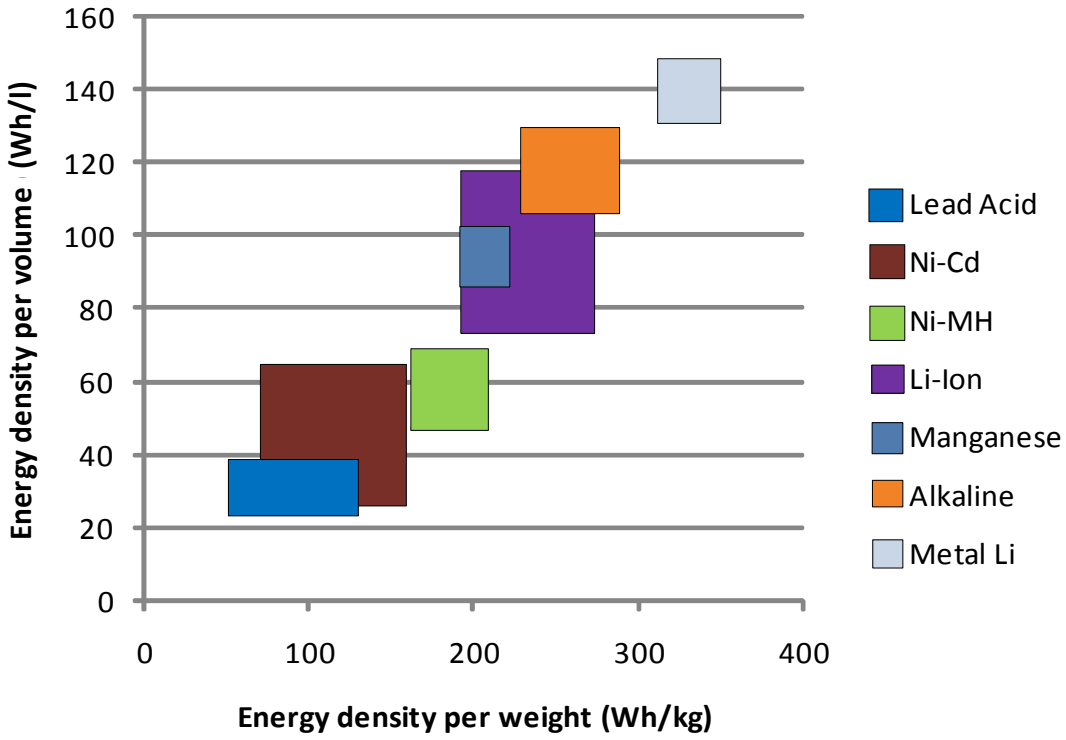

Figure 7: Energy densities of various cells (Aponte, Andujar & Schroder, 1996).

The gravimetric and volumetric densities determine where the different cell chemistries are used. If the equipment requires portability then the more expensive, higher density Li-Ion cells would be used e.g. electric vehicles (current pricing does not yet allow this). However, if the equipment is stationary all the time, the cheaper, lower density lead acid or Ni-MH cells could be used e.g. a battery room.

2.2.6 Summary

Table 3: Advantages and disadvantages of Ni-Cd, Ni-MH and Li-Ion cells (Microchip Technology Incorporated, 2007).

Battery	Advantages	Disadvantages
Ni-CD	High cycle life	Relatively low energy density
	Excellent load performance	Not environmentally friendly
	Economically priced	
Ni-MH	30%-40% Higher capacity than Ni-CD	Limited cycle life
	Environmentally friendly	Limited load performance
		High self-discharge
Li-Ion	High energy density	Requires protection circuitry
	Relatively low self-discharge	Subject to aging even when not in use
	Low maintenance	Expensive

Since the only measurable parameter is the voltage of the individual cells, the measurement device has to be extremely accurate. System specification for accuracy is set to lower than 10mV per cell and if this can be achieved a highly accurate system will be built.

3 Chapter Three: Methods Commonly Used for Galvanic Isolation.

Different types of cells have been discussed and the following section investigates potential methods to measure individual cell voltages.

The following topologies have been identified for investigation

- Isolation ICs
  - Isolation Amplifier
  - Optocoupler
- Isolation Transformer
- DC-to-DC Converter
- Stackable Battery Management ICs
  - MAXIM (MAX11068)
  - ATMEL (ATA6870)
  - LTC (LTC6802-1)

3.1 Isolation Amplifier:

3.1.1 Theory

Isolation amplifiers are commonly used for the following purposes (Band & Unguris, 1997):

1. To sense analog signals where the common mode voltage exceeds the supply voltage.
2. As analog safety isolators.
3. To break ground loops.

Isolation amplifiers contain an input that is separate from its output through galvanic isolation. The voltage information is transferred from the input to the output through three different methods: transformer, capacitive and optical coupling.

The voltage signal is converted to either a pulse width modulated (PWM) or a voltage dependant frequency signal which can be transferred over the isolation barrier. The PWM signal is generated through a modulation process and the voltage dependant frequency signal is generated by a voltage controlled oscillator (VCO) (Meyrath, 2005).

Some isolation amplifiers generate their own isolated power supply from an external power source through another isolation transformer. This is needed to supply the input circuitry with power while not loading the stack and thereby still keeping voltage isolation between the input and the output.

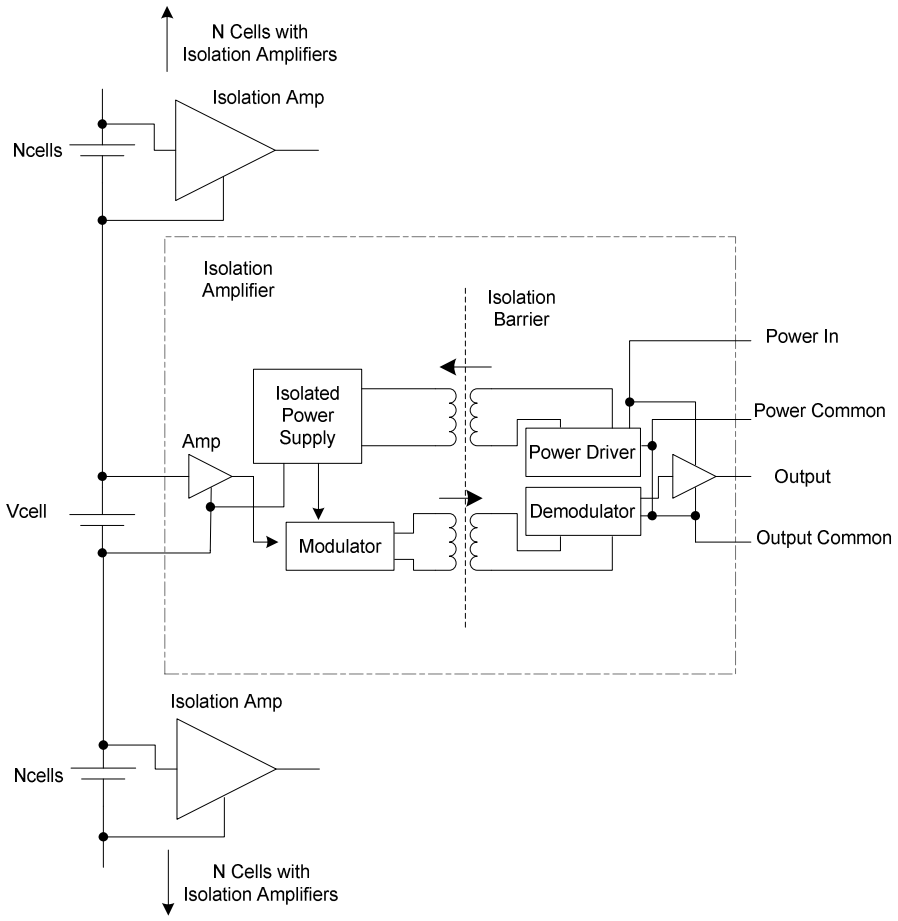

Figure 8: Block diagram of the ISO120 isolation amplifier from Burr Brown (Burr Brown, 1992).

3.1.2 Circuit Design

A potential system design is shown in figure 9. This system would allow twenty four isolation amplifiers to be connected so that twenty four cells can be measured.

At the isolation amplifiers output, which is galvanically isolated from the input, the cell voltage can be measured. These measured values have to be collected and temporarily stored in memory before it is passed onto an electronic system for display (Van Putten, 1996).

Numerous multiplexers and a controller would allow this system to be realised. Multiplexers range from 2:1 (two inputs, one output) to 16:1 (sixteen inputs, one output) with the output controlled from a code by a controller. If an 8:1 (eight inputs, one output) multiplexer is chosen, up to eight isolation amplifiers can be measured with one high accuracy analog to digital converter (ADC).

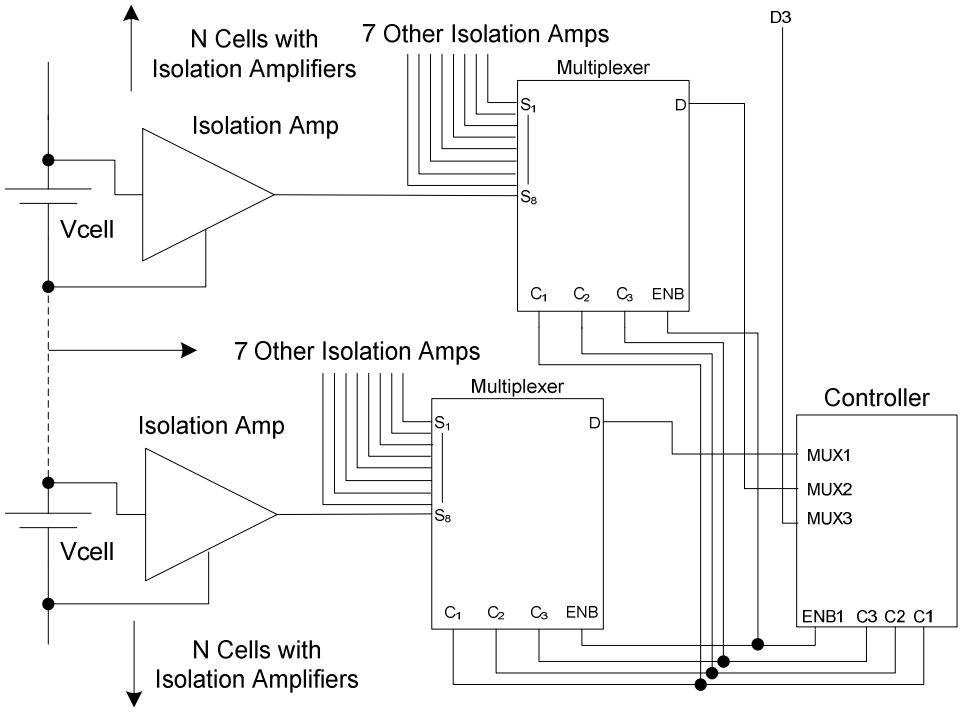

Figure 9: A system that would allow multiple isolation amplifiers to be connected to a single controller through multiplexing (Van Putten, 1996).

Stacking of individual blocks is done to increase the amount of cells that can be measured. The maximum voltage restriction is the isolation voltage of the isolation amplifier, which is normally around 5kV. Isolation amplifiers are fast, with current draw in the milliamps and accurate to 14 bits (Meyrath, 2005).

3.1.3 Simulation

No simulation will be done for isolation amplifiers due to the high cost and unsuitability as a system which will be explained in the next section.

3.1.4 Summary

Isolation amplifiers are expensive and complex which make them unsuitable for this design.

Although high accuracy and low current consumption makes isolation amplifiers desirable, high costs and difficulty in connecting them in a system that would allow twenty four individual cell voltages to be measured makes isolation amplifiers a non-feasible option.

3.2 Single Optocoupler

3.2.1 Theory

Unlike isolation amplifiers, optocouplers do not convert the potential difference at the input to a determined switching waveform. The input potential difference generates a constant current through a light emitting diode (LED) which radiates light onto a receiving phototransistor. Ideally the input current should be linearly proportional to the output current when the current transfer ratio (CTR) is one, but having the CTR stay linear over the full operating and temperature range is unlikely. The following problems are faced when using optocouplers:

1. The ratio between collector current and drive current of the LED is called the CTR. This should ideally be constant to allow for high accuracy measurements. This however, is not the case as mass fabrication and optocoupler packaging allow normal tolerances that can lead to the CTR having a gain of less than one to a gain of four.
2. The large junction area of the photodiode creates a parasitic capacitance between the base and collector of the phototransistor which decreases high frequency gain.

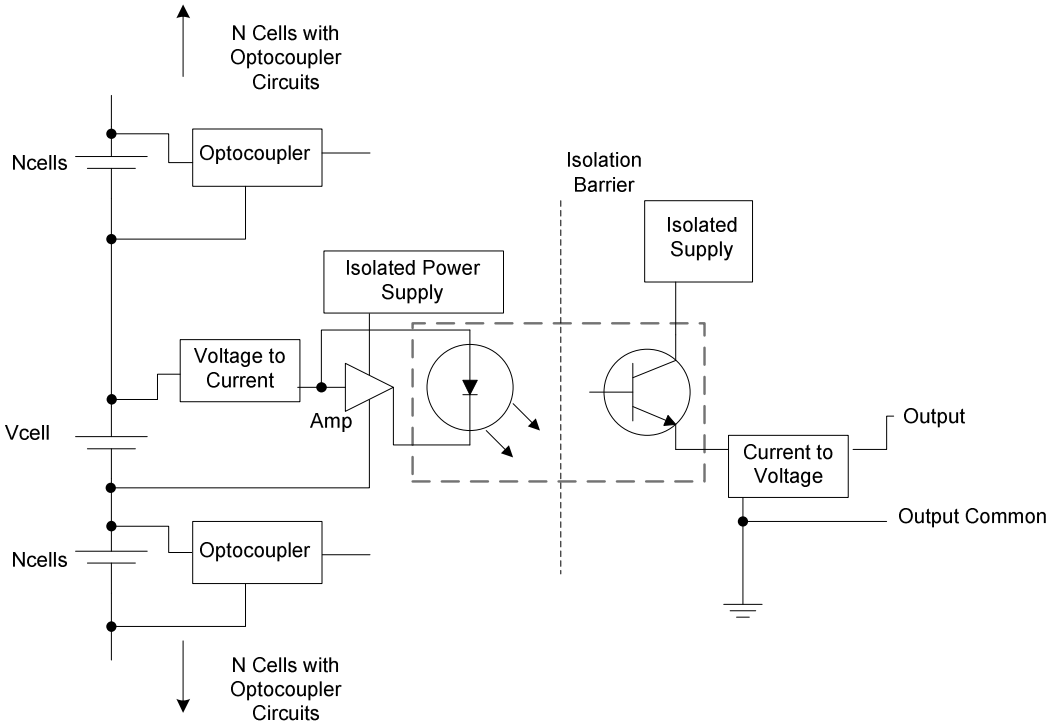

Figure 10: Optocoupler circuit with transresistance and transconductance amplifiers (Band & Unguris, 1997).

By using the optocoupler with an operational amplifier and incorporating feedback, non-linearity over the operating range can be minimized and the optocoupler can be linearised. The last cause of input to output non linearity mentioned above will not affect this system's performance as it is an AC parameter and the LED is operated from a cell which is a DC source.

Voltage is converted to a current which drives a LED in the optocoupler. The amount of flux that radiates from the LED is proportional to the forward current through the LED. The light is then transferred through the galvanic isolation barrier (which is free space) to the phototransistor on the output. At the output the current is converted back to a voltage, which is then multiplexed (as per the previous example) and measured by a controller (Ibanez & Dixon, 2004) (O'Loughlin, 2009).

3.2.2 Circuit Design

Some circuits were researched and none of them allowed a simple, low cost circuit to be used for the design. All of them required many optocouplers or operational amplifiers to linearise the circuit. This would have been ideal if only one cell (or any other application requiring galvanic isolation) had to be measured. Since twenty four cells need to be measured, a complex circuit will defeat the proposal of having an accurate measurement device of minimal size.

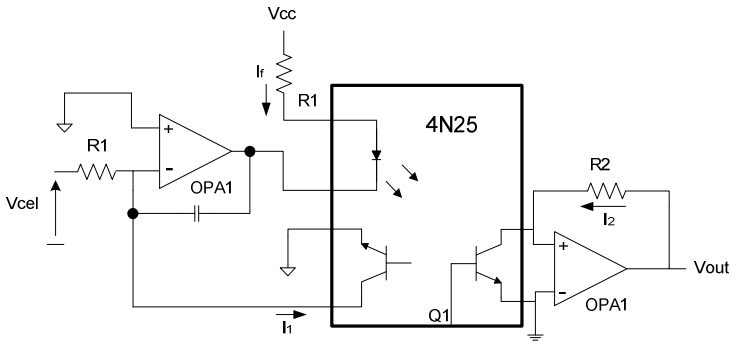

Figure 11: Dual optocoupler in photovoltaic mode with compensation circuitry to linearise measurements (Band & Unguris, 1997).

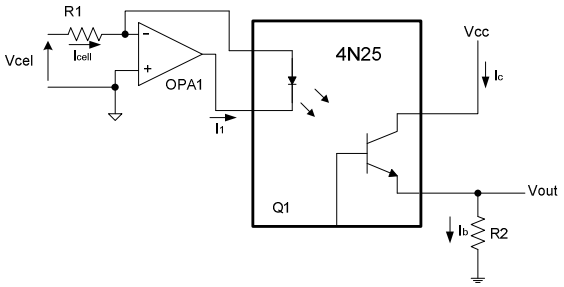

Figure 12: Redesigned single photoconductive mode optocoupler circuit used for tests.

The circuit in figure 11 was found and redesigned (Band & Unguris, 1997). The new circuit is shown in figure 12. This circuit is simulated in the next section with actual results compared to the simulation.

When a voltage is applied to OPA1s input the output of the opamp goes negative and current is sourced from the input through the LED. This ensures that a constant current flows through the LED which is directly proportional to the input voltage.

3.2.3 Simulation

Simulation results for this circuit were obtained by using LTSpice IV which was designed by Linear Technology Corporation.

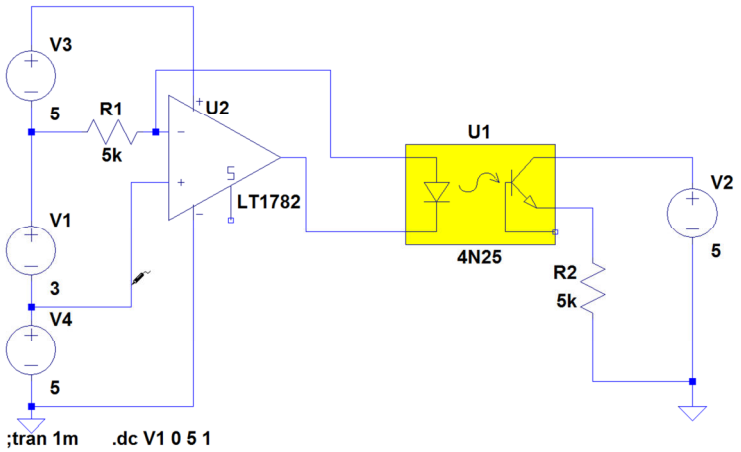

Figure 13: Optocoupler circuit with operational amplifier and gain compensation through resistors.

Figure 13 shows that V1 is a DC voltage of 3V, but the simulated parameters of V1 for the circuit were set as a DC sweep function. The function ramps from 0V up to 5V.

3.2.4 Results

Results for the optocoupler circuit can be seen in figure 14. Although the results are linear, the input to output voltage does not correlate well. This is due to the CTR of the optocoupler not being unity.

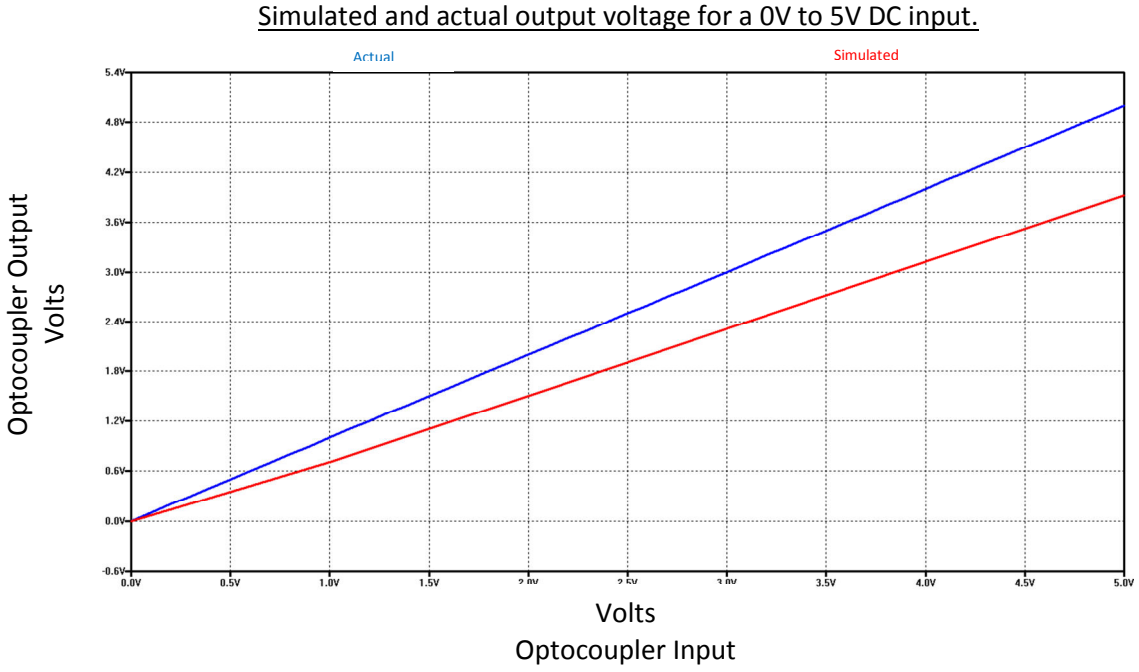

Figure 14: Simulated results of the single optocoupler.

The opamp linearised the circuit, but did not compensate for the non-unity CTR. This can be compensated for by introducing amplification in the output circuit. This poses a problem in that the different optocouplers required in building a system, each having a different CRT because of manufacturing differences, will each introduce measurement errors if the components are standardised for all optocouplers. Simulation results for output compensation are shown later.

Simulated voltage minus actual voltage for a 0V to 5V DC input.

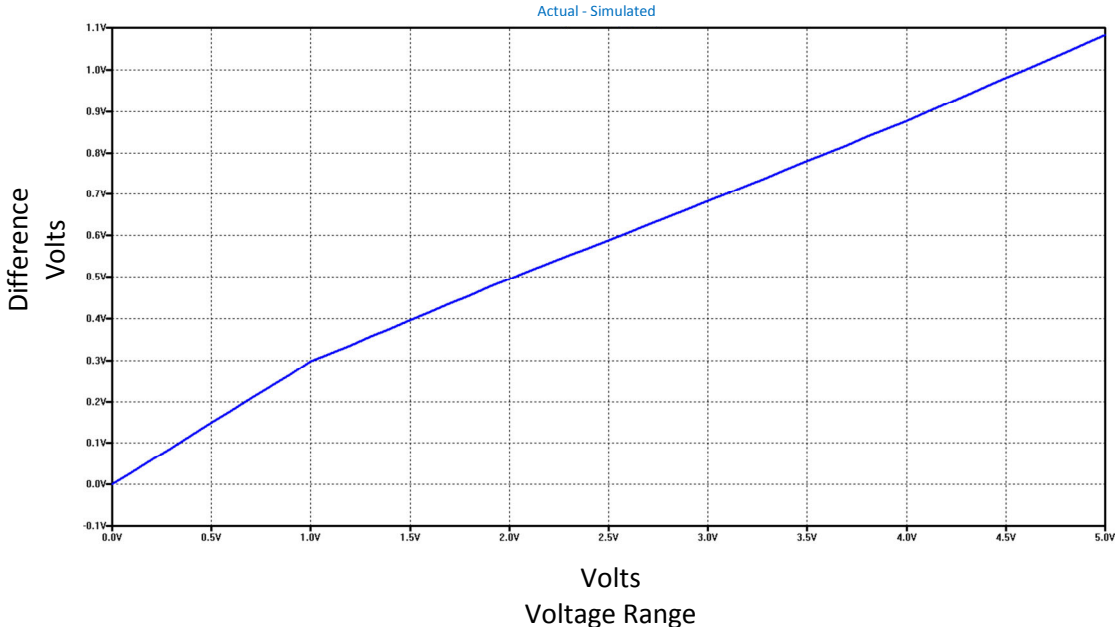

Figure 15: Simulated output voltage subtracted from actual output voltage.

The measurement accuracy, as can be seen in figure 15 is much lower than system specification. The circuit was built and tested to see how well the actual circuit will correlate with the simulation.

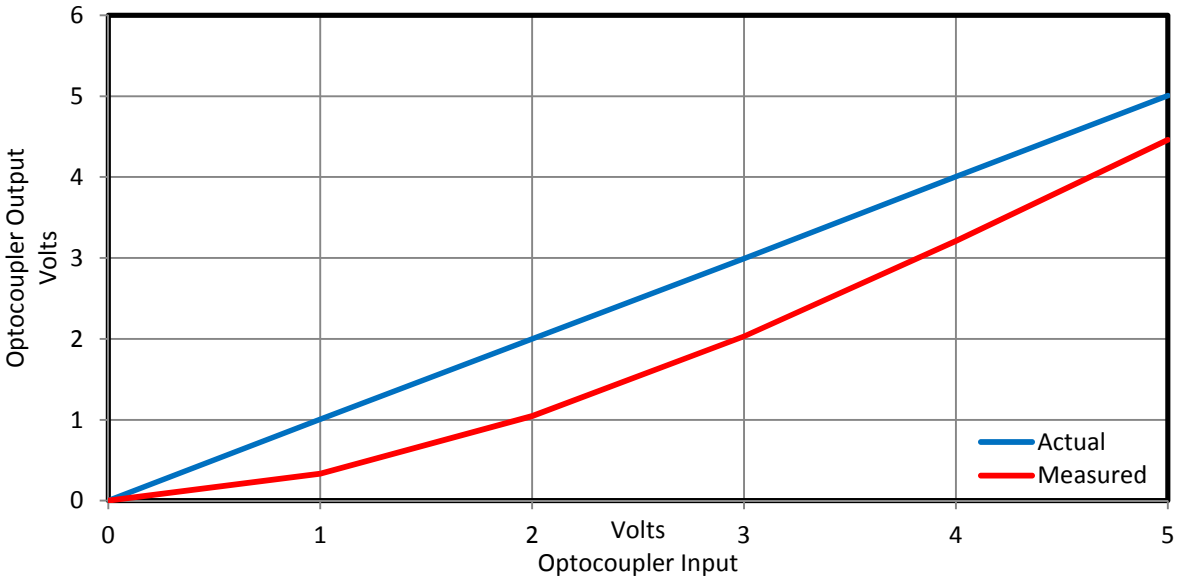

Figure 16: Measured and actual results of the single optocoupler.

Figure 16 shows the results of the prototype. The results do correlate with the simulated results, the main difference being in the linearity of the output. This can be seen in figure 17.

Measured voltage minus actual voltage for a 0V to 5V DC input.

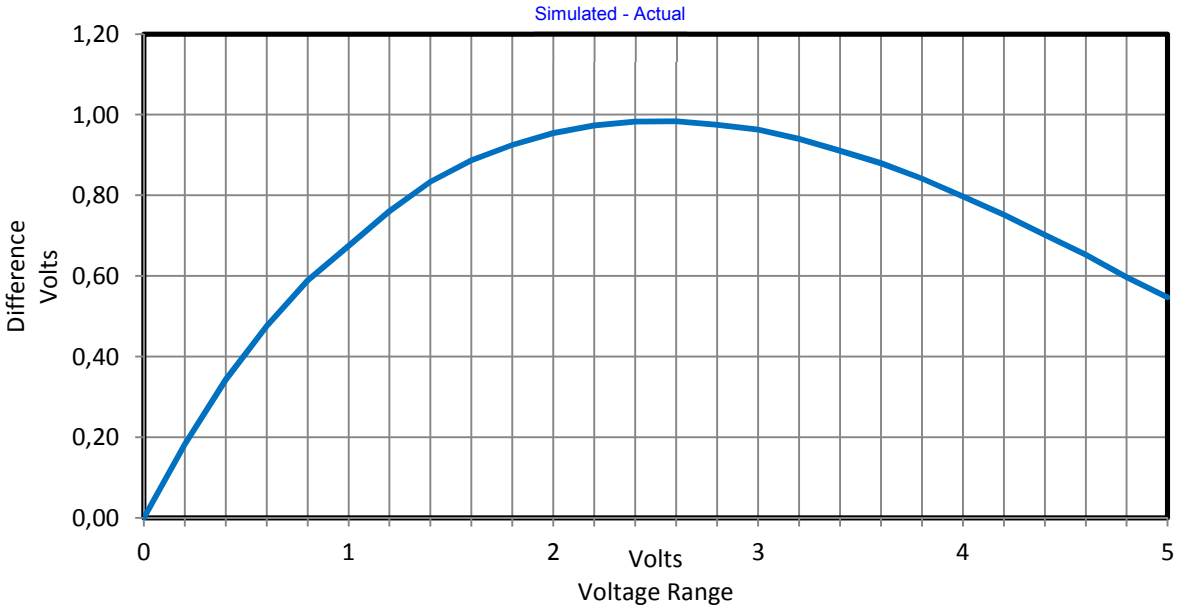

Figure 17: Measured voltage subtracted from actual voltage of the single optocoupler.

To increase accuracy of the circuit, the output was amplified. This was done by increasing the resistance of R2. The simulated output can be seen in figure 18.

Simulated and actual output voltage for a 0V to 5V DC input.

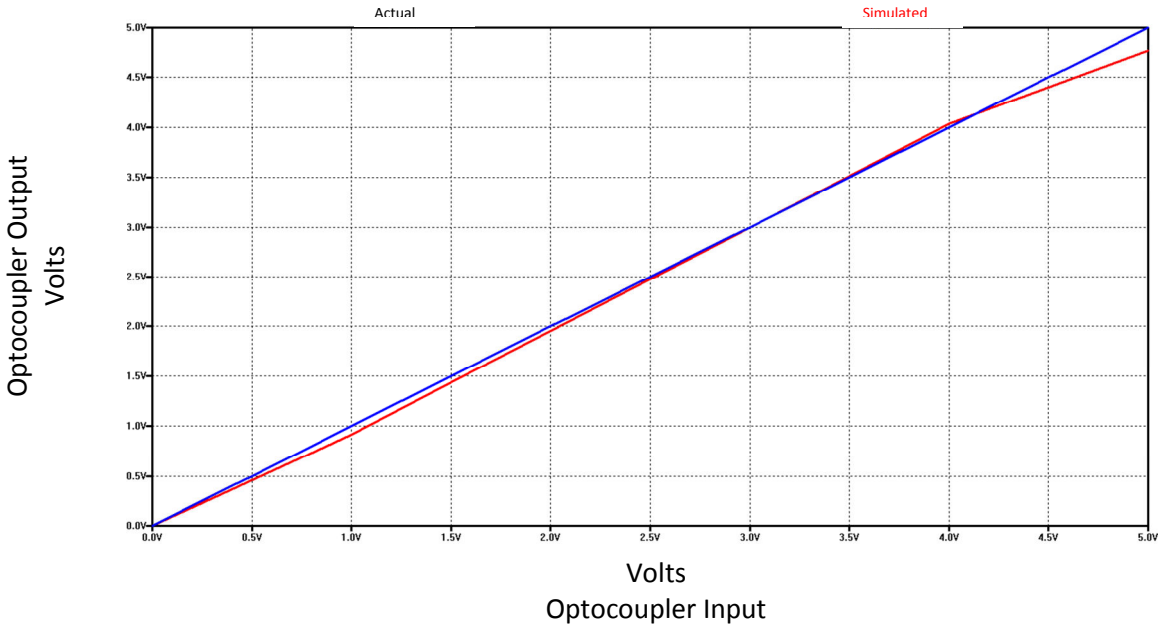

Figure 18: Simulated results of the single optocoupler.

The accuracy is lower than system specification which states that measurement accuracy has to be smaller than 10mV. The largest differences of actual voltages compared to the simulated results were about 230mV. This can be seen in figure 19.

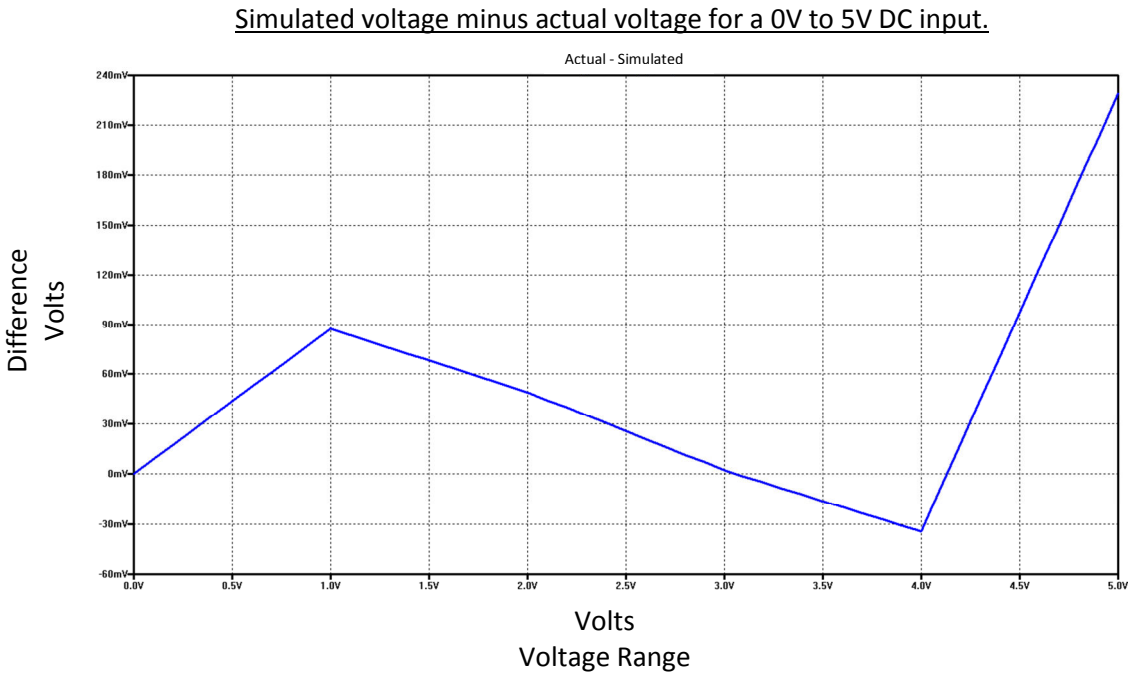

Figure 19: Simulated voltage subtracted from actual voltage of the single optocoupler.

Maximum deviation between the two curves can be seen at an input of 1V, 4V and 5V. The highest linearity was roughly between 2.75V and 3.25V. This is a narrow range and do not fall within any cell's operating voltage. The circuit was not modified and shown as any results obtained similar to that of the simulation would not allow accurate cell voltage measurement.

3.2.5 Summary

The optocoupler is the most economical of all the methods discussed in this thesis. Problems with the topology include high current consumption and complex integration.

Simulation results showed promise, but measured results were different from simulation results. Because of the low accuracy this circuit cannot be used to measure the voltage of a cell. Another optocoupler topology, the dual phototransistor optocoupler method will be investigated next.

3.3 Dual Optocoupler

3.3.1 Theory

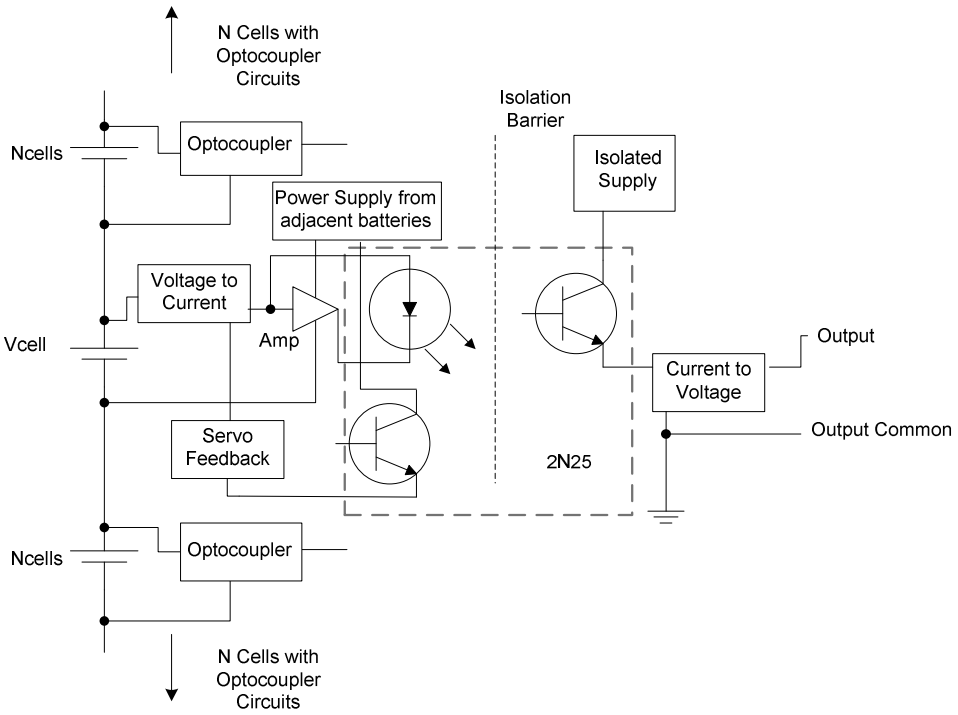

Figure 20: Optocoupler with dual integrated phototransistors.

The second topology available is an optocoupler with two phototransistors and one photodiode in photoconductive mode (Del Rio & Cao y Paz, 2002).

Using the input photodiode in a servo feedback configuration, linearity is greatly improved because the circuits non-linear dependence on the LED's forward current is removed i.e. what affects the one photodiode affects the other photodiode (with regard to temperature and frequency) while both operate on flux from the same LED (Alessandrello et al., 1998) (Camina et al., 2004).

Circuit accuracy can be further increased by using the optocoupler in photovoltaic mode. In this mode no external voltage is supplied to the primary phototransistor and it operates only on the radiated light it receives. This provides high accuracy because the output is completely independent form the LEDs input current as well as supply voltage and current to the phototransistors.

Phototransistors generate a current when they are radiated by the LED flux. Since the gains of the two phototransistors are the same and they each receive the same amount of flux, the input is equal to the output. Theoretical accuracy of up to 14 bits can be obtained using this method (Camina et al., 2004).

3.3.2 Circuit Design

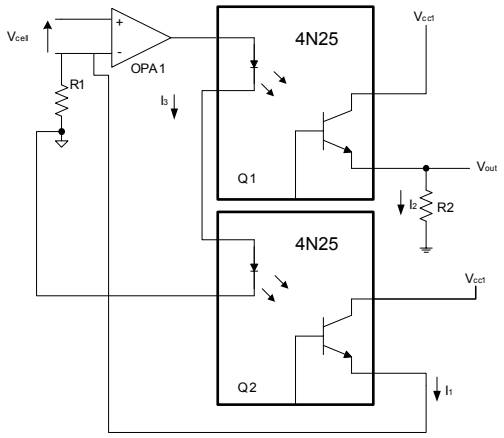

Figure 21: Dual optocoupler design

Instead of using a single optocoupler with one emitting LED and two phototransistors, two optocouplers were used. The design can be seen in figure 21 and the simulation in figure 22. The design is based on the previous single optocoupler circuit, but with modifications for the second optocoupler.

When a voltage is applied to the input of OPA1 the output goes positive and current flows through the two diodes. Q1 and Q2 receive the same amount of flux and because they are connected to the same supply voltage and have the same resistance values (R1 and R2) they generate the same voltage over the resistors.

The output of the opamp remains high until the voltage over R1 is equal to the input and thus a closed loop is created with the input voltage being transferred to the output with galvanic isolation incorporated.

3.3.3 Simulation

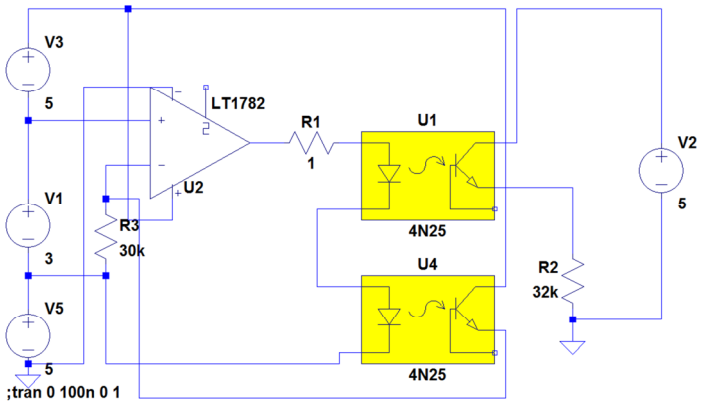

Figure 22: Dual optocoupler circuit used to increase accuracy over the single optocoupler circuit.

Two identical optocouplers are used in a servo feedback configuration to try to linearise the CTR of the circuit.

3.3.4 Results

Simulated and actual output voltage for a 0V to 5V DC input.

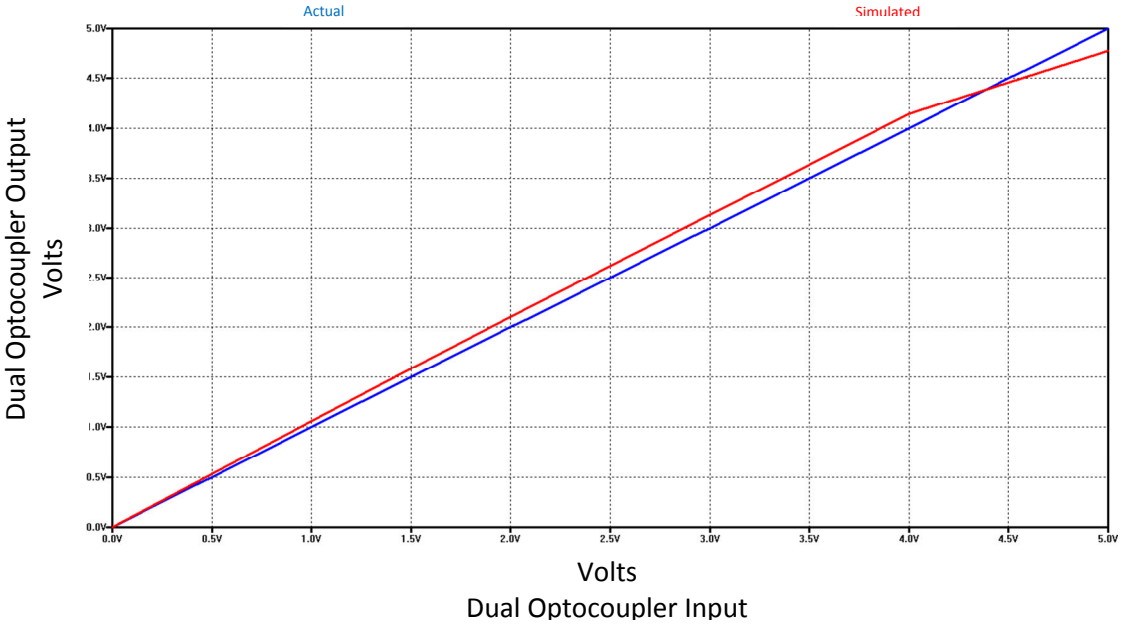

Figure 23: Simulated results of the dual optocoupler

Simulated voltage minus actual voltage for a 0V to 5V DC input.

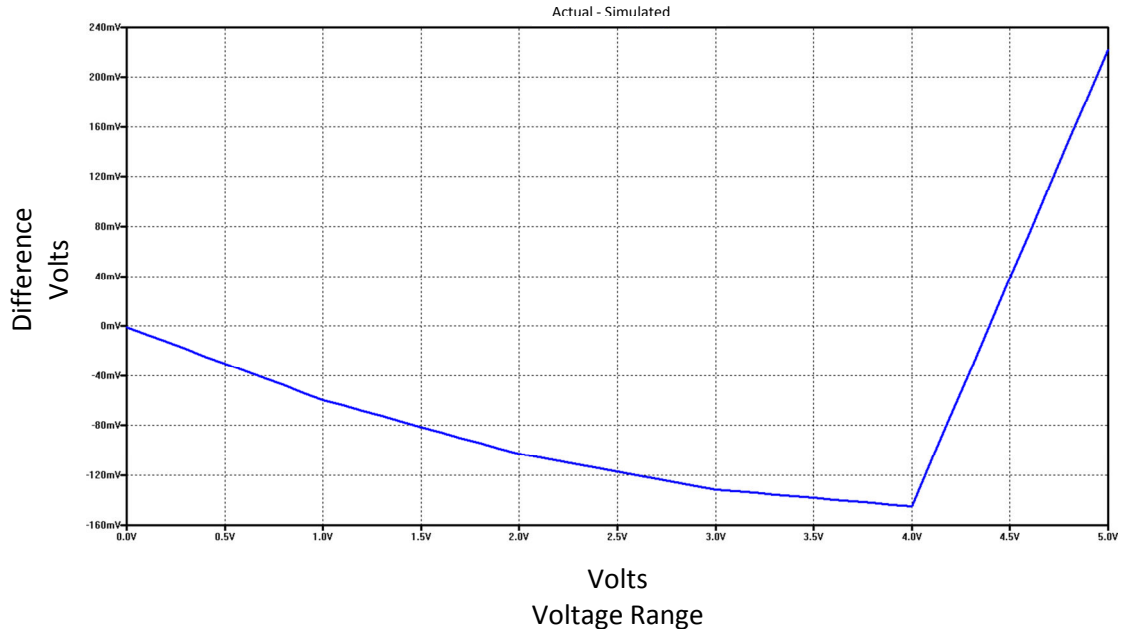

Figure 24: Simulated voltage subtracted from actual voltage of the dual optocoupler.

Figure 23 shows that the circuit performs similar in the simulation as the single optocoupler circuit. Contrary to what was originally suspected. The use of the double optocoupler was assumed to negate the differences of the CTR and linearise the optocoupler.

Since two separate optocoupler were used in simulation, instead of one, they were both set to have identical properties, thus the error cannot be attributed to the abovementioned factor.

As mentioned above, figure 23 has the same error figures as the single optocoupler circuit. The difference is that the error starts at a minimum and rises to a maximum as can be seen in figure 24 where after it turns around and moves to a maximum in the other direction.

Maximum deviation between the two curves can be seen at an input of 5V. The highest linearity can be seen at 0V to 0.5V. This, like the previous circuit is a very narrow range and no cells will be accurately measured.

Measured results are similar to the simulated results and can be seen in figure 25. Even though the results obtained are better than the single optocoupler circuit they are not accurate enough for accurate voltage measurements. The results do not comply with the system specification and this circuit is unusable, just like the previous circuit.

Measured and actual output voltage for a 0V to 5V DC input.

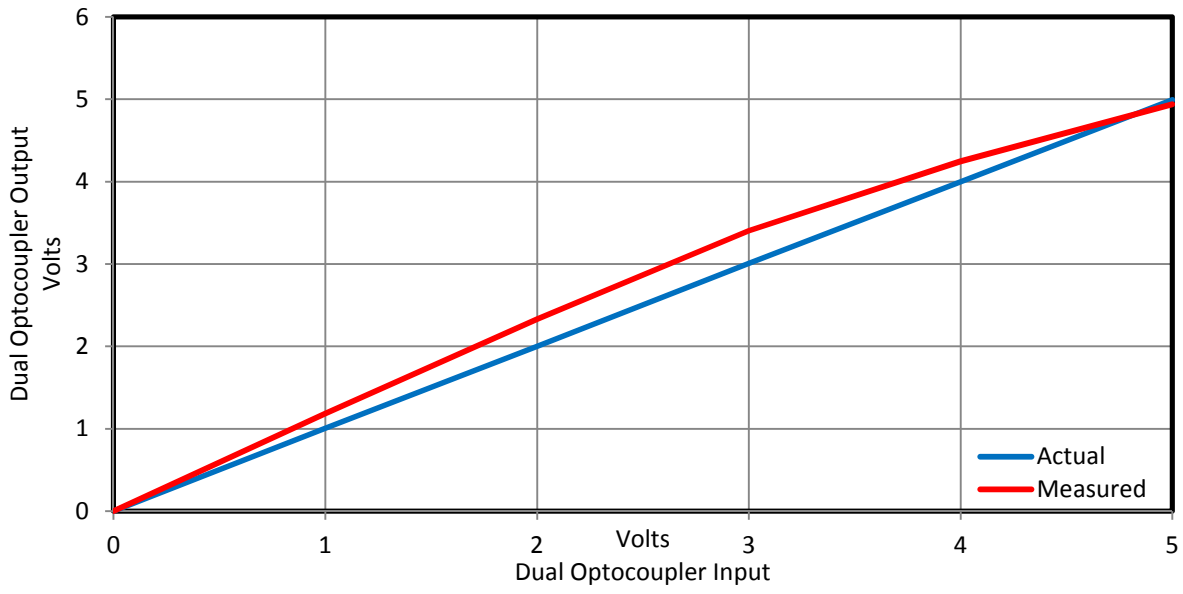

Figure 25: Measured and actual results of the dual optocoupler.

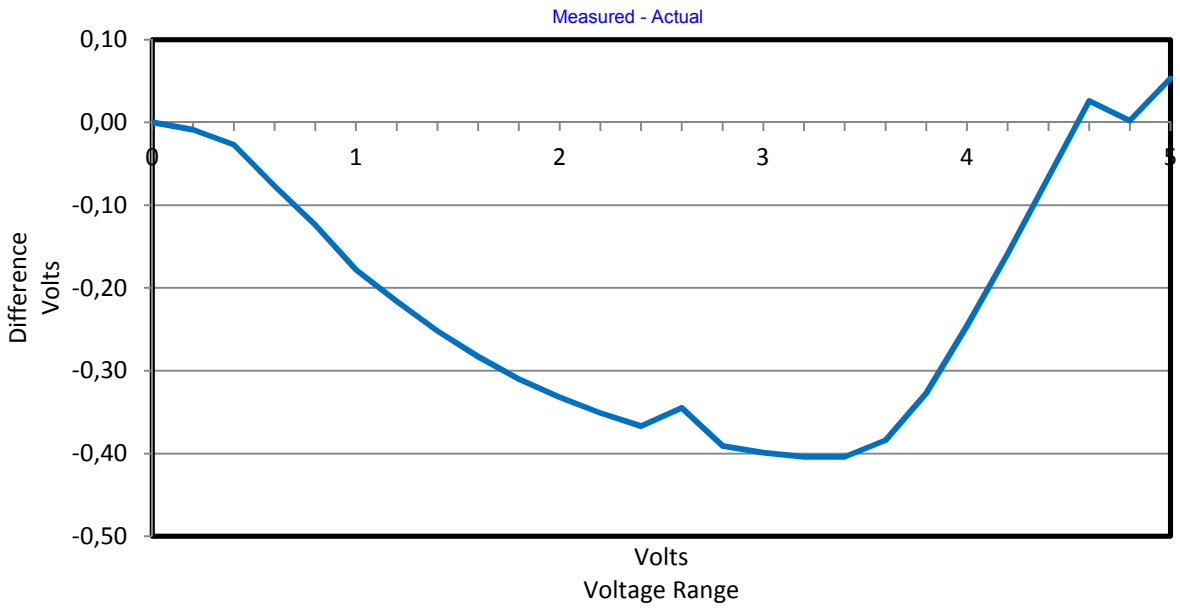

Figure 26: Measured voltage subtracted from actual voltage of the dual optocoupler.

3.3.5 Summary

It was suspected that the dual optocoupler would perform better than the single optocoupler. Although this was the case in the actual tests, the single and dual optocoupler circuits are both unusable as accurate voltage measurement devices.

3.4 Isolation Transformer

3.4.1 Theory

An alternative to isolation amplifiers that are produced by manufacturers is to build them. Using different techniques, the voltage information can be transferred over an isolation barrier, while still being accurately measured as was the case for the isolation amplifiers.

Figure 27 shows a schematic of a circuit that uses a signal transformer and a current pulse generator to isolate the input from the output. A study by Linear Technology covers an actual system design, which can be seen to be neither simple nor small (Williams & Thoren, 2008).

The circuit in figure 27 will, when the primary is pulsed with a short current pulse, produce a voltage related to the secondary voltage on the primary winding. The secondary voltage that appears on the primary winding is the cell voltage plus the forward diode drop of the transistor along with an added error from the transformer.

The voltage on the primary winding is left to settle for a short period and then sampled where after gain correction can be done electronically or with a software algorithm.

3.4.2 Circuit Design

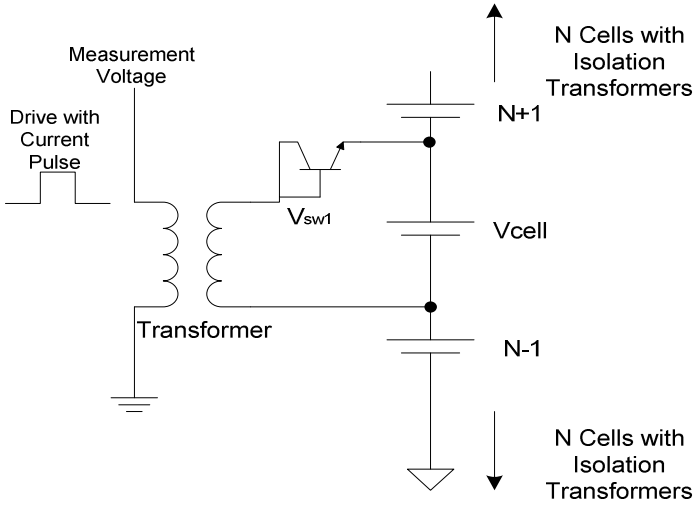

Figure 27: Transformer used to isolate the cell from ground while still allowing the cell to be measured (Williams & Thoren, 2008).

The circuit design with timing is complex and only the transformer part of their design is shown. This allows the circuit to be simulated. The diode was replaced by a transistor as they are more stable over temperature and can be matched easier than diodes (Williams & Thoren, 2008).

3.4.3 Simulation

The simulation of this circuit was done with PSim 9 from Powersim Inc. Results from the simulation were exported to a text document and Excel was used to draw graphs for the simulation.

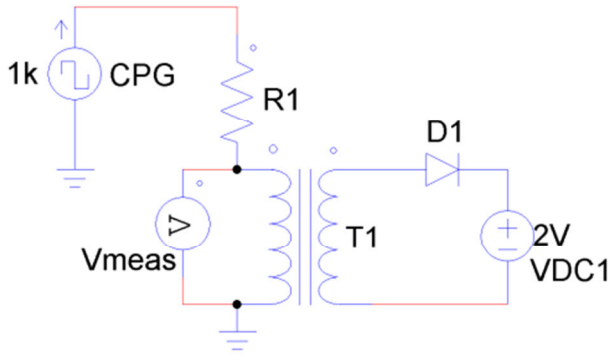

Figure 28: Isolation transformer circuit being supplied by a 1 kHz current pulse to measure a 2V DC source.

Figure 28 shows all the necessary components required to operate this circuit. A diode, D1, was used instead of a transistor because of the adjustable ideal conditions it possesses as well as the ease of its use in the simulation package.

3.4.4 Results

Simulated and actual output voltage for a 0V to 5V DC input.

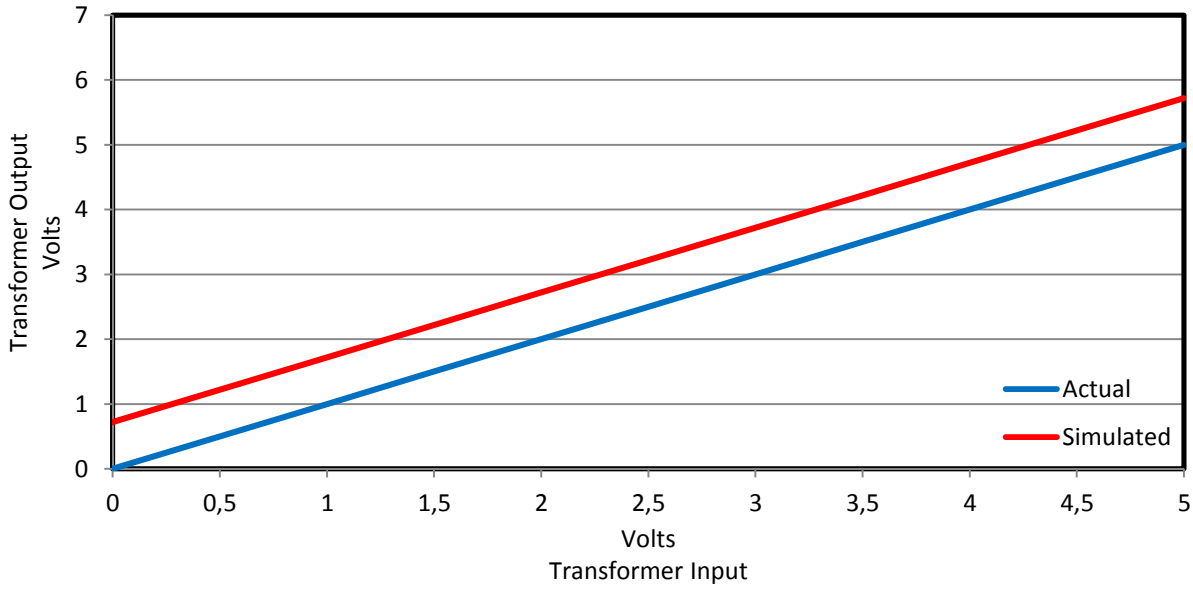

Figure 29: Waveform showing the transformers primary voltage plus the diode voltage drop and the error added by the transformer over a 0V to 5V input.

In figure 29 it can be seen that the secondary voltage appears on the primary of the transformer, along with some error, when pulsed. The diode and the transformer add a constant error to the

circuit that can be compensated for later. This error as well as the value of the error can be seen in figure 30.

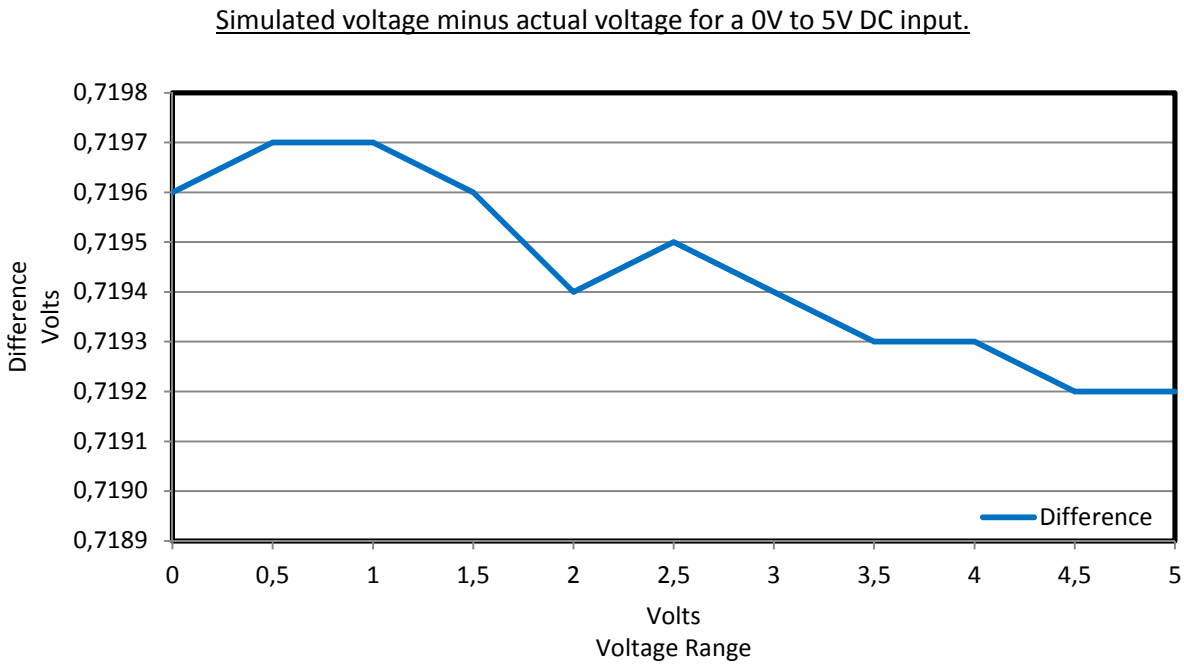

Figure 30: Simulated voltage subtracted from actual voltage of the isolation transformer.

3.4.5 Summary

Accurate results can be obtained using this topology. A small transformer size can be used as no power is transferred through the transformer. The topology suffers when multiple of these single sections are put together, as the system becomes complex, slow and less accurate with temperature change.

Extra circuitry is required to negate the effects of the transformers negative recovery excursion after being pulsed. This topology, like the isolation amplifier, would only work satisfactory when small cell stacks need to be measured.

3.5 DC to DC Converter

3.5.1 Theory

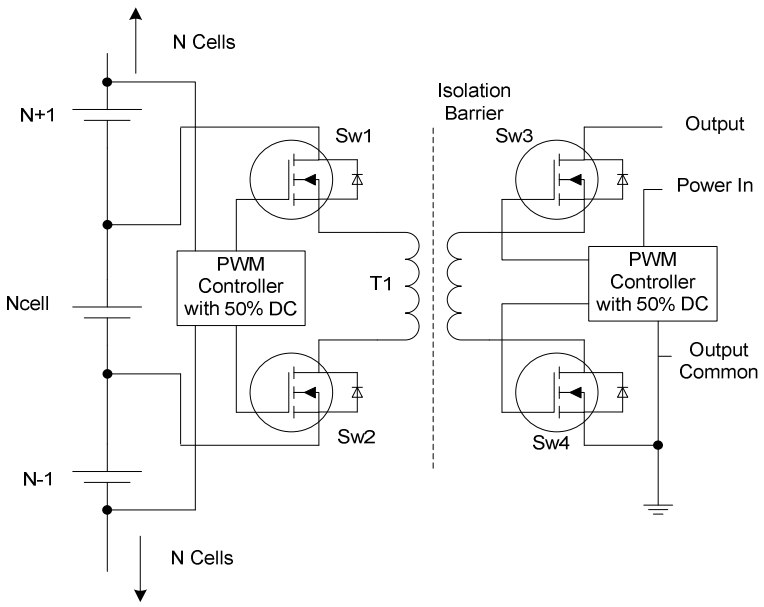

Figure 31: DC-to-DC converter which can be used to measure individual cells (Wagener, 2009).

A DC-to-DC converter (figure 31) can also be used to measure cell voltages while providing galvanic isolation. A prototype system for this topology was available. It was used to test if high currents could be transferred from one cell to another when a small potential difference between cells exists as part of active cell balancing tests (Wagener, 2009).

The circuit consists of a full bridge with transformer isolation. The transformer was chosen to be a planar transformer to decrease overall footprint while providing low leakage inductance and a high efficiency.

With a 1:1 transformer and 50% duty cycle for both sides of the PWM controllers the voltage on the secondary side was directly transferred to the primary side.

The circuit was originally designed for power conversion and thus have higher rated components than are required for a voltage monitoring system. A voltage monitoring systems power requirements are kept to a minimum and should draw minimum current. It is therefore not necessary to use high powered devices or a big transformer.

3.5.2 Circuit Design

This circuit was designed and built by Johan Wagener while doing testing on active cell balancing. The circuit operation allowed for bidirectional current transfer over an isolation barrier.

The circuit was unsuccessful as an active cell balancer due to high circuit resistances, but as the current requirement for modern ADCs are minimal, a highly accurate measurement device was realised.

The circuit was simulated and the prototype device was used for voltage measurement tests. For detailed circuit design and description refer to “Investigation and Design of a magnetically isolated DC-DC converter for use in cell balancing" (Wagener, 2009).

3.5.3 Simulation

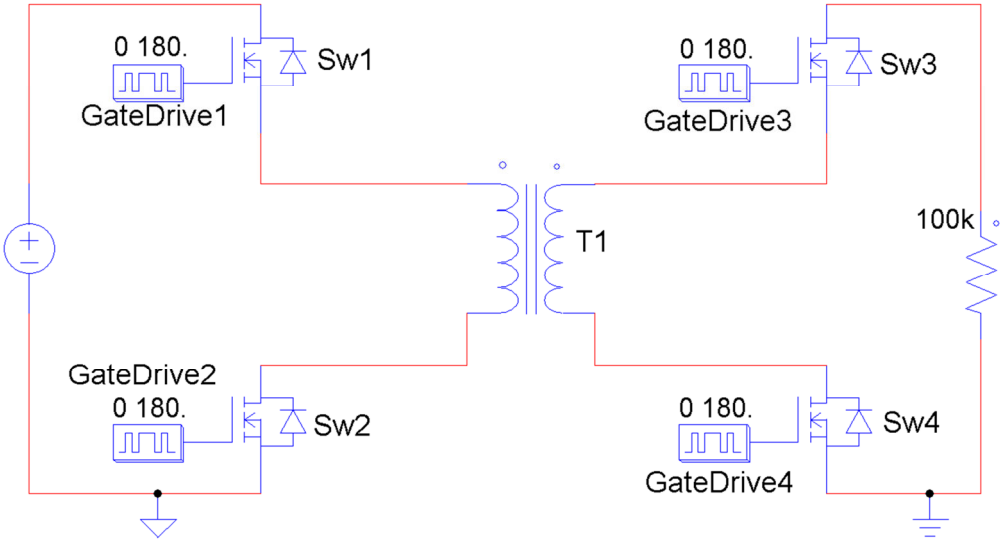

Figure 32: Circuit used for simulating the DC-to-DC converter (Wagener, 2009).

The circuit in figure 32 was used for simulation of the DC-to-DC converter. It was built in PSim and as no DC sweep function exists, a low frequency (1 Hz) triangular voltage was used to simulate this (the triangular wave generator is left out of the figure).

Transformer specifications were set to the final design prototype specifications as calculated and tabulated in the thesis (Wagener, 2009).

3.5.4 Results

Simulated and actual output voltage for a 0V to 5V DC input.

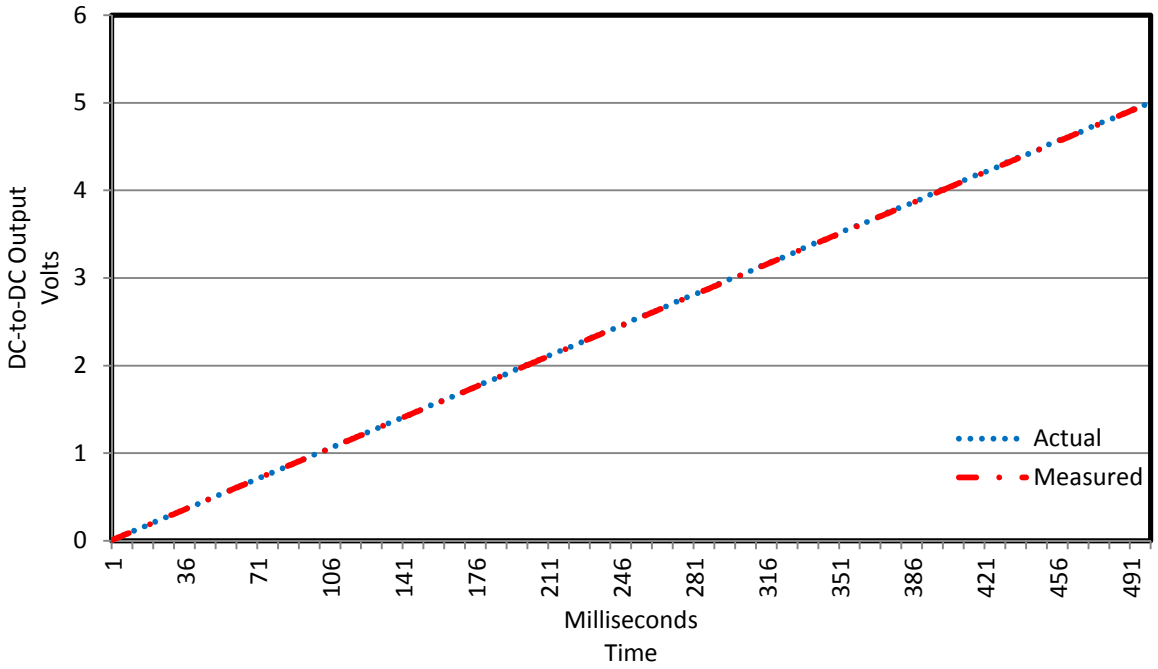

Figure 33: Simulated results of DC-to-DC converter.

The output voltage (figure 33) follows the input voltage so precisely that only a straight line can be seen.

Simulated voltage minus actual voltage for a 0V to 5V DC input.

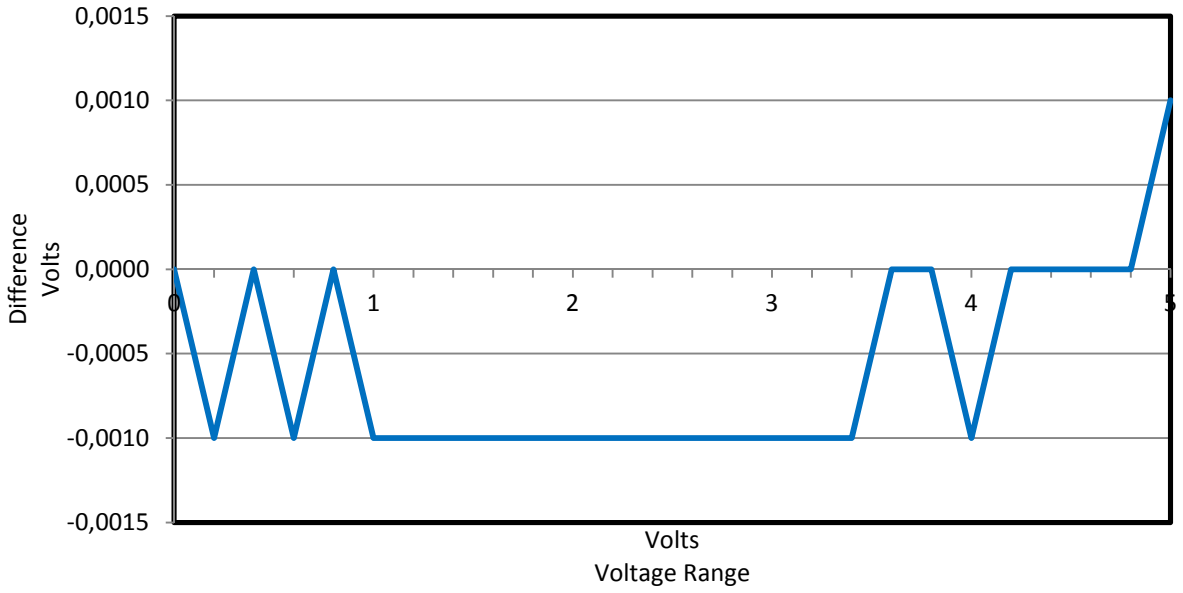

Figure 34: Simulated voltage subtracted from actual voltage of DC-to-DC converter.

Figure 33 shows that the measurement accuracy of the measurement device is high as can be seen from the difference measurement in figure 34.

Measured and actual output voltage for a 0V to 5V DC input.

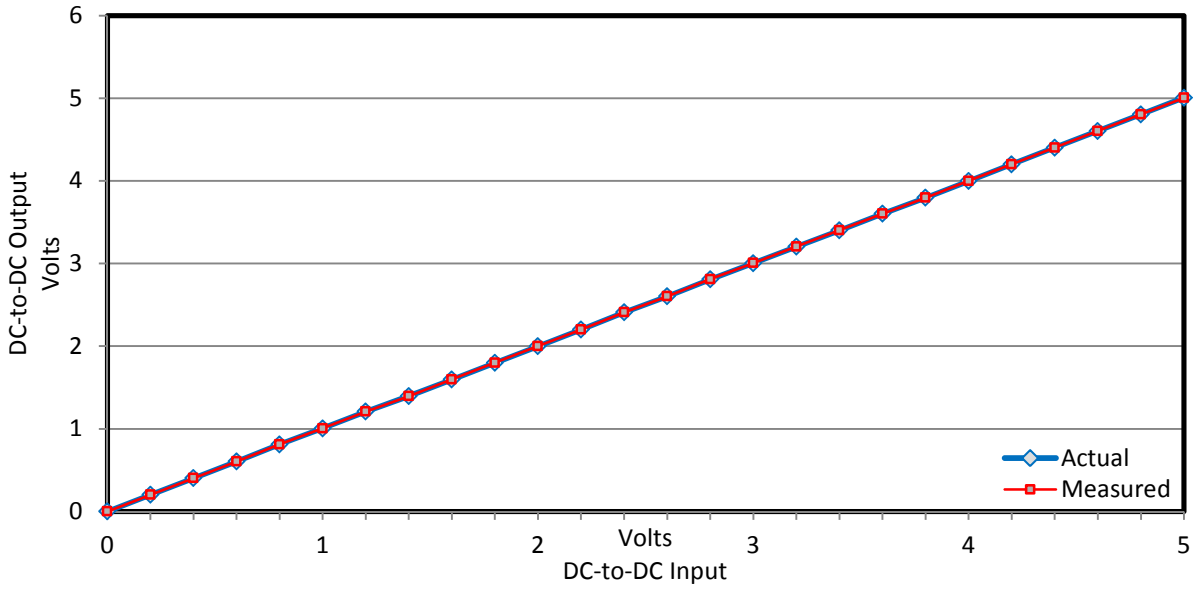

Figure 35: Measured and actual results of the DC-to-DC converter circuit.

Measured voltage minus actual voltage for a 0V to 5V DC input.

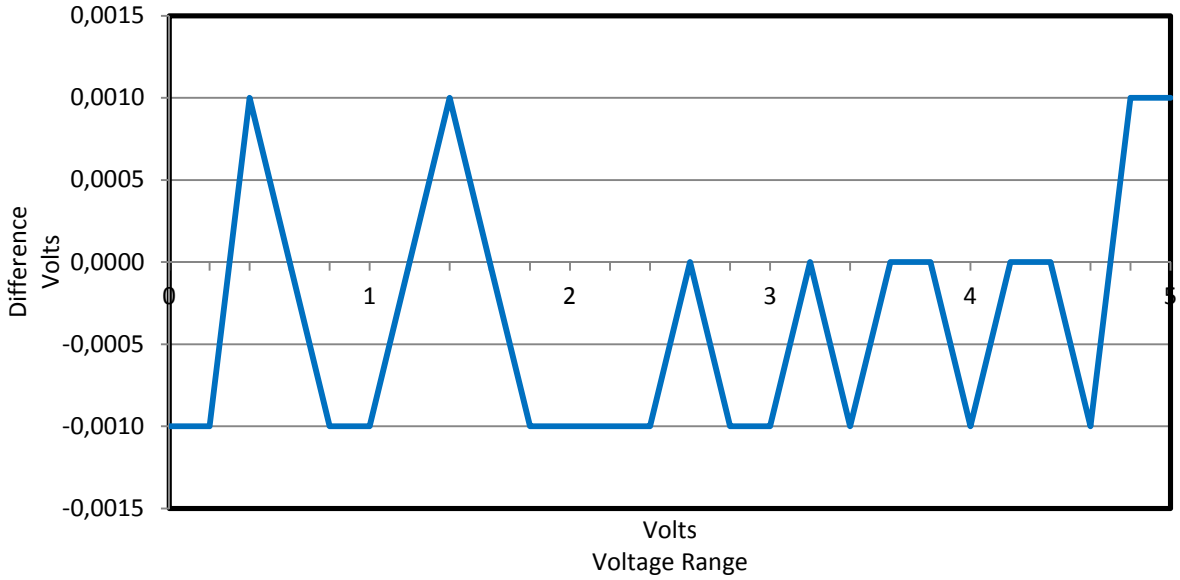

Figure 36: Measured voltage subtracted from actual voltage of DC-to-DC converter.

Figure 35 shows the measured values follow the actual measurements precisely and the difference measurements in figure 36 prove this. Values were rounded to the nearest millivolt.

This allows this circuit to be used in an accurate voltage measurement system as the error over the full measurement range is an order of magnitude smaller than the system specification.

3.5.5 Summary

This topology delivered excellent results and could be used as an accurate cell measurement device with very little drift over a large temperature range. The prototype was great in size, but could be scaled down using low power components. If the previous multiplexer system example is followed the topology still suffers from being complex to connect in a large system.

3.6 Conclusion

Of the four topologies covered thus far, three would have been suitable for accurate cell measuring systems. They are the isolation amplifier, transformer isolation and the DC-to-DC converter. All of them had the necessary accuracy for cell measurement, but system integration was complex.

Simply put, these are the traditional methods of measuring voltages while incorporating galvanic isolation. IC manufacturers realised this and saw that with the increasing use of cell stack application that a new method was needed to measure individual cells in large cell stacks in a small easily integrated package. They are battery management ICs and analysis on them follow in the next chapter.

4 Chapter Four: Stackable Battery Management ICs

4.1 Introduction

Battery management ICs are relatively new to the portfolio of IC manufacturers. A need for a simpler, cheaper and smaller way of battery management was needed. An IC allows all the necessary measurement hardware to be incorporated into a single small package.

The search criteria for the ICs that were considered had to include the following

1. Could measure at least 6 individual cell voltages at the same time.
2. Had either a SPI or I^2C communications interface.
3. Had some form of passive cell equalising.
4. Readily available from the manufacturers.

Three ICs were chosen for investigation. They are the MAX11068 by Maxim, the ATA6780 by Atmel and the LTC6802-1 by Linear Technology Corporation.

4.2 Maxim (MAX11068)

The first IC considered for use is the MAX11068. This IC is stackable to allow cell measurements of high series cell stacks. Cell data is acquired through a 12 bit successive approximation ADC which also allows internal and external temperature measurements to be taken. These features already allow 12 series connected cells as well as three temperature measurements to be measured in a package that is 9.7mm x 4.4mm. This is significantly smaller than any of the other topologies previously discussed.

A short discussion of the IC follows

Figure 37 shows the internal components of the MAX11068. The switch bank consists of a high voltage multiplexer that switch between the cells. As they are switched they are measured by the 12bit ADC and stored in internal memory in the package. Total measurement time is stated as 80us for 12 cells, with an increase of 1us for every 12 cell stack after that. A 12 bit successive approximation (SAR) ADC allows the quick measurement time of all cells in the stack.

All cells are measured with reference to a high precision voltage reference of 2.5V for high accuracy.

If, while charging, it is seen that any cell voltage is higher than the rest an internal switch can be used to shunt a resistance over the cell and this allows the cell to charge slower than the other cells. If

power dissipation is too high inside the IC package an external, high power MOSFET and resistor can be used as an alternative switch.

The whole system is controlled by a 6MHz internal oscillator clock which sets the communication speed and internal operation of the ADC and multiplexer. A watchdog timer (WDT) is included to allow for fault detection. No preventative action is taken by the device itself should fault occur and all functional control rely solely on the controller controlling the devices.

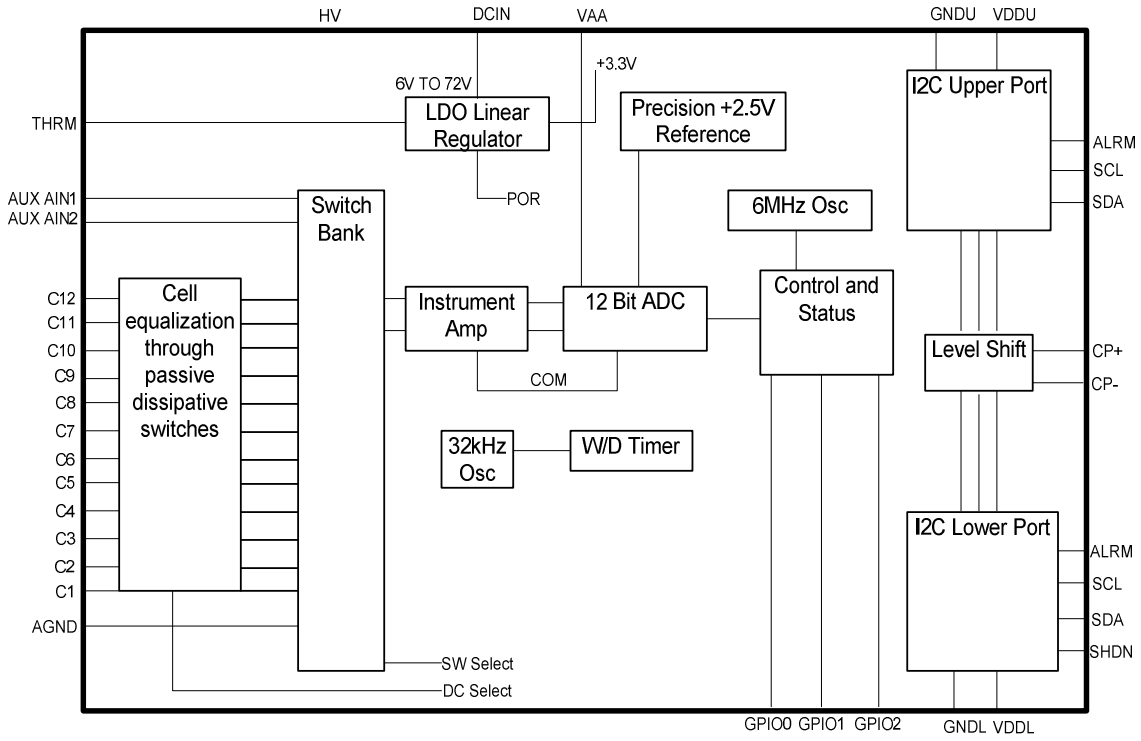

Figure 37: Internal block diagram of the MAX11068 IC by Maxim (Maxim, 2008).

Communication to and from this IC is via the two wire protocol called I^2C and a maximum frequency rate of 200 kHz is possible through the ICs communication ports. Isolation to higher stacked devices is through the isolation ports and no external isolation or components are required.

4.3 Atmel (ATA6870)

The latest battery management IC mentioned here is the ATA6870. This IC allows measurement of 6 series stacked cells to be measured with one IC and is series stackable up to a maximum of 16 ICs i.e. 96 stacked cells maximum.

A short discussion follows.

Operation of this IC is different to the previous IC in that each cell has its own 12 bit ADC which is level shifted after it has been measured. These measurement readings are stored in internal memory registers and can be accessed by a controller. Data acquisition time is 9.5 milliseconds, and 11.5 milliseconds for four units that would allow twenty four series connected cells to be measured.

Cell balancing can be done in exactly the same way as for the previous IC, either internally or externally through semiconductor switches and resistors.

Internal and external temperature measurements are possible through the same type of 12 bit ADC, although a separate ADC is used for the cell measurements.

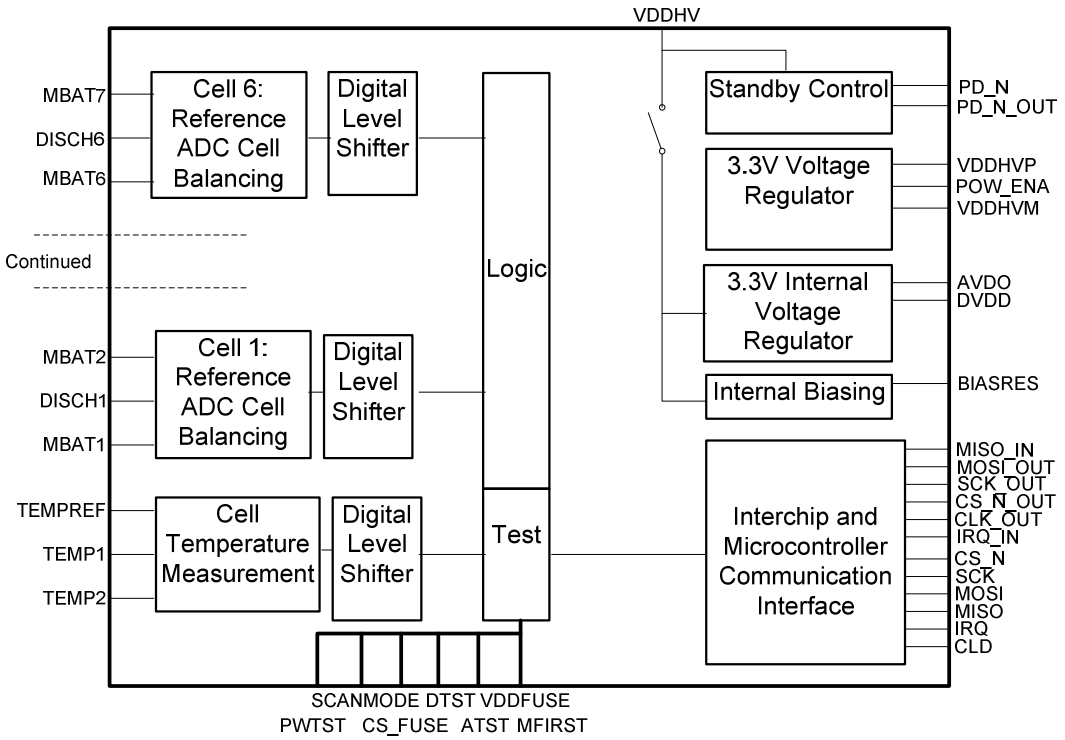

Figure 38: Internal block diagram of the ATA6870 IC by Atmel (Atmel, 2009).

The communication protocol is based on the SPI protocol and can operate at a maximum frequency of 250 kHz. Isolation to an IC allocated higher in the stack is provided internally, through diodes. Just like the MAX no external components are needed.

The main advantage of this IC is that it supplies power to the controller from the top of the stack. This voltage is level shifted and regulated internally along the ICs to the bottom IC where it is referenced to ground. The advantage of doing this is that no further imbalance is created by drawing power from the lower cells in the stack (Atmel, 2009).

4.4 Linear Technology Corporation (LTC6802-1)

The LTC was the first IC that was obtained and was considered most acceptable. Appendix A contains a comparison summary of the LTC, MAX and ATA ICs as done by eLithion (Andrea, 2009). Discussion of the LTC and how it compares to the other battery management IC follow in the next section.

4.4.1 Delta Sigma ADC

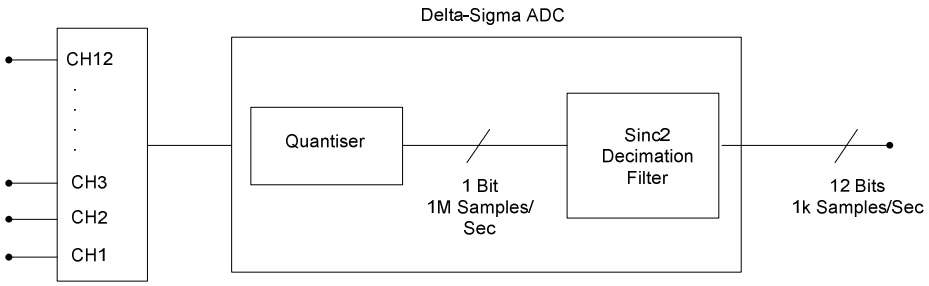

Figure 39: Delta sigma ADC sampling at 1M samples per second with a throughput of 1k samples per second (Kultgen, 2009).

The LTC6802-1 allows twelve cells to be measured using a 12 bit delta sigma ADC. The MAX11068 uses a SAR ADC and the ATA6870s ADC was unspecified. Using a delta sigma ADC, the input is sampled a number of times and then the output is filtered to produce an average binary value. SAR ADCs sample the input once and quantise an output code dependant on that snapshot.

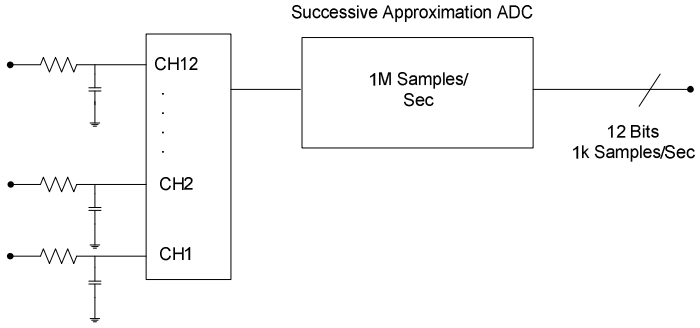

Figure 40: SAR ADC sampling at 1M samples per second with an effective throughput of 1k samples per second (Kultgen, 2009).

Figure 39 and 40 show the two different ADCs. Both ADCs sample at 1M samples/sec but the delta sigma output is followed by a $sinc^2$ decimation filter and the SAR ADC is preceded by a single pole RC filter. The $sinc^2$ filter uses an accumulator and a differentiator and is applied twice after one another. The accumulator is the averaging component and allows the system to average. This means that the SAR ADC is significantly faster than the delta sigma which shows why the MAXs sampling time of 80us is much quicker than the LTCs sampling time of 13ms. (Kultgen, 2009).

The advantage of delta sigma is that the results of the sinc^2 filter will have lower quantisation and interference noise.

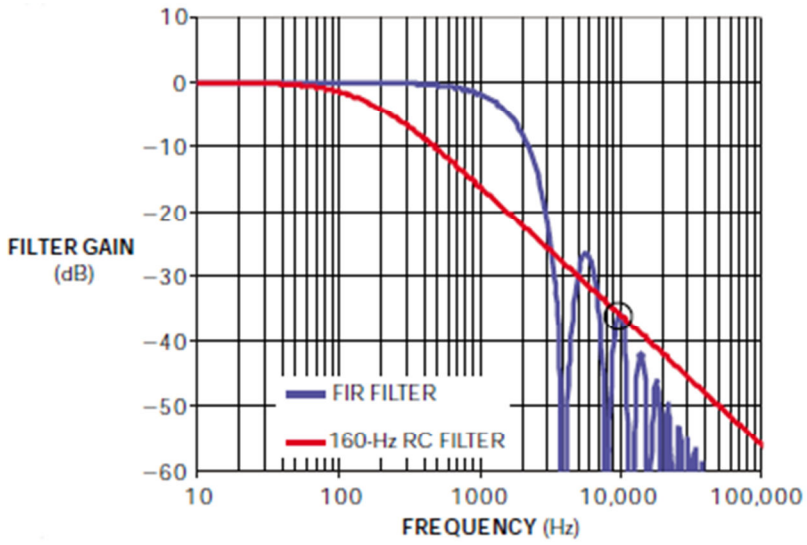

Figure 41: A single pole RC and a finite input response (FIR) filters attenuation shown and marked at 10 kHz (Kultgen, 2009).

Figure 41 shows the filtering of the sinc^2 filter compared to that of the single pole RC filter.

4.4.2 High Voltage Multiplexer

The high voltage multiplexer, which can safely operate at voltages up to 60V, uses a switched capacitor sampling design. This eliminates the CMRR restrictions commonly facing discrete designs. A single, high accuracy delta sigma ADC, discussed previously, measures all the voltages that are switched to it via the multiplexer. This ensures high accuracy as only one ADC needs to be calibrated and all measurements thereafter are done using the same ADC and precision voltage reference.

4.4.3 Discharge Switches

Passive discharging is included by utilising internal semiconductor switches. The switches have a guaranteed minimum resistance of 20 ohms. This allows the switch to be used directly as the discharge resistance over the battery, although care must be taken not to overheat the package through excessive power dissipation in the IC.

The chip datasheet does not specify the maximum power dissipation of the IC, but with temperature shutdown at 150°C and internal monitoring, this can be avoided and necessary precautions taken in time. The switches can also be used to switch on a higher power semiconductor switch and resistor if needed.

4.4.4 Communication Protocol

Communication to and from the IC is done via the four wire SPI protocol. The data width is eight bits and can achieve a maximum operating frequency of 1MHz. Several settings such as the configuration settings, ADC start signal, discharge switch setting, 10 or 12 cells per chip and over and under voltage settings are set via the SPI ports.

When the ADC conversions are done all the data is stored inside registers in the IC and can be accessed with an external controller that has SPI capability.

4.4.5 Watchdog Timer

The watchdog timer is used to ensure that an error never occurs by resetting the IC when normal operation does not reset the watchdog counter. When the watchdog is enabled and SPI communication activity ceases for more than two seconds the watchdog will reset all the registers to their default settings. This ensures that the chip will go into standby mode.

4.4.6 Internal Temperature Measurement

The IC cannot operate at temperatures above 150°C. If this temperature is reached it will reset the ICs registers to their default values. This will put the IC into low power mode, disable the discharge switches and stop operation. Operation will only be allowed to continue if the temperature has reached 145°C.

The internal temperature measurements are done with the same delta sigma ADC as is used for the cell measurements. This allows for full control of the discharge switches by allowing the disabling of switches if the temperature reaches the ICs limit.

Full measurement accuracy is guaranteed until 80°C and if operation above this is required, the temperature readings can be used for temperature compensation. There are also two extra inputs that can be used for temperature sensing on or around the battery stack to measure temperature specific areas (Linear Technology Corporation, 2009) (Wright, 2006).

By allowing the measurements of potential hotspots on the battery stack allows for the necessary action to be taken to prevent excessive heating, thus the reliability as well as the lifetime of the stack is increased (Chen & Evans, 1995).

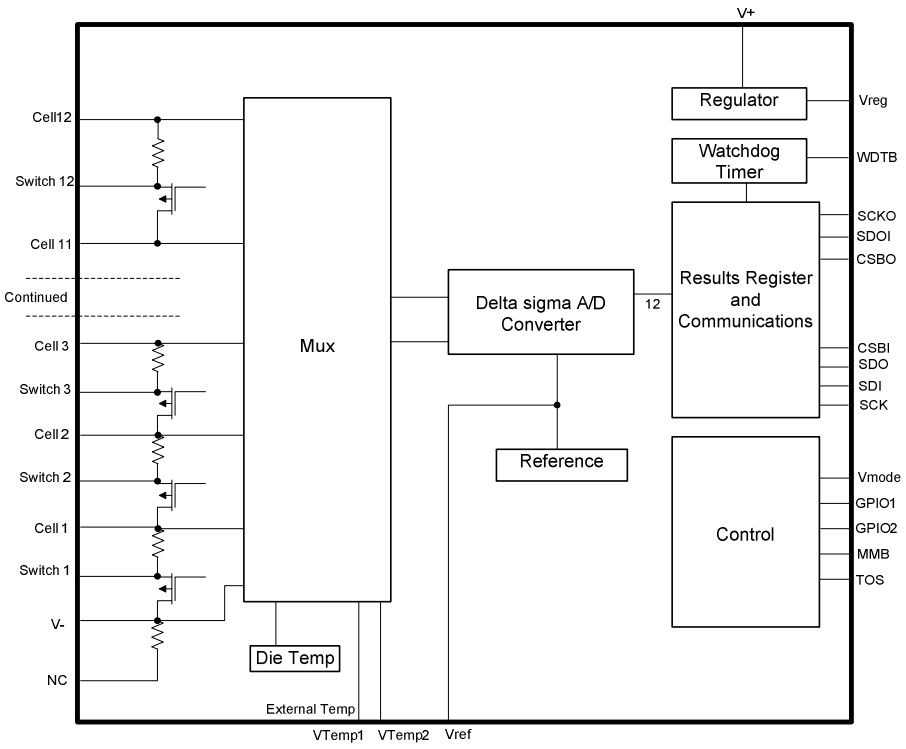

Figure 42: Internal block diagram of the LTC6802 IC by Linear Technology Corporation (Linear Technology Corporation, 2009).

4.5 Conclusion

These ICs allow all the functionality that is desired from a multiple cell measurement system to be incorporated into a small package for high accuracy, small footprint, low current consumption, ease of integration and speed when measuring individual cell voltages of high count cell stacks.

5 Chapter Five: System Considerations

When choosing a controller to control the battery management IC the following needs to be taken into account:

1. The required operating speed of controller.
2. Peripherals required from the controller.
3. Necessary housekeeping requirements.
4. Cost of the controller.

A microcontroller will be capable of handling the necessary communication to and from the battery management IC at maximum speed and perform the necessary low maintenance housekeeping. Microcontrollers are cost effective when compared to more powerful controllers like field programmable gate arrays (FPGAs) and complex programmable logic devices (CPLDs). FPGAs and CPLDs are customisable to user requirements with high clock rates, large memory capacity and they are compact, but require more power than a microcontroller to operate. Considering the controller requirements an 8 bit microcontroller was chosen for this purpose.

A microcontroller is a programmable computer on a chip that includes a controller (CPU), memory and programmable inputs and outputs. Having a programmable chip also means that a programming device must be used to program the control code onto the microcontroller.

The first section of this chapter discusses the different serial communication protocols used in the proposed system as well as two others for comparison purposes, various microcontrollers and their peripherals, programming languages and a programming device to program the selected microcontroller.

The second section will answer the following questions:

1. Should discharge switches be enabled and should external switches be utilised?
2. How many battery management ICs are required?

After the selection of the controller and the design considerations of the LTC the design and development of a development board for the chosen microcontroller will be investigated. This is done to ensure proper operation of the microcontroller and programming before the prototype board is designed.

The fourth section deals with the data that is collected using the LTC6802-1, how will this data be handled and stored and how this data will be made available to the user. A graphical user interface is defined and fields that need to be displayed are discussed.

5.1 Microcontrollers

There are numerous microcontroller manufacturers and their products and ranges are ever expanding. Microcontrollers differ from microprocessors in that they have built in memory and peripherals; hence they are “computers on a chip”. The major differences between the different microcontrollers are the architectures and instruction sets with which they operate.

68HC11 (Freescale Semiconductor), AVR (Atmel), H8/500 (Hitachi), Z80 (Zilog), 80196 (Intel), MSP430 (Texas Instruments), PIC16F (Microchip), LPC2000 (Philips), are just a few of the ranges produced by different manufacturers.

Choosing the right microcontroller for the application is very important and while all microcontrollers with the same peripherals can perform the same functions, one might be more suited to a specific application.

The following microcontrollers will be compared to see which one best suite the application:

1. Microchip (PIC16F877A)
2. Atmel (ATmega16)
3. Texas Instruments(MSP430)

When choosing a microcontroller, the following should be considered:

1. I/O pins available.
2. Supported hardware peripherals.
3. Speed in million instructions per second (MIPS).
4. Memory, both flash and electrically erasable programmable read only memory (EEPROM).

5.1.1 PIC 16F877A

The PIC microcontroller has been around for a long time and has a lot of development software and online support.

The internal architecture of the micro is based on the Harvard architecture. This allows for the program memory and data memory to be separated and accessed from different busses. The 16F series has an instruction set of 35 instructions and requires four oscillator cycles to perform one machine cycle. This allows the microcontroller to process 1MIPS for every 4MHz of oscillator frequency up to a maximum of 20MHz for this specific microcontroller series (Microchip, 2003).

On board peripherals include ADCs, comparators, pull-up resistors, PWM modules, Universal Asynchronous Receiver and Transmitter (USARTs), timers, SPI (Master and Slave modes) and I^2C (Microchip, 2003).

Table 4: PIC16F87XA series specifications (Microchip, 2003).

Device	SRAM, bytes	EEPROM, bytes	I/O	10 bit ADC	PWM	SPI	USART	I^2C	Timers, 8 bit	Comparators
PIC16F876A	192	128	22	5	2	Yes	Yes	Yes	2	2
PIC16F877A	368	256	33	8	2	Yes	Yes	Yes	2	2

5.1.2 AVR ATmega16

The AVR, like the PIC, is based on the Harvard architecture, but have a reduced instruction set code (RISC) core running on single cycle instructions. This allows the micro to run at 1MIPS for every 1MHz and up to a maximum of 16MHz (Atmel, 2002).

The AVR range has a 131 instruction set, which is considerably more than the PIC. This allows for more efficient code to be written where space is limited. On board peripherals include internal oscillators (8MHz maximum), timers, USARTs, SPI (Master and Slave modes), I^2C (named Two Wire Interface [TWI]) pull-up resistors, PWM modules, ADCs and comparators (Atmel, 2002).

Table 5: ATmegaX series specifications (Atmel, 2002).

Device	SRAM, bytes	EEPROM, bytes	I/O	10 bit ADC	PWM	SPI	USART	I^2C	Timers, 8 bit	Comparators
ATMega8	512	256	32	8	4	Yes	Yes	Yes	2	1
ATMega16	1K	512	32	8	4	Yes	Yes	Yes	2	1

5.1.3 TI MSP430F2254

The MSP430 is a very low power micro specifically designed for low power applications that require fast mathematical computations. A RISC architecture is incorporated into a 16 bit CPU that operate with 16 bit registers (Texas Instruments, 2009).

Peripherals on this specific device include an internal oscillator (up to 16MHz), UARTs, SPI ports, I^2C, ADC's, PWM modules and timers.

An instruction set of 51 commands allows a midway between functionality and complexity between the Atmel and the PIC.

Table 6: MSP430X22X4 series specifications (Texas Instruments, 2009)

Device	SRAM, bytes	EEPROM, bytes	I/O	10 bit ADC	PWM	SPI	USART	I^2C	Timers, 16 bit	Comparators
MSP430F2254	256	512	32	12	3	Yes	Yes	Yes	3	2
MSP430F2274	256	1K	32	12	3	Yes	Yes	Yes	3	2

5.1.4 Summary

As has been shown, all microcontrollers can perform the same function although in a different way.

The following example shows a simple C program that is written and compiled for three different architectures and what the code size and execution time is for all three. The MSP430 is not included in this list as it has a 16 bit architecture and an older 8051 core was included.

```
int max(int *array)
{
 char a;
 int maximum=-32768;

 for(a=0;a<16;a++)
   if(array[a]>maximum)
    maximum=array[a];
 return (maximum);
}
```

Table 7: Three different architectures compared with respect to code size and execution time.

CPU Architectures	Code size	Execution time (cycles)
8051	112	9384
PIC16	87	2492
AVR	46	335

This example shows the AVR as having the smallest code size and fastest execution time, but being application specific, speed is not always the deciding factor. For this application the microcontroller will be in sleep mode most of the time to conserve power. The quicker it can wake up, take measurements and go back to sleep the more power will be saved (Raynus & Freidline, 2006).

Table 8: Current draw specification and pricing for the PIC, Atmel and MSP (Atmel, 2002) (Texas Instruments, 2009) (Microchip, 2003).

Specification	PIC16F877A	ATMEGA16	MSP430
Power consumption (8 MHz), active (mA)	5.5	5.5	4
Power consumption, low power mode (uA)	95	40	25
Cost per micro	R55	R28	R37

Pricing as was on Sunday, 27 December 2009.

After viewing the three possible microcontrollers and considering cost, speed and flexibility the Atmel AVR microcontroller was selected for the prototype system.

5.2 Communication Protocols

Communications between different parts of the system will be needed, therefore a quick overview of the different communication protocols will be done as well as how they compare to one another.

SPI, I^2C, USART and USB are all serial communication protocols. General concepts for serial communications are (Mathivanan, 2007);

1. Asynchronous communication does not have a clock line.
2. The same wires can be used for different devices depending on the protocol.
3. No slave can communicate with another, except through the master.

5.2.1 SPI

Serial peripheral interface (SPI) is a four wire full duplex synchronous communication protocol. It consists of three wires plus an extra select wire per device connected to the master.

The three main wires are master out slave in (MOSI), master in slave out (MISO), serial clock (CLK) and chip select (CS) / slave select (SS). The protocol was originally developed by Motorola for serial communication between their devices. The protocol is mostly used to connect microcontrollers to one another and to peripherals. Only one master can be active at any time that communicates to one or multiple slaves, although there may be many masters that act as slaves when not in master mode.

The number of wires required for operation is 3 + n, where n is the number of devices connected to the master. The master controls the clock of the system and the transfer speed is a function of the master's clock frequency.

Operation is as follows. Master selects the slave with which it wants to communicate by pulling the CS/SS line low. Data is shifted out serially through the MOSI port, from the master to the slave, and received through the MISO port, from the slave to the master. This happens bit by bit, until one byte has been sent. As data is sent, it is also received through shift register operation. Thereafter the next train of bits commence until the next byte is finished or until all the data is sent and received and thus information is interchanged between master and slave (Mathivanan, 2007).

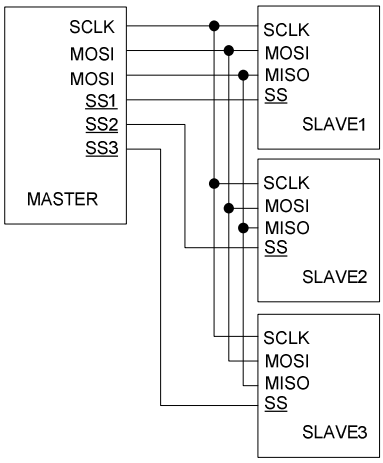

Figure 43: Master connected to three slaves with communication through the SPI protocol (Philips, 2003).

5.2.2 I²C

Inter IC (I^2C) was originally designed by Philips to provide a way of communication between multiple ICs using the minimum number of pins and wires. The protocol has become standard throughout the embedded industry and many manufacturers support this protocol (Philips, 2003).

I^2C operates via a two wire bus. The two wires are serial data (SDA) and serial clock (SLC) and communication is half duplex. The bus is able to operate with multiple masters, of which only one is active at any time, which operate on an arbitration feature. Each connected peripheral has its own unique address, up to a maximum of 127 addresses. Dedicated peripheral devices contain complete built in communication interfaces.

The master initiates the data transfer process by signalling a start condition. The master then chooses the slave with which it wants to communicate by transmitting its address via the SDA line. The slave has to respond by sending an acknowledge signal on the SDA line. If the acknowledge signal was not received the master chooses some predefined path. It can either resend the address or move on to another device or action. If acknowledgement from the slave was successful, data can be sent via the SDA line, eight bits at a time. The slave will then have to send another acknowledge signal to let the master know that the transfer was successful or not. If the transfer was not successful the master can resend the data or follow another pre-defined path.

Acknowledge signals do not guarantee data integrity, it guarantees that reception of transferred data was done correctly. The master can keep transmitting data if it is necessary or cease the transmission when no more data needs to be sent, after which it will send a stop condition on the

SDA line and again choose some predefined path. Data can be transferred up to rates of 400 kHz in normal mode, with no lower clock limit.

If the master is faster than the slaves, the slaves can stretch the pulse (slow the master down), by keeping the SCL line low for a short duration of time, until it is ready to receive or send data (Mathivanan, 2007).

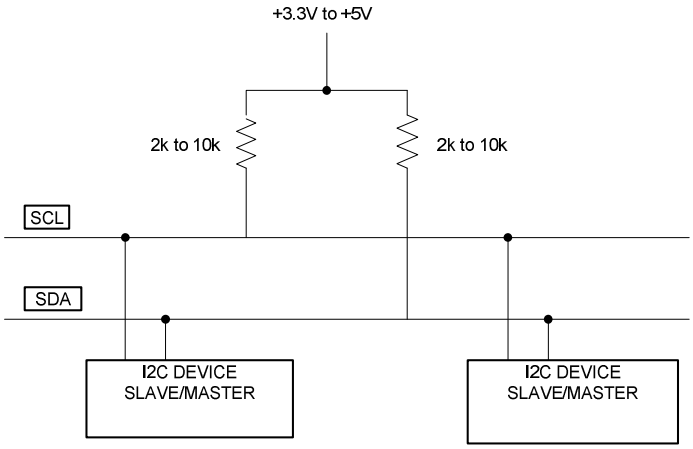

Figure 44: Master and a slave connected on an I^2C bus (Philips, 2003).

5.2.3 USART

The universal synchronous and asynchronous receiver and transmitter (USART) was first implemented in the 60's. Since the protocol has been used for a lengthy period, it is well understood and well supported, although it is now almost fully replaced by USB. Simple RS-232 to USB converter cables allow for easy integration to new systems. Data transfer speeds of up to 1Mbits/s can be achieved using this protocol.

Devices can operate in either asynchronous or synchronous mode. In asynchronous mode the microcontroller and the device which is being communicated to have to agree on a speed before the data is sent. This predetermined speed is referred to as the BAUD (bits per second) rate. Data that is sent out serially is preceded by a start bit. The data then follows the start bit, bit by bit, for data lengths of six, seven or eight bits (which is predetermined), followed by an optional parity bit (to check data integrity) before the stop bit (or two) is sent (Mathivanan, 2007).

5.2.4 USB

Universal serial bus (USB) was originally created to connect peripherals to a single master and to replace older ports with a universal one. One master can have up to 127 connected slave devices through multiplexing.

The protocol is “plug and play” compatible with standardised ports and cables. USB connections are now commonly available, achieving what it was meant to do. Devices configure themselves when plugged into a master.

USB converters are widely available to convert signals from the other protocols to USB and vice versa.

5.2.5 Summary

Table 9 show the maximum values for different parameters of the protocols. Protocols like CAN and IEEE 1394 that have not been discussed are also included for comparison purposes.

Table 9: Available designed protocols with their specifications (Philips, 2003) (Mathivanan, 2007).

Bus	Data rate (bit/sec)	Length (meters)	Length limiting factor	Nodes Typ. number	Node number limiting factor
I^2C	400k	2	Wiring capacitance	20	400pF max
I^2C with buffer	400k	100	Propagation delays	Any	No limit
I^2C high speed	3.4M	0.5	Wiring capacitance	5	100pF max
CAN 1 wire	33k	100	Total capacitance	32	Load resistance and transceiver current
CAN differential	5k	10km	Propagation delays	100	
	125k	500			
	1M	40			
USB (Low-speed 1.1)	1.5M	3	Cable specs	2	Bus specs
USB (Full-speed)	1.5/12M	25	5 cables linking 6 nodes	127	Bus and hub specs
Hi Speed USB (2.0)	480M				
IEEE 1394	100 to 400M	72	16 hops, 4.5M each	63	6-bit address
SPI	>100k	<10	Wiring capacitance	Any	Master outputs
USART	20	30	Wiring capacitance	2	Bus specs

With the exception of the CAN and USB protocols, all the communication protocols are slow, as well as short distance. This allows them to be successfully implemented on PCB without any special design considerations.

All these protocols can be designed and implemented using a programmable controller.

Having reviewed communication protocols that will be used (except I^2C that will be used with the MAX11068) to transfer data to and from different parts of the system, the programming language will be discussed next.

5.3 Programming Language

The most widely used programming languages for microcontrollers are Assembler and C. Every manufacturer has their own compiler and there are numerous software developers that support the different microcontroller manufacturers. Assembler is considered where specific timing loops or precise control is required, but with modern C compilers timing and code space are comparable to assembler language.

5.3.1 Assembler

Assembler code generally takes one machine cycle to complete each instruction. This depends on the code instruction as well as on the architecture of the device as some instructions do take longer.

Because the code is directly converted to machine language the code size is smaller than higher level design languages and specific timing can be achieved.

5.3.2 C

C is a high level design language (“Black Box”) and is used throughout the embedded industry. C was written by Dennis Ritchie and later on he and Brian Kernighan co-wrote a book called “The C Programming Language” which is a commendable book for learning the C language (Kernighan & Ritchie, 1988).

5.4 Programming Interface

5.4.1 AVR Studio 4.17

Since the ATmega16 will be used; Atmel’s own software, specifically designed for their microcontrollers, will be used. It is based on the C language and code can be written, compiled and programmed through AVR Studio 4.17.

Having all three those functions integrated into one program decreases programming time and allows for a user friendly program. The software is called AVR Studio 4.17 and can be downloaded for free from their website after registration.

5.4.2 Win AVR 20090313

Functions are needed to perform the different operations within the microcontroller. These can range from putting the microcontroller into sleep mode, using the USART or even a basic millisecond delay routine. Instead of having to rewrite all the functions a program like Win AVR can be downloaded with the most used functions specifically for the AVR series prewritten.

With the downloaded file you get a compiler, programmer and a debugger, but as all this is included in the AVR Studio they will not be used. Using these functions it is possible to complete any programming in less time. Win AVR can be downloaded for free from their website.

5.5 Programming of the ATmega16

Programming of the ATmega16 was done through a 6 pin in system programmer (ISP). The ISP is known as the AVR ISP MKII and was used to program the code onto the microcontroller as well as read data from the microcontroller.

The ISP programmer is plugged into the USB port of the PC and the 6pin connector is plugged into the on board connector of the development board or the prototype.

The speed of the programmer can be set up to a quarter of the speed of the microcontroller i.e. if the microcontroller operates at 8MHz the programmers maximum communication speed can only be 2MHz. Speed was however set to 1MHz which allowed the device to be programmed in about one second.

5.6 Cell Simulator

To obtain twelve cells of different chemistries, or at least four, which is the minimum for this IC, would have been expensive. A cell simulator was thus developed to simulate the cells, and more specifically all the cell chemistry types.

The cell simulator can be adjusted via a potentiometer to obtain voltages as low as 0V (when the potentiometer is shorted) and over the maximum rated voltage of 5V by decreasing all except one potentiometer.

Each simulator board would be supplied from a different power supply and each board would simulate twelve cells.

The schematic is shown in figure 54 and is discussed with the system design.

5.7 Simulation of Discharge Switch Tests

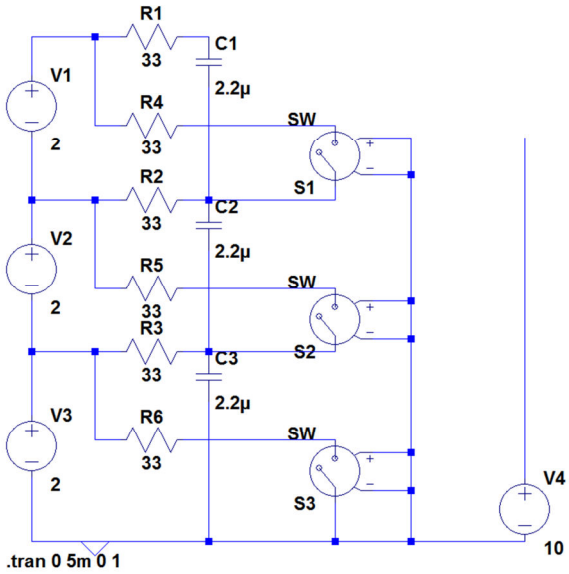

Figure 45: Bottom three cells with filters and simulated internal switches of the LTC6802.

Figure 45 shows three cells with their switches, internal to the IC. The reason for simulating this condition is that the internal discharge switches will be used during charging operation of the stack. The problem with using the internal switches while measuring the cell voltage (STCVDC command – start cell voltage A/D conversions and poll status with discharge permitted) is that false readings will be taken. If the switch is on, the voltage over the switch will drop accordingly and an inaccurate reading will be made for the cell being measured as well as the cell right below the one with the switch enabled.

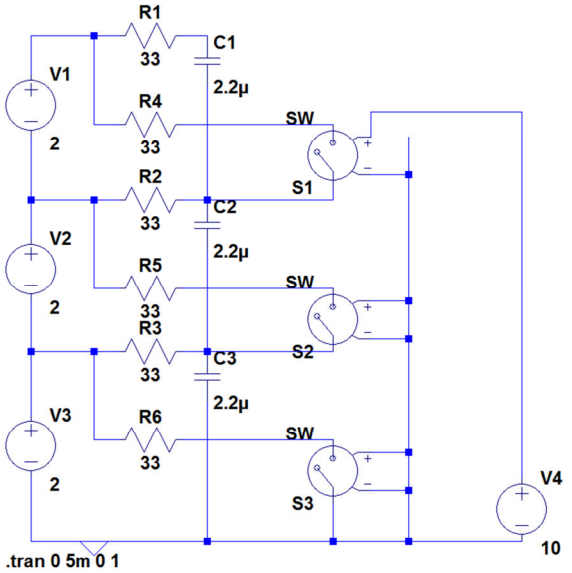

Figure 46: Top cell's (V1) discharge switch activated.

When the discharge switch of V1 in figure 46 is activated the voltage over the switch should drop as well as the voltage over cell V2. This is because if the switch is closed, R4 and R2 are in parallel with the cell. A voltage divider is thus formed and because the values for both filter resistor and discharge resistor are the same the value that is measured is half of the cell's voltage i.e. 1V. This assumes that the switch itself has no resistance, which it does.

Since there is now also a 1V drop over R2, cell V2 voltage is increased by this amount. This gives a reading of 3V for V2. The value of V3 remains the same and so would a cell that was allocated higher in the stack than V1.

This shows that the cell over which the discharge switch is located, as well as the cell right below the cell being discharged, will show a different measured voltage than actual voltage.

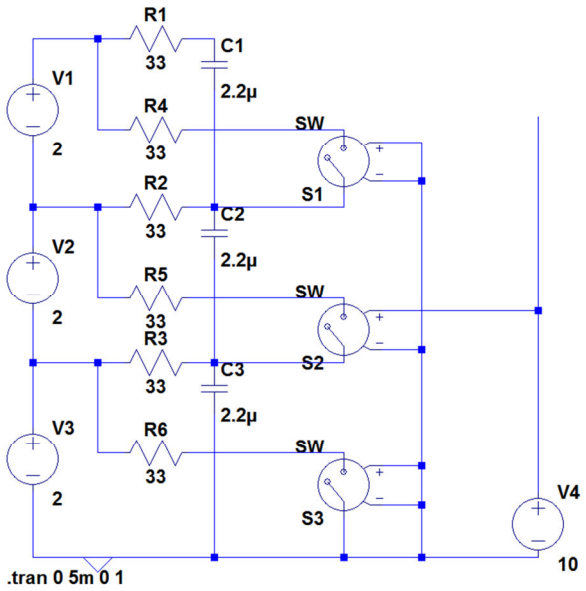

Figure 47: Middle cell's (V2) discharge switch activated.

When the next cell's (V2) discharge switch is activated (figure 47) the same can be seen happening to its measured voltage as well as V3s voltage.

V2s voltage is decreased by 1V while V1s voltage is increased by 1V.

It should be noted that the voltages used here were all equal and switch resistance was ignored and as such discharge would not have been used. The value that is added or subtracted is a function of the discharge and filter resistor as well as the voltage of the individual cells and the switch resistance.

Since discharge switches can be used in the charging and discharging cycle, all cells will first be measured and then discharge switches will be turned on (STCVAD command – start cell voltage A/D conversions and poll status). This will allow quick accurate measurement of the cells and, after some

calculations the discharge switches can be enabled and the overcharged cells can be equalised for two seconds. After a reset, the switches will be turned off, measurements taken and the appropriate discharge switches reactivated until different operation or use of the stack is required.

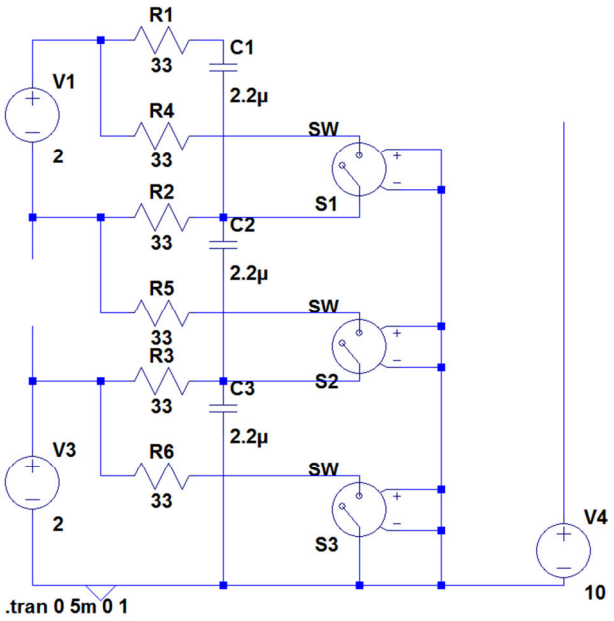

Figure 48: Cell 2s (V2) voltage disconnected.

If one of the cells is disconnected (figure 48), either intentionally or by accidental disconnection, it should be measureable and not cause any problems to the stack.

The ADC measurement with the disconnected cell will show a value of 0V. The only problem this will cause is that the discharge switch directly below the disconnected cell will permanently be on, because of the proposed algorithm, however the system will give notice of this in the measurement readings and action can be taken to rectify the problem.

5.8 Development Board

As the ATmega16 microcontroller was not well known, a development board was made. The development board that was available for testing was the STK500, but as the development board's layout was different from the proposed system layout a new development board was made.

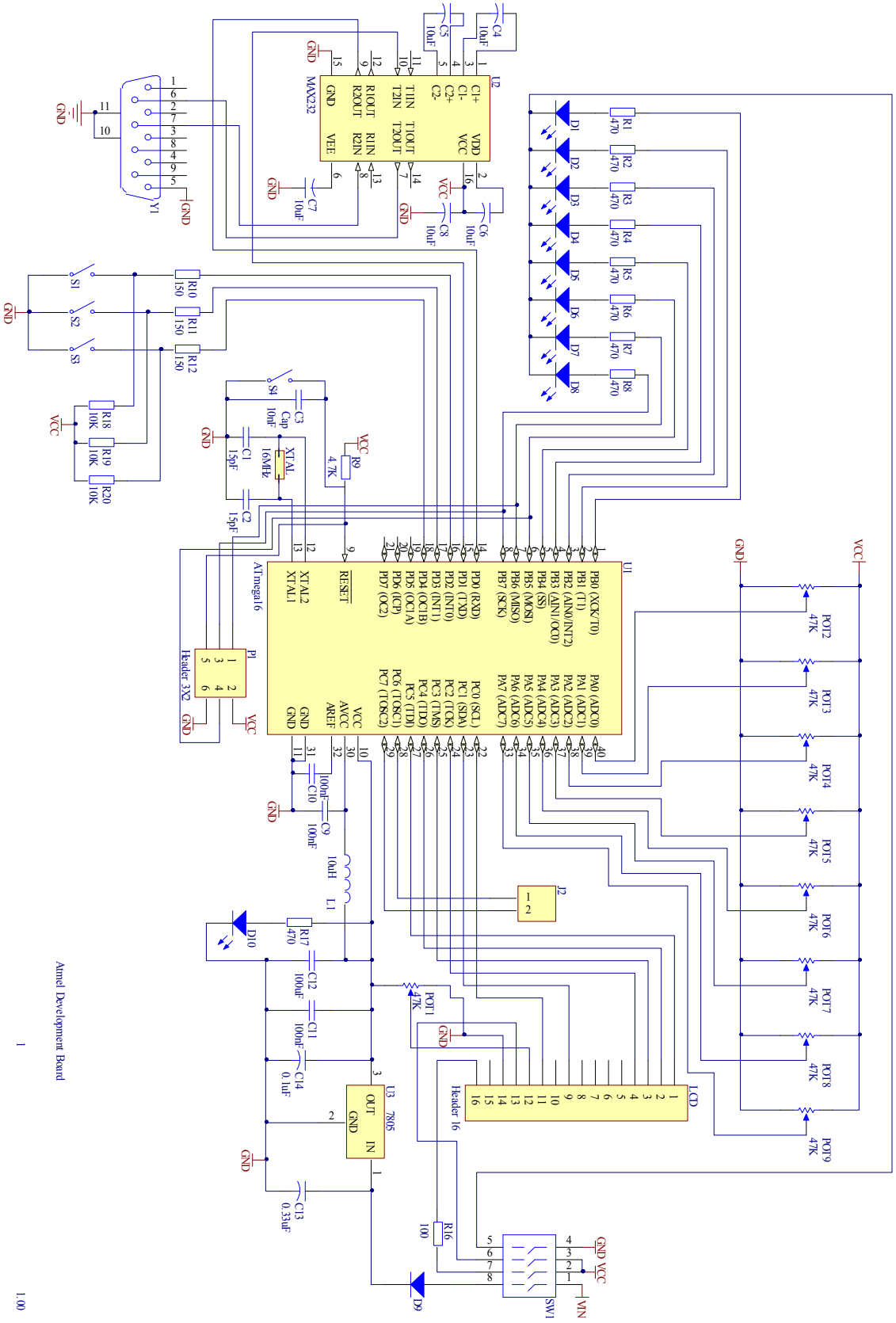

Figure 49: Schematic of the development board for ATmega16.

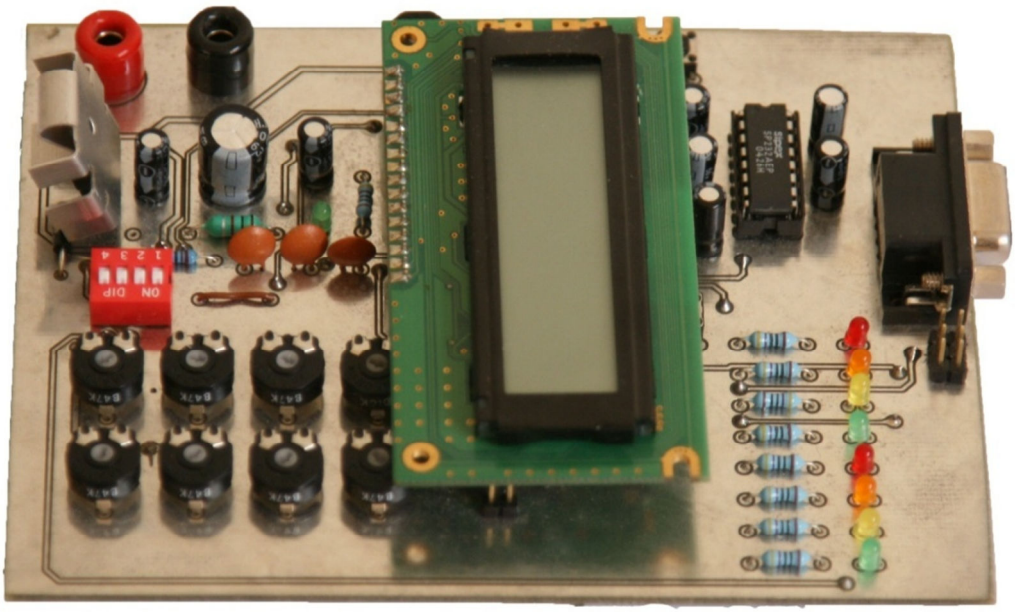

Figure 50: Development board used for tests.

Figure 49 and 50 show the schematic and the actual development board respectively. This allowed experimentation of the microcontroller with the different on board peripheral devices as well as the in system programmer with the microcontroller.

5.9 Data Logging History

During the normal use of the stack, the cell voltage will vary depending on how the stack is being used. This data is important to keep because of the information it tells about the stack and cells.

5.9.1 Instantaneous data

This will show the immediate state of the battery stack as well as individual voltages. This can be updated once every five hundred milliseconds, one second or once every two seconds. This data will be displayed on the screen via a graphical user interface (GUI) and could be saved to file for future analysis.

Because of the limitations on memory space in the microcontroller, the bulk of the data will be stored on a PC. Maximum and minimum values will be saved on the microcontrollers EEPROM and can be downloaded at any time. This allows a more economical design by saving data on a hard drive rather than in the microcontroller or on external flash memory.

5.9.2 Graphical User Interface

The graphical user interface will consist of an Excel spread sheet with a Visual Basic overlay. The first sheet will contain all the command buttons and features. The following sheets will contain information on the cells as well as graphs.

The required information the GUI needs to display about the cell stack is the following:

1. Individual cell voltages.
2. The overall stack voltage as calculated from the individual cells.
3. The temperature of the LTC ICs.
4. Over and under voltage as obtained from the controller.

Buttons will be required for the following functions:

1. Start button – To enable the system to interact with the controller and display the measurements (There must be clear indication on-screen of operating state).
2. Stop button – To disable the display of measurements even though the controller still takes them (There must be clear indication on-screen of operating state).
3. Log button – To enable the software to log information obtained from the controller.
4. Clear button – To allow the user to clear any unwanted data.

5.9.3 Saved data

Cell data that is captured using the GUI can be exported to a MS Access database. This data can later be manipulated and used as required.

5.10 Conclusion

A quick summary of the design components will be done.

Instead of buying twenty four cells of different chemistries a cell simulator will be used that will be able to simulate Lead Acid, Ni-Cd, Ni-MH and Li-Ion cells. To measure the different cell voltages a battery management IC, the LTC6802-1, will be used. Since one IC can measure twelve cells, two will be required.

Communication to and from the device will be done via a microcontroller, the ATmega16, from Atmel. It has all the necessary functionality and peripherals needed for this system. The microcontroller will be programmed with the AVR ISP MKII programmer with code written and compiled by Atmel's AVR Studio 4.17.

Communication to and from the PC will be through a RS-232 port and all data will be shown using a GUI. Data will be saveable for later use.

6 Chapter Six: Design

Connections for the LTC can be derived from the LTC6802-1 datasheet and will only be briefly explained here (Linear Technology Corporation, 2009).

The schematics are divided into three sections. The first schematic is the microcontroller interface board. The second schematic is the bottom LTC IC with cell simulator connector. The third schematic is the top LTC IC with cell simulator connector. The design allows for testing one LTC IC and if favourable results are obtained, a second IC can be added to allow a system that can measure twenty four individual cell voltages.

Figure 51, shows the microcontroller with its various connections. The prototype board was designed to run from two different power supplies for initial tests. The power supply has a constant voltage of 10V, so a simple linear voltage regulator with filtering was adequate to supply all the components on the board.

A DB 9 connector with a MAX232 IC was used for communication with a PC via the serial port. This was chosen as the initial setup. USB or other wireless communication can later be implemented with ad-on-modules. A USB to serial converter cable was used for testing. As this cable simulates a serial communications port the system would be able to operate on PCs that do not have a RS-232 port.

A standard 6 pin ISP port was included to allow for on board programming and software updates. Indication LEDs were included for software testing and to be used later on to indicate when the system is running and when the system is in sleep mode.

Figure 52 show the bottom of the stack LTC IC. Note the minimum amount of external components that the IC requires to operate. A 13 pin connector is used to connect the cell simulator to the LTC. This is a plug in unit and can be removed or changed if damage occurs to the cell simulator board or to plug in an actual battery stack.

Isolation and data transfer from the top device to the bottom device and vice versa is done via the three diodes on the communication lines. These are low voltage, low rating fast switching diodes.

Figure 53 show the second LTC IC that measures the top of the battery stack. This IC as well as the bottom IC was made, so that by changing some components, one or both could be used. For initial tests only the bottom most IC was used.

Figure 54 show the cell simulator. This circuit allows any cell voltage to be simulated. The cell simulator has no definite ground or power connection. The power connections are dependent on where the simulators are with respect to another one.

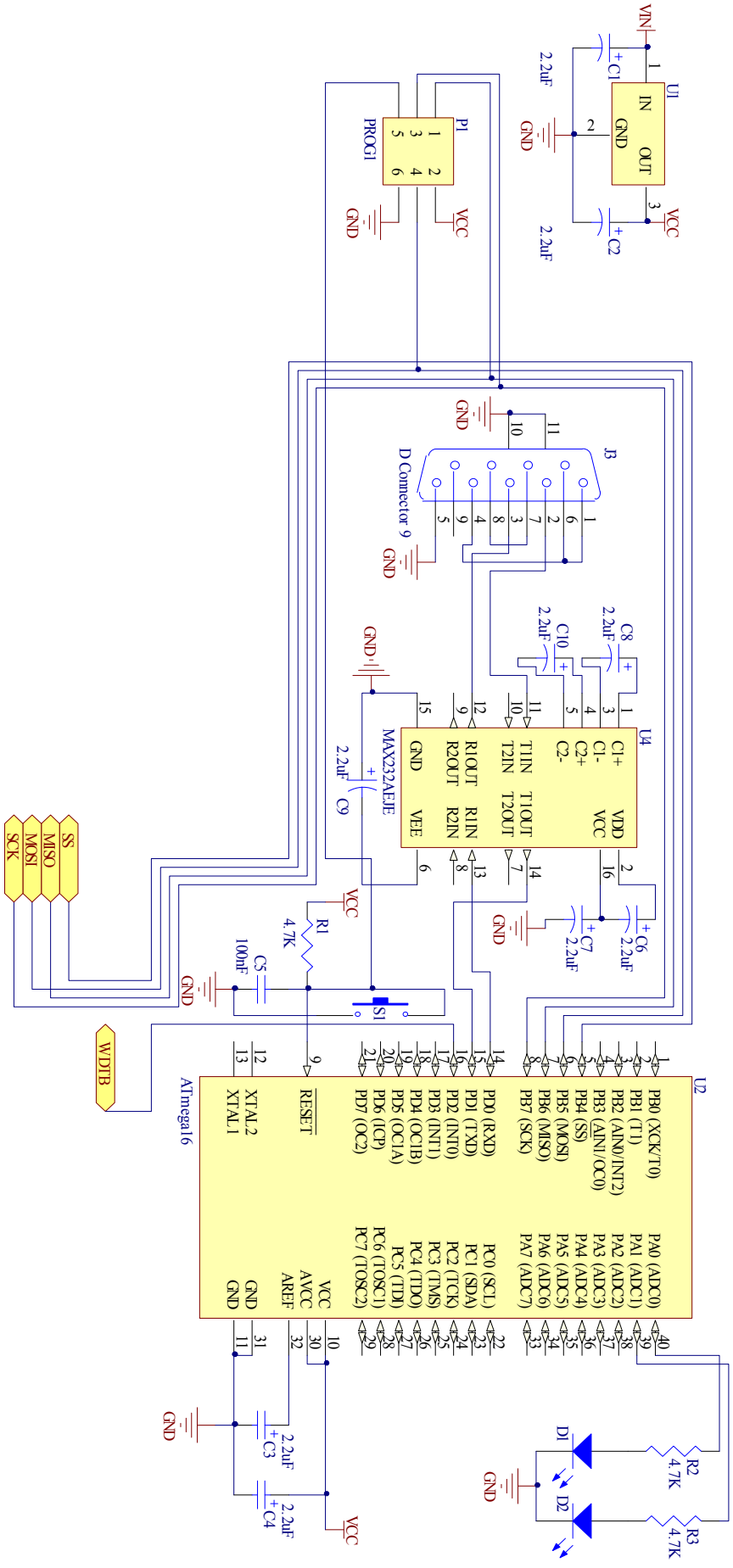

Figure 51: LTC integration circuit with microcontroller, indication LEDs, power circuit and RS232 IC.

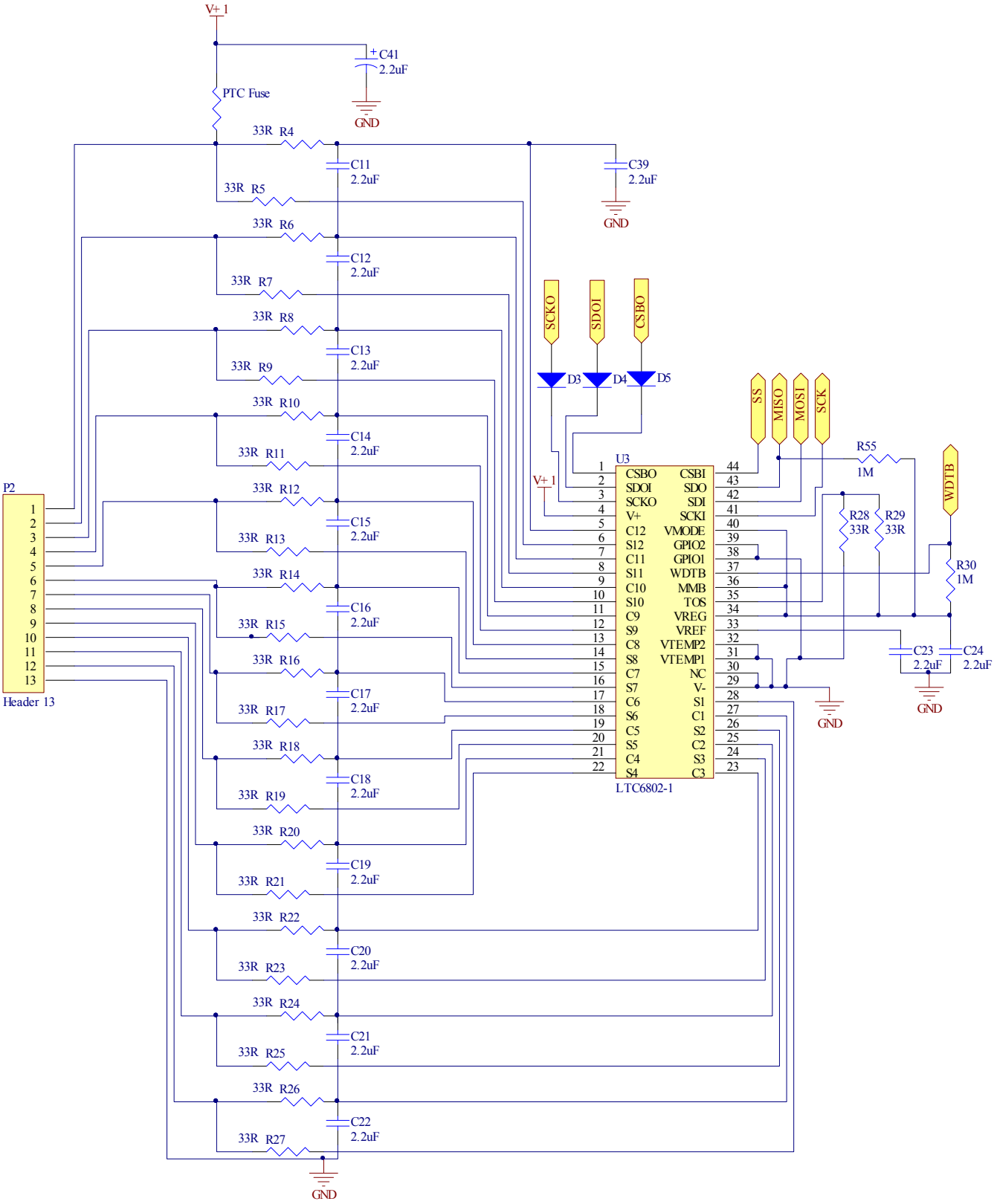

Figure 52: Bottom LTC IC with connector and filter section.

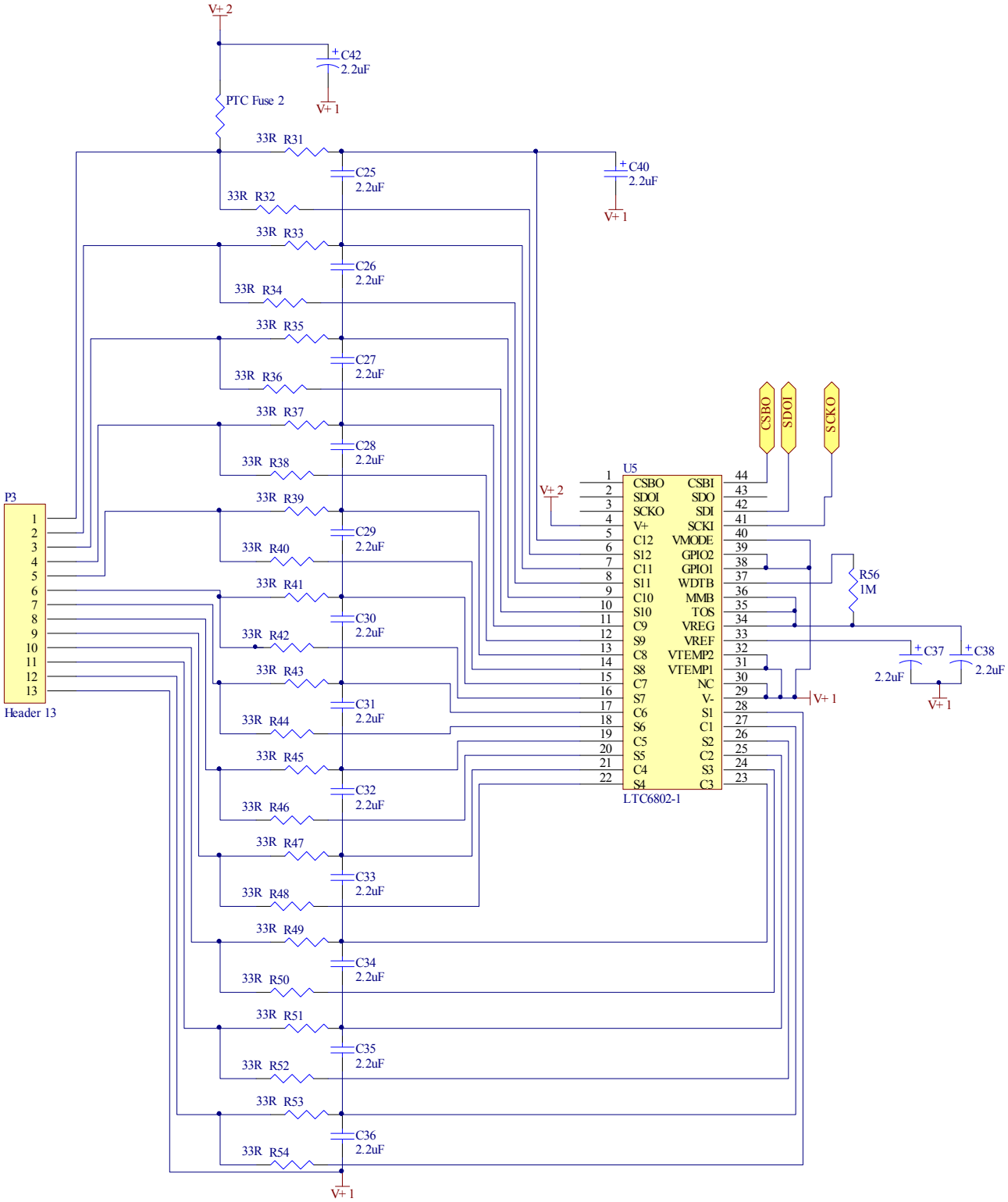

Figure 53: Top LTC IC with connector and filter section.

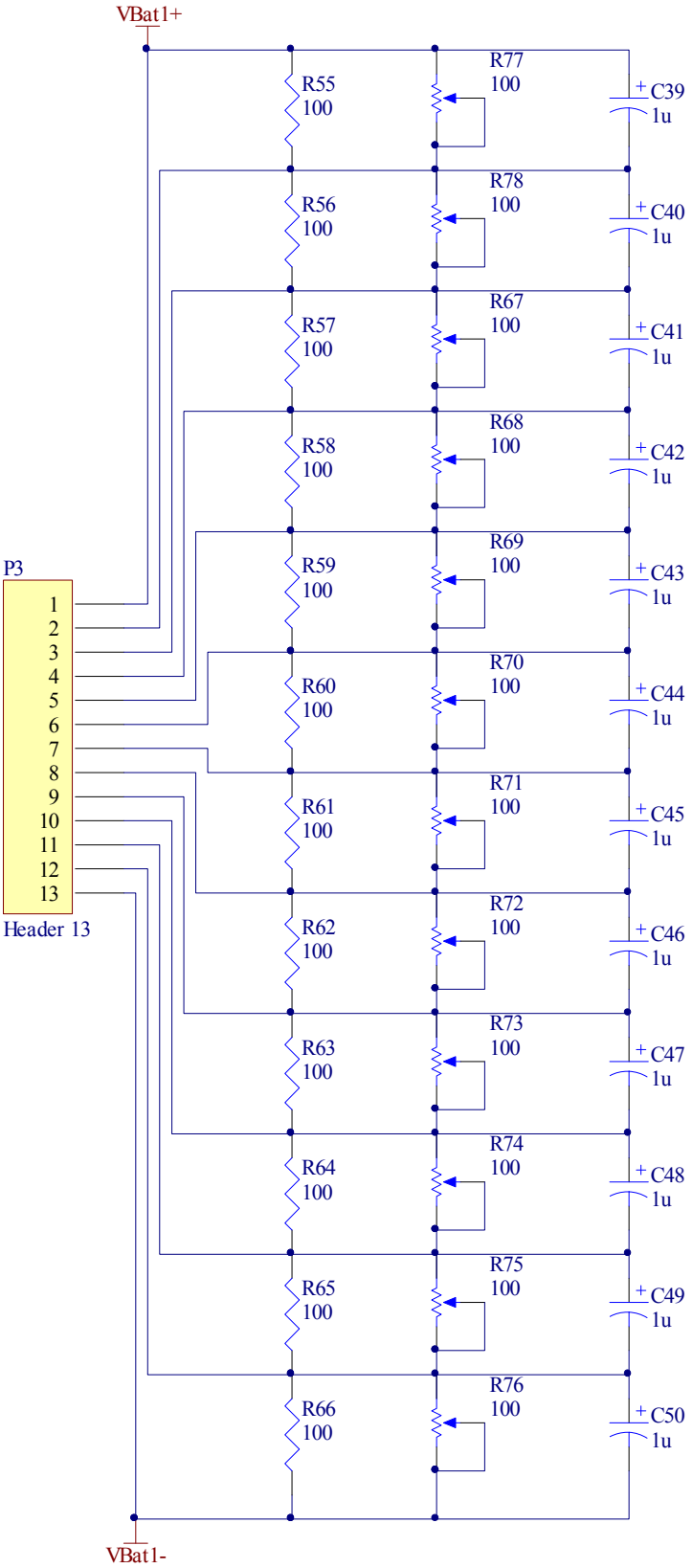

Figure 54: Cell simulator schematic used to simulate different cell type chemistries.

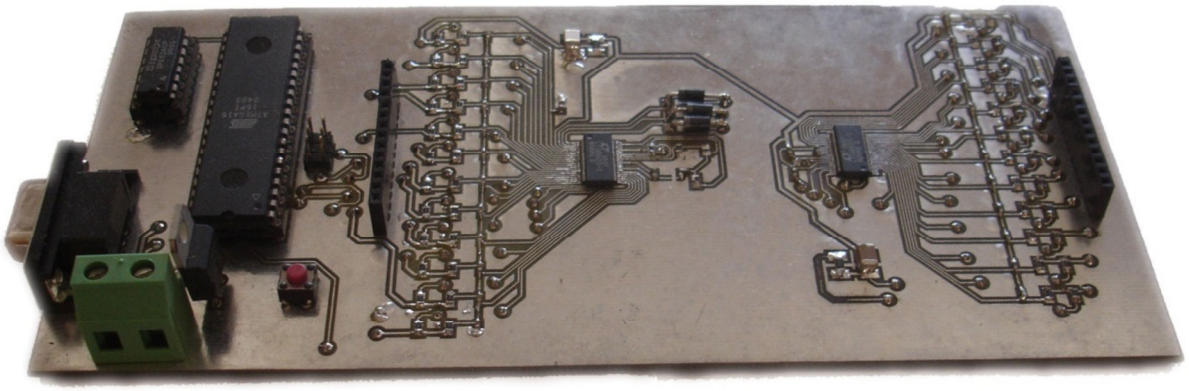

Figure 55: Prototype board without cell simulators

Figure 55 shows the prototype board and layout with regard to components and space. The board allows measurements to be taken without accidental shorts being made.

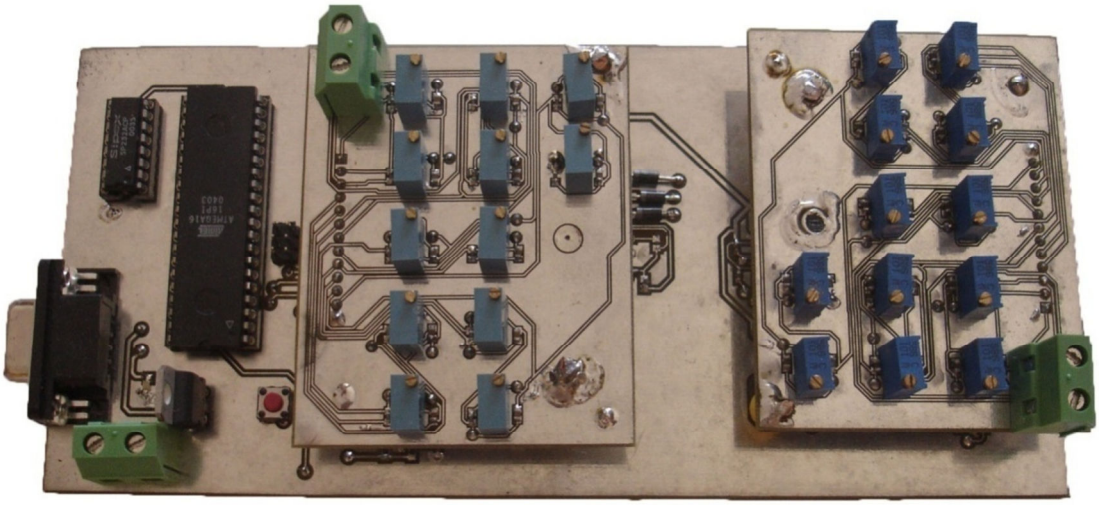

Figure 56: Prototype board with two twelve cell simulators connected.

Shown in figure 56 is the full prototype board with cell simulators used for all initial tests. Due to the space consumed by the through-hole components, the prototype board is considerable in size. The most noticeable are the MAX232 and ATmega16 ICs. However, if the board size ever needs to be reduced it can be done by replacing the components with smaller SMT components and the layout of the board can be redone.

Overall this prototype board produced excellent results and never gave any problem throughout all the tests and experimentations.

7 Chapter Seven: Programming

Programming of the microcontroller was a major part of this thesis. The task was quite complex and programming cannot be done without a definite flowchart. The first step in the programming algorithm was done in the design methodology section. This set an outline of what is needed from the system. The code was divided into parts and a flowchart for each section is drawn.

7.1 Flowcharts

Flowcharts were used to simplify coding algorithms and to create a logical programming code. Flowcharts were only created for the C program. A short discussion of the main parts of the flowcharts will follow.

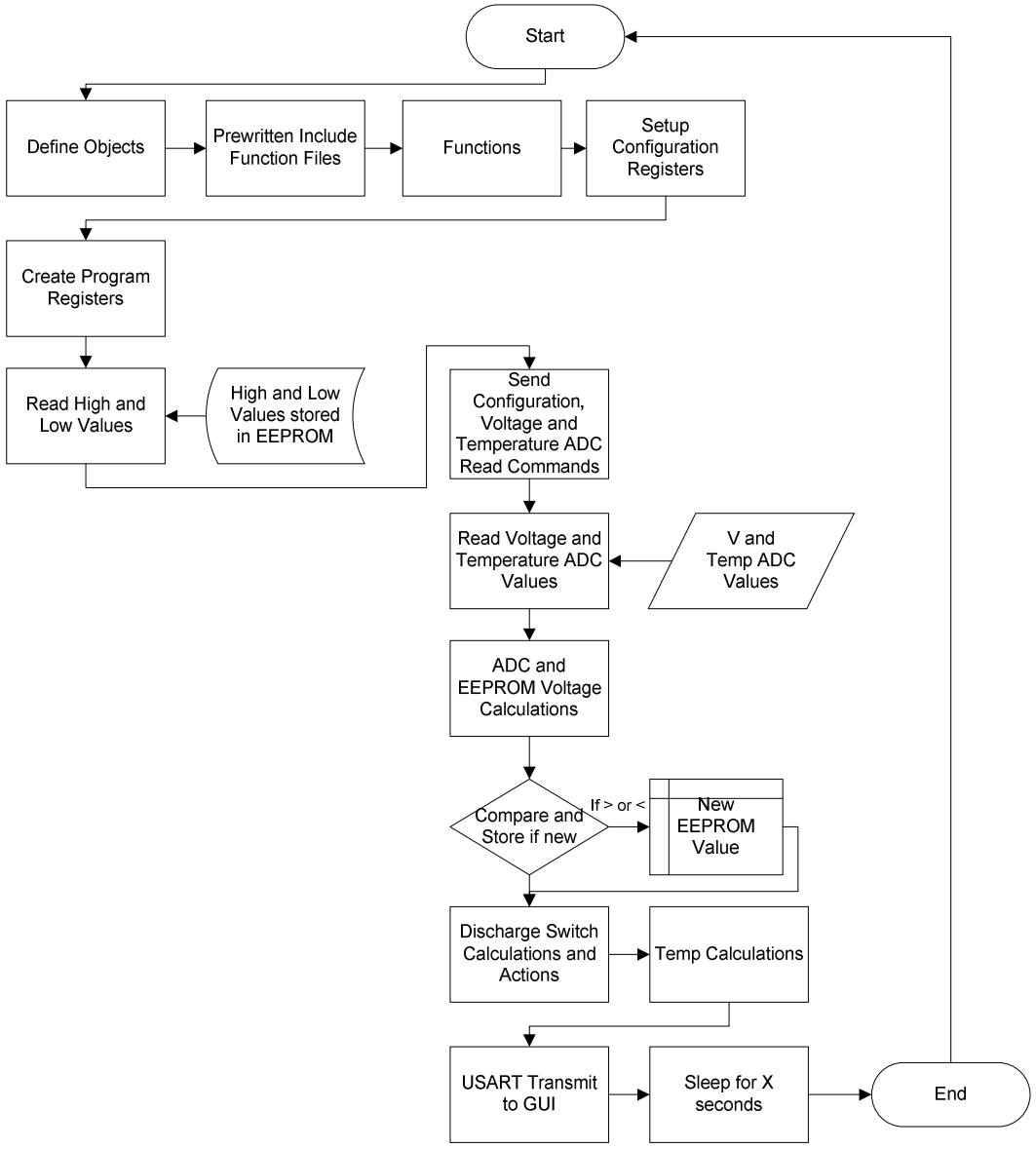

Figure 57: Full simplified flowchart of C program.

Figure 57 shows the full simplified flowchart diagram. Each step that the program follows is included and basic operation of the program can be followed.

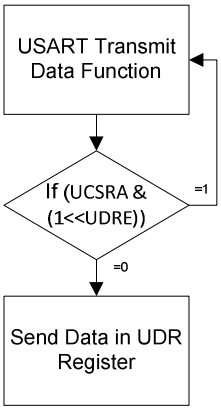

Figure 58: USART transmit function that transmits data from the data register via the USART peripheral.

Figure 58 shows the USART transmit function used throughout the program and which is used to perform USART communications. Data is passed to the function and the function checks if any data is being sent. If data is being sent it loops until the USART data register empty (UDRE) bit is cleared where after it sends the data stored in the USART data register (UDR) register via the USART port.

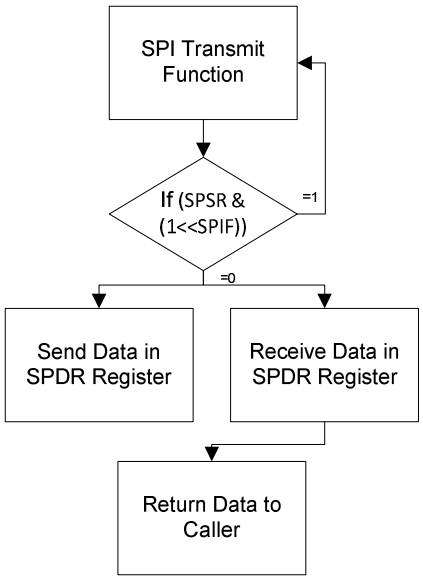

Figure 59: SPI transmit function that transmits and receives data simultaneously via the on-board SPI peripheral.

The SPI transmit function starts by checking if any data is currently being sent by monitoring the SPI interrupt flag (SPIF) bit. If data is being transmitted it loops until the bit is cleared. When the bit is cleared the function transmits data from the SPDR register, while also receiving data in the SPI data

register (SPDR). If any data was received it can be found in the SPDR register that the function returns.

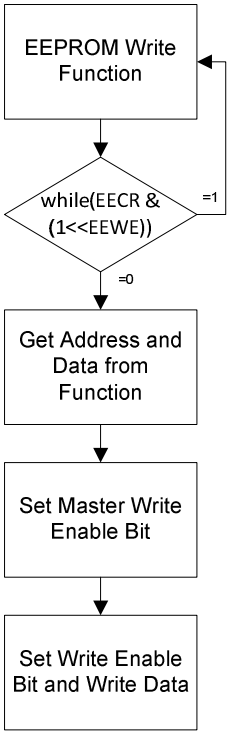

Figure 60: EEPROM write function that writes data to a specific address in the EEPROM.

The function starts by checking if any data is being written to the EEPROM by monitoring the EEPROM write enabled (EEWE) bit. If data is being written, the function loops until the EEWE bit is cleared. When calling this function, an address and data variable must be passed from the caller. The function uses these two values to know what and where to write inside the EEPROM.

Before data can be written a sequence of events must take place. This is done to ensure that data is not mistakenly written to the EEPROM, overwriting important values.

The sequence requires the master write enable bit to be set. After this bit is set the write enable bit must be set within the next four clock cycles otherwise the data write will be invalid. If the write enable bit was set within four clock cycles then data that is stored in the EEPROM data register (EEDR) will be written to the set address.

The read function, shown in figure 61, starts off by checking if data is being written to the EEPROM by checking the EEWE bits status. If data is being written the function loops until the bit is cleared. If the bit is cleared the function uses the address that was passed to it by the caller to read the value of the specified address. The address is loaded into the EEAR register where after the read bit is set enabled and the value present in the EEDR register is returned to the caller function.

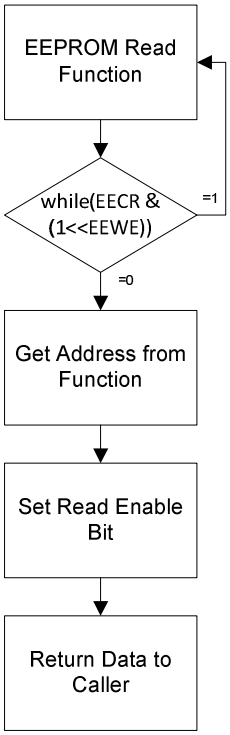

Figure 61: EEPROM read function that reads a value from a certain address and returns the value to the caller.

The calculations of the cells differ from one another in that even and odd cell values are calculated in different ways.

The even cells will be discussed first and the flowchart is illustrated in figure 62. For this process to take place two things should have already happened. They are:

1. ADC reading of cell voltages should have been measured, retrieved from the LTC and stored internally in the microcontroller's memory.
2. The high and low values of all the cells should have been read from the EEPROM and stored in their individual variables.

The register is eight bits wide and the ADC value is 12 bits. This means that some bit manipulation will have to take place and the flowchart shows this process.

The values of the voltage ADCs are loaded into two temporary registers called Temp1 and Temp2. This should not be confused with temperature reading variables. Temp3 is created by masking the upper 4 bits of Temp2. Cell1 is then given the value of Temp3 shifted left by 8 bits and adding Temp1. This gives Cell1 a value of 16 bits of which only the 12 LSB contain information and the 4 MSB contains zeros.

This is done exactly the same for CellHV1 and CellLV1, which are respectively the highest and lowest value obtained by cell 1 over the entire operational time of the system. The current cell value is then

compared to the stored highest and lowest values. If the two differ, the new value is written to the EEPROM and the program continues to the next cell value.

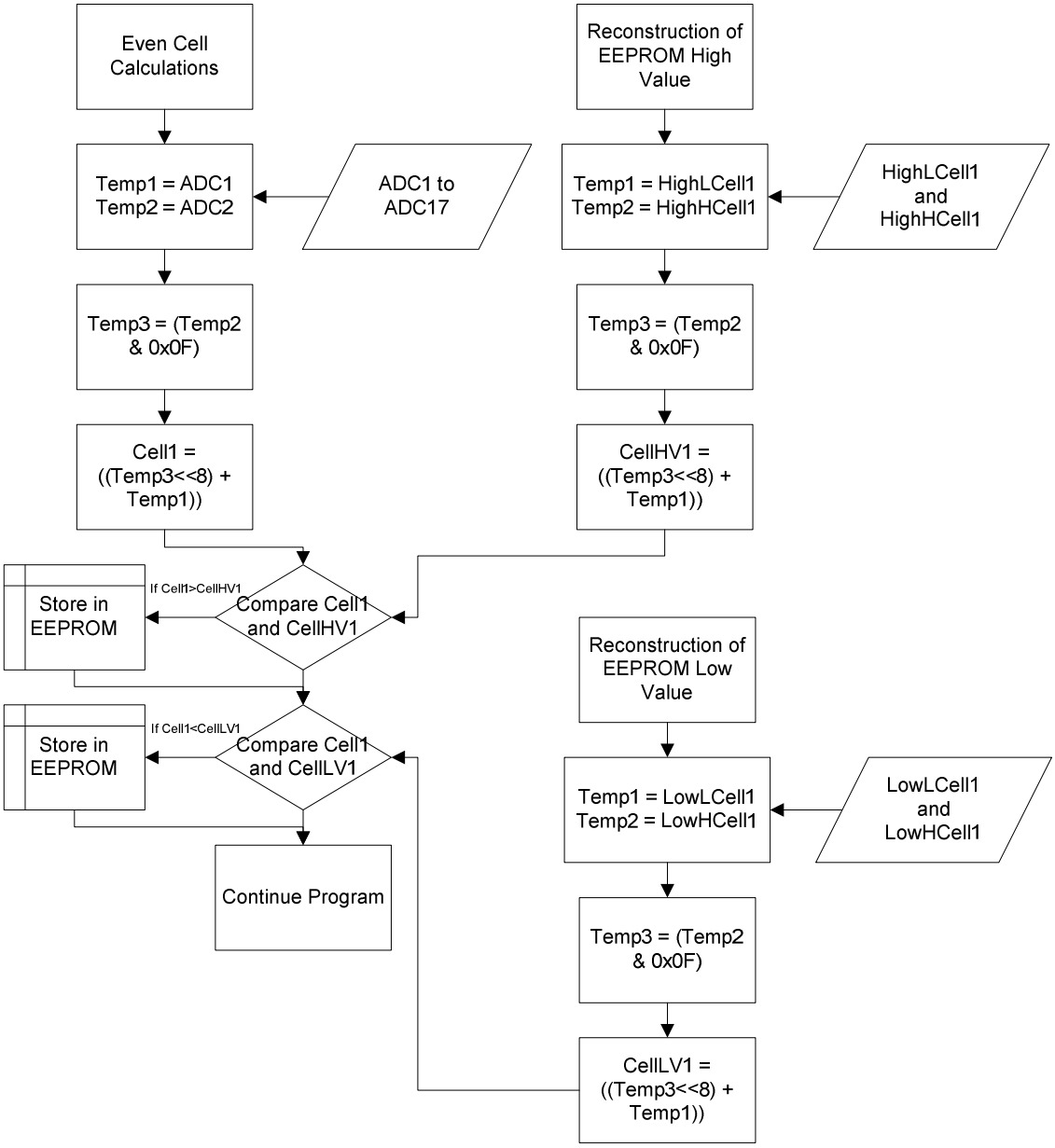

Figure 62: Process that reconstructs even cell voltages and then compare them to new values to see if they should be written to the EEPROM.

Figure 63 illustrate the calculation of the odd cells. The calculations for the odd cells are very similar to the even cells, but with noticeable differences. The differences come from the way the LTC stores the cell voltage ADC values in its registers. The same conditions hold for the required values as it did for the even cells. Since the program is similar only the parts that change will be discussed.

Temp1 and Temp2 still receive ADC values but now Temp1 receives the value of ADC2 and Temp2 the value of ADC3. Temp3 is the value of Temp1 with the 4 LSB masked. The value of Cell2 is then Temp4 shifted left by 8 bits and adding Temp1. The value of Cell2 is then right shifted by 4 so that

the register presents data in the same way as did Cell1. The 4 MSB contains zeros and the twelve LSB contains the voltage data of cell2.

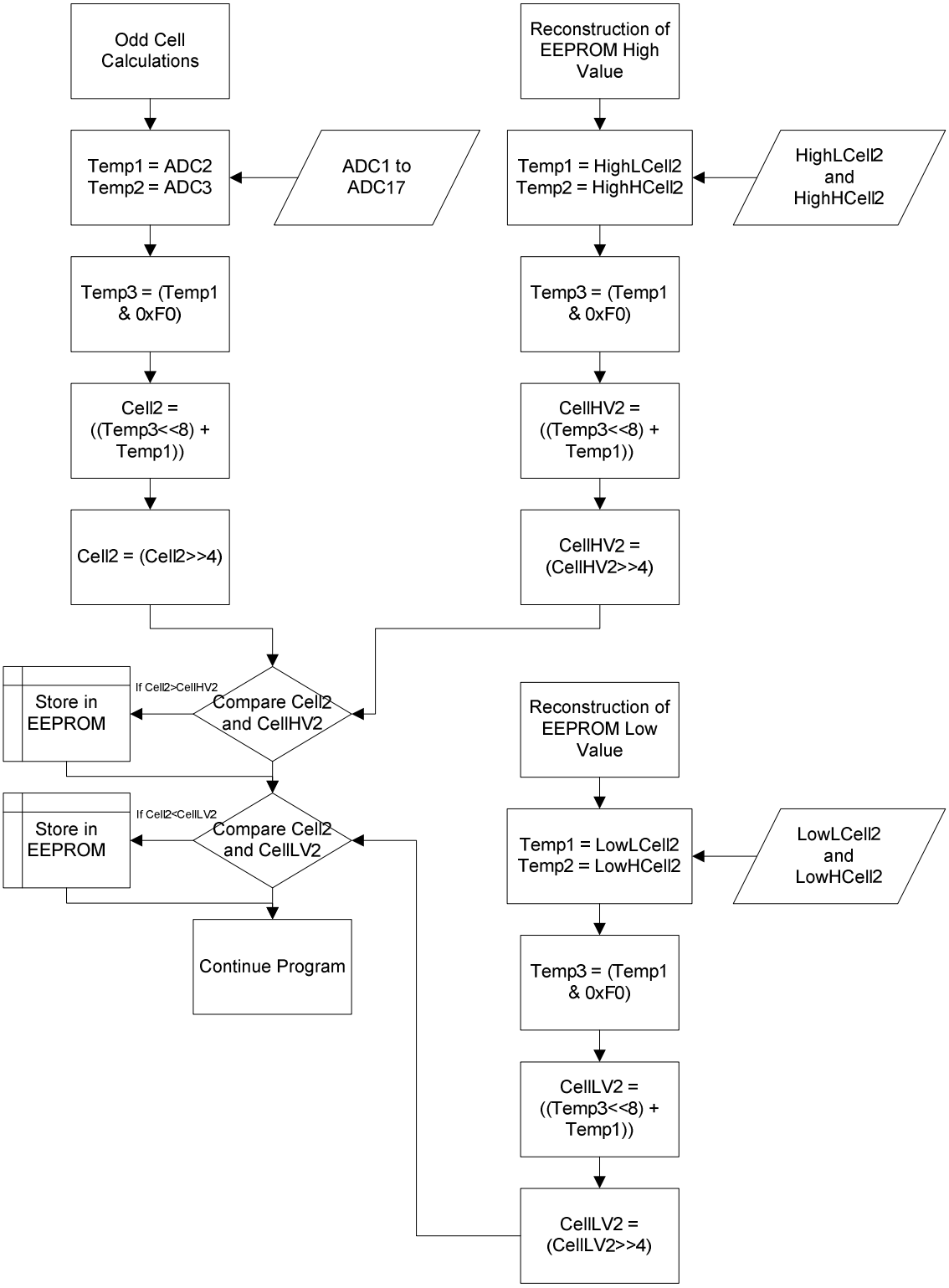

Figure 63: Process that reconstructs odd cell voltages and then compare them to new values to see if they should be written to the EEPROM.

The comparison of cell values and the processes that follow are exactly the same as for the even cells.

This process is done for all the cells in the stack after which the code proceeds to the next section.

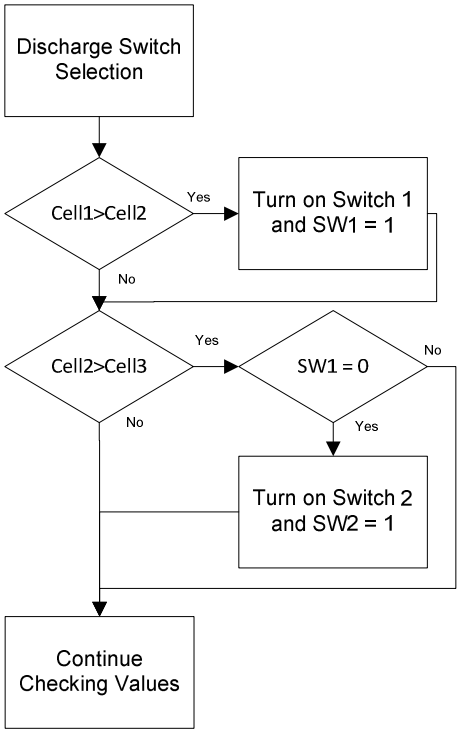

Figure 64: Process that checks if discharge switch needs to be enabled and if the switch can be enabled.

To check if any discharge switches need to be enabled, the flowchart which can be seen in figure 64 was implemented.

The first check that happens is to examine if the value of Cell1 is higher than Cell2. If so, then the discharge switch over Cell1 can be activated. The value of SW1 is then set to one. Cell1 is the only part of the code that operates this way since it is the first cell and there is no other cell to consider other than the initial comparison.

If Cell1 was not bigger than Cell2, the code is skipped and the next comparison starts. If Cell2 is larger than Cell3 the code goes to another decision. If SW1 is zero then the switch over Cell2 can be turned on and the variable SW2 given a value of one. If SW1 had a value other than zero the discharge switch activation will be skipped.

This ensures that no two adjacent discharge switches are ever on simultaneously.

A comparison following this procedure is done to the last (highest cell allocated) cell in the stack.

7.2 Programming Code

The full code of both C and Visual Basic are available in Appendix D. The code will be discussed here. Note that the code will not compile as loops are cut and coding omitted as this section is only used to discuss sections of the code.

7.2.1 Hex Code

This is part of the hex code that was written by writing a simple program that, when written to the microcontrollers EEPROM, provides the initial values for the highest and lowest voltage. The initial value of these two settings for all cells is 1024, which corresponds to 1.536V. This can be changed by using the created program and must be loaded when the system is first used. The format that the code is written in is called Intel Hex and was developed by Intel circa 1970.

The next line of text shows what the Hex code looks like; it is useless to humans and contains information such as address, stored values and a checksum on the entire code.

```
:10000000400004000404400440000400044004D0
...
```

This code, which is 33 lines long, fills the EEPROM to about one third.

7.2.2 C Code

The code starts by giving some basic information on the author and the different versions of the code.

```
LTC6802.c
Control Program for LTC6802-1 Development Board
Nick Prinsloo
204112923
Cape Peninsula University of Technology 2009
Rev 4.00

Rev Dates and Changes Made:
1.00    18/10/2009          Created
2.00    25/10/2009          Added the second LTC IC
2.50    06/12/2009          Changed data to allow for communication with a GUI
3.00    19/12/2009          Changed watchdog timer settings and start loop
4.00    21/12/2009          Added cell discharge switches along with Min and Max voltage storage in EEPROM
*/
```

Function names are defined differently because they are easier to use and remember; only a few are shown.

```
//================= DEFINES ===========================

/* Extra variables defined */
#define BAUD19200           25                          // Register value for 19200 Baud
#define BAUD38400           12                          // Register value for 38400 Baud
#define STATUS      PORTA
#define RUNNING     PORTA1                              // Blue LED
#define STOP        PORTA0                              // Red LED
```

The register values of the LTC are defined in the same way. All the functions can be obtained from the datasheet. Creating proper names creates ease of use.

```
#define WRCFG       0x01                    // Write configuration register group
#define RDCFG       0x02                    // Read configuration register group
#define RDCV        0x04                    // Read cell voltage register group
#define RDFLG       0x06                    // Read flag register group
```

Pre written include files from WinAVR that allow functions to be used without writing them are included. These included input and output port names, interrupt setup and sleep functions. This made programming easier by only having to read what the function requires and then using the function.

```
/* Initialisations and Includes */
#include <avr\io.h>                         // Basic input and output declaration
#include <avr\interrupt.h>                  // Interrupt declarations
#include <avr\sleep.h>                      // Sleep mode declarations
```

Some more functions are defined, these were specifically written for this project. They include SPI data send and receive, EEPROM reading and writing and USART functions

```
// Send Data via SPI
void SPI_MasterTransmit(char cData)
{

        SPDR = cData;                       // Load data into transfer register

        while(!(SPSR & (1<<SPIF)))          // Wait for SPI transfer to complete
        ;

}

void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{

        while(EECR & (1<<EEWE))             // Wait for previous write to complete
        ;

        EEAR = uiAddress;                   // Set address
        EEDR = ucData;                      // Set data to be written

        EECR |= (1<<EEMWE);                 // Set master write enable bit

        EECR |= (1<<EEWE);                  // Set write enable bit and write
}

unsigned char EEPROM_read(unsigned int uiAddress)
}
```

The main program is preceded by special register setup settings that will be loaded into the LTC ICs later on. The main variables that receive sent data from the ICs are also created here. If it was ever required to be able to set settings from the PC GUI they would have to be changed here, but this is not a current requirement and they are static.

```
                                            // Set config values for CFG registers
//      BOTTOM CHIP CONFIG REGISTERS
        char CFG1R0 = 0b01100001;           // Watchdog and CDC settings
        char CFG1R1 = 0b00000000;           // Discharge switches
        char CFG1R2 = 0b00000000;           // Discharge switches and masking
        char CFG1R3 = 0b00000000;           // Masking interrupts
        char CFG1R4 = 0b01100100;           // 2.4V Undervoltage
        char CFG1R5 = 0b10010110;           // 3.6V Overvoltage
        char Dummy = 0b01010101;            // Dummy variable for SPI reads
```

```
//      CHIP1
        char ADC1, ADC2, ADC3, ADC4, ADC5, ADC6, ADC7, ADC8;
        char ADC9, ADC10, ADC11, ADC12, ADC13, ADC14;
        char ADC15, ADC16, ADC17, ADC18;
        char PECVOLT, PECTEMP, PECFLG1;
        char TMPR0, TMPR1, TMPR2, TMPR3, TMPR4;
        char FLGR0, FLGR1, FLGR2;

//      STORED CELL VALUES
        unsigned char HighHCell1, HighLCell1, LowHCell1, LowLCell1;
        unsigned char HighHCell2, HighLCell2, LowHCell2, LowLCell2;
```

The watchdog timer is normally used in case the program does not perform as programmed and resets the device. The watchdog timer is regularly reset so that the watchdog does not time out. In this program it is however used to reset the device at specific intervals. The reason for doing this is to create easier code with fewer loops and the desired end result.

```
        wdt_enable(WDTO_2S);                    // Setup watch dog timer for 2sec reset

        DDRA = 0xFF;                            // Set PORTA all outputs for status LEDs

        STATUS = (1<<RUNNING);                  // Enable blue LED
```

Data is sent via SPI protocol to the LTC IC. This data is the setup code for the LTC chips. This tells the chip what to do and how to calculate values. Some of the functions include directions of general purpose input and outputs (GPIOs), masking of the highest two cells, setting over and under voltage settings and the setting of discharge switches amongst other things.

```
// START Send Config                        // Send a command to the LTC Chips


//      CHIP HIGH
        PORTB &= ~(1<<SS);                  // Pull slave select low

        SPI_MasterTransmit(WRCFG);          // Send command address
                                            // Send config registers, they are the
                                            // different for all the LTC Chips
        SPI_MasterTransmit(CFG1R0);         // Send config register 0
        SPI_MasterTransmit(CFG1R1);         // Send config register 1
        SPI_MasterTransmit(CFG1R2);         // Send config register 2
        SPI_MasterTransmit(CFG1R3);         // Send config register 3
        SPI_MasterTransmit(CFG1R4);         // Send config register 4
        SPI_MasterTransmit(CFG1R5);         // Send config register 5
```

The highest and lowest voltage obtained by any cell is read from the EEPROM. This will be the last value that was saved to the EEPROM or the initial value, depending on how long the stack has been running and when the EEPROM code was loaded. This function simply fetches the values from the EEPROM and saves them in variables. No calculations are done on any of the values.

```
//      READ STORED CELL VALUES             // Read values out of the EEPROM
        HighLCell1 = EEPROM_read(1);        // from address 1 to 96 and store
        HighHCell1 = EEPROM_read(2);        // in the appropriate variable
        LowLCell1 = EEPROM_read(3);
        LowHCell1 = EEPROM_read(4);
```

Cell conversions are started by sending a STCVAD command to the ICs. This does not allow discharge to be permitted while doing ADC conversions. Reasons for doing this have been covered in Chapter 5. Accordingly each cells voltage is measured and saved in memory for later retrieval.

```
// START Cell Voltage AD Conversions and Poll Status                    // Start ADC conversions
        PORTB &= ~(1<<SS);                                              // Pull slave select low
        SPI_MasterTransmit(STCVAD);                                     // Send start conversion command
        _delay_ms(18);                                                  // Wait for conversions to complete
        PORTB |= (1<<SS);                                               // Pull slave select high
```

The cell's voltage data that has been quantified to a binary value is then read to the memory of the microcontroller and stored in variables. This is a 12 bit value and the variable storage space in the microcontroller is only 8 bits wide. The LTC passes data along in 8 bit sections which can later be manipulated to be 12 bits as seen in the flow charts.

```
        PORTB &= ~(1<<SS);                                              // Pull slave select low
        SPI_MasterTransmit(RDCV);                                       // Send read voltage command
//      CHIP LOW
        SPI_MasterTransmit(Dummy);                                      // Transmit dummy and receive data
        ADC1 = SPDR;
        SPI_MasterTransmit(Dummy);                                      // Same for all the registers
        ADC2 = SPDR;
```

The internal IC temperature of the LTC is obtained in exactly the same way.

```
// START Temperature AD Conversions and Poll Status
        PORTB &= ~(1<<SS);                                              // Pull slave select low
        SPI_MasterTransmit(STTMPAD);                                    // Send start conversion command
        _delay_ms(5);                                                   // Wait for conversions to complete
        PORTB |= (1<<SS);                                               // Pull slave select high
// END Temperature AD Conversions and Poll Status
// START Read Temp Registers
        PORTB &= ~(1<<SS);                                              // Pull slave select low
        SPI_MasterTransmit(RDTMP);                                      // Send read voltage command
//      CHIP LOW
        SPI_MasterTransmit(Dummy);                                      // Transmit dummy and receive data
        TMPR0 = SPDR;
        SPI_MasterTransmit(Dummy);                                      // Same for all the registers
        TMPR1 = SPDR;
        SPI_MasterTransmit(Dummy);
```

Flag registers were used in the beginning to see how the device responded to overvoltage conditions. When an overvoltage condition appears the flag is set and as soon as that condition disappears the flag is reset. This code is not used anymore as calculation of the actual voltage is used rather than the flags.

```
// START Read Flag Registers
        PORTB &= ~(1<<SS);                                              // Pull slave select low
        SPI_MasterTransmit(RDFLG);                                      // Send read voltage command
```

```
//      CHIP LOW
        SPI_MasterTransmit(Dummy);                  // Transmit dummy and receive data
        FLGR0 = SPDR;
        SPI_MasterTransmit(Dummy);                  // Same for all registers
        FLGR1 = SPDR;
        SPI_MasterTransmit(Dummy);
        FLGR2 = SPDR;
```

The next section describes the method that was used for creating one 16 bit value from two 8 bit registers. The actual voltage calculation of each cell is done by the VB program and only bit manipulation is done here.

```
// START Cell Voltage Calculations

        unsigned char Temp1, Temp2, Temp3 = 0;      // Temp bit registers for voltage
        char buf[5];

//      CHIP1
        volatile uint16_t Cell1, Cell2, Cell3, Cell4, Cell5;
        volatile uint16_t Cell6, Cell7, Cell8, Cell9, Cell10;
        volatile uint16_t Cell11, Cell12;

        volatile uint16_t TEMPEX1, TEMPEX2, TEMPINT;    // Temp bit registers for temperature

//      HIGH AND LOW VOLTAGE FROM EEPROM            // Variables for Highest and Lowest voltage
        volatile uint16_t CellHV1, CellLV1, CellHV2, CellLV2;
        volatile uint16_t CellHV3, CellLV3, CellHV4, CellLV4;

//      CHIP LOW
//      CELL1
        Temp1 = ADC1;                               // Move 8 bits to variable
        Temp2 = ADC2;                               // Move 8 bits to variable
        Temp3 = (Temp2 & 0x0F);                     // Mask the 4 MSBs
        Cell1 = ( Temp3 << 8 ) + Temp1;

//      Reconstruct High and Low values
//      High
        Temp1 = HighLCell1;                         // Move 8 bits to variable
        Temp2 = HighHCell1;                         // Move 8 bits to variable
        Temp3 = (Temp2 & 0x0F);                     // Mask the 4 MSBs
        CellHV1 = ( Temp3 << 8 ) + Temp1;
//      Low
        Temp1 = LowLCell1;                          // Move 8 bits to variable
        Temp2 = LowHCell1;                          // Move 8 bits to variable
        Temp3 = (Temp2 & 0x0F);                     // Mask the 4 MSBs
        CellLV1 = ( Temp3 << 8 ) + Temp1;

        if (Cell1>CellHV1)                          // Check if current value is higher
        {
        EEPROM_write(1, ADC1);
        EEPROM_write(2, ADC2);
        }

        if (Cell1<CellLV1)                          // Check if current value is lower
        {
        EEPROM_write(3, ADC1);
        EEPROM_write(4, ADC2);
        }
```

Passive discharging is included and the algorithm for selection of discharge switch follows. Two adjacent switches can never be allowed to be on at the same time. Enabling of the discharge switches is permitted if the stack is being charged or discharged. This is done to check the initial effectiveness of the passive balancing and the effect it has on the LTC ICS with respect to heat over time.

```
//      DISCHARGE SWITCHES                                  // Variables for discharge switch state
        Int SW1, SW2, SW3, SW4, SW5, SW6, SW7;
        int SW8, SW9, SW10, SW11, SW12;
        SW1 = SW2 = SW3 = SW4 = SW5 =SW6 = SW7 = 0;
        SW8 = SW9 = SW10 = SW11 = SW12 = 0;

//      BOTTOM LTC
        if(Cell1>Cell2)                                     // Compare against previous cell
        {
        CFG1R1 |= 1;                                        // Turn switch ON
        SW1 = 1;
        }

        if(Cell2>Cell3)                                     // Compare against previous cell
        {
                if((SW1==0))                                // If the previous switch is off
                {
                CFG1R1 |= 2;                                // Turn switch ON
                SW2 = 1;
                }
        }
```

If any switches were enabled these new configurations must be sent to the registers of the LTC. This is done in exactly the same way as it was done previously in this code.

```
// START Send New Config                                    // Send discharge switches to LTC Chip

//      CHIP HIGH
        PORTB &= ~(1<<SS);                                  // Pull slave select low

        SPI_MasterTransmit(WRCFG);                          // Send command address
                                                            // Send config registers, they are the
                                                            // different for all the LTC Chips
        SPI_MasterTransmit(CFG1R0);                         // Send config register 0
        SPI_MasterTransmit(CFG1R1);                         // Send config register 1
```

Calculations of the temperature values in the next section are based on exactly the same principle as was discussed for voltage previously. The bits are rearranged, but no voltage calculations are done.

```
// START Temperature Calculations

//      CHIP LOW
//      External Temp 1
        Temp1 = TMPR0;                                      // Move 8 bits to variable
        Temp2 = TMPR1;                                      // Move 8 bits to variable
        Temp3 = (Temp2 & 0x0F);                             // Mask the 4 MSBs
        TEMPEX1 = ( Temp3 << 8 ) + Temp1;
```

USART communication to the PC follows in the next section of the code. It should be noted that data must be sent out in a certain order so that the VB GUI can interpret the data correctly. A protocol was created for this and is implemented in the GUI as will be seen in the VB code in.

```
        USART_Enable();                                     // Enable transmit and receive

//      CHIP LOW
//      START CONDITION
        USART_TransmitString("B");
```

```
//      CELL VOLTAGE START
        num = Cell1;                                    // Give num the value of Cell1
        if (num<0x3E8)                                  // if num < 1000
        {
        USART_TransmitString("0");                      // Add a 0 and transmit string
        USART_TransmitString(itoa(num, buf, 10));
        }
         else
        {
        USART_TransmitString(itoa(num, buf, 10));       // Otherwise just transmit string
        }

//      OVER AND UNDERVOLTAGE

        num = CFG1R4;
        USART_TransmitString(itoa(num, buf, 10));

        num = CFG1R5;
        USART_TransmitString(itoa(num, buf, 10));

//      END CONDITION
        USART_TransmitString("E");
        USART_TransmitString("*");
        USART_TransmitString("*");

        USART_Disable();
```

When all the data is sent, the USART port is disabled and the device is put into power down mode, waiting for a reset from the watchdog. This ensures minimum power consumption by the voltage measurement device from the stack.

```
        PORTB |= (1<<MOSI)|(1<<SCK)|(1<<SS);            // All pins high

        STATUS = (1<<STOP);                             // Enable red LED to show end condition

        set_sleep_mode(SLEEP_MODE_PWR_DOWN);            // Select power down mode
        sleep_mode();                                   // Activate sleep mode and wait for
                                                        // reset from watch dog timer
```

7.2.3 Visual Basic Code

To read values into an Excel sheet via the serial communications port an extra file was needed. The file is called MSCOMM32.OCX and it has to be installed and registered in the windows registry as well as in Excel. Installation procedures and instructions can be found in the reference (Yes Telecom, 1996).

After doing that, the GUI was created using VB. All the components were placed and named. The following code declares the operation of the different boxes, buttons and functions.

The first section states that when data, of unknown length, is received (comEvReceived command) it should be known by the variable Data.

```
Private Sub MSComm1_OnComm()                                        'START OF PROGRAM
If Worksheets("Display").MSComm1.CommEvent = comEvReceive Then      'If data received then
    Worksheets("Display").MSComm1.InputLen = 0                      'Data of unknown length
    Data = Worksheets("Display").MSComm1.Input                      'Assign the input to DATA
```

Since data is sent out in a serial string, the data is preceded by a B (for Begin) and ended with an E (for End). Doing this ensures that a full string of data is captured (complies with the protocol) before any calculations are done on the data. The Mid$(X, Y, Z) command is explained just following this section.

```
 If Mid$(Data, 1, 1) = "B" Then          'check Header               'Check start condition

    If Mid$(Data, 112, 1) = "E" Then     'check Tailer               'Check end condition
```

(Data, 2, 4) specifies that the second value, for 4 consecutive values, in the Data variable should be used as the value for TexBoxCell1. The value from that calculation should be multiplied by 0.0015 (1.5mV) to convert the decimal value of the ADC to the actual cell value in volts. This conversion factor was obtained from the datasheet. This value should be rounded to three decimal places and this is accomplished by the Round(X, 3) command. This is done for all 24 cells, although only two is shown.

```
        TextBoxCell1 = Round((Mid$(Data, 2, 4) * 0.0015), 3)        'Give TextBoxCell1 the value of Cell 1
        TextBoxCell2 = Round((Mid$(Data, 6, 4) * 0.0015), 3)        'Goes on till all 24 Cells are displayed
```

The temp calculation is done in exactly the same way as the preceding code. The answer is multiplied by 1.5 and divided by 8. This gives the temperature in Kelvin, so that if 273.15 is subtracted from that the answer will be in degrees Celsius. This calculation holds for both temperature readings.

```
        TextBoxTemp = ((Mid$(Data, 98, 4) * 1.5 / 8) - 273.15)      'Give TextBoxTemp the value Temp1
```

Again the code is the same as the preceding code, but with a different mathematical algorithm.

```
TextBoxUVL = ((Mid$(Data, 106, 3) * 0.0015 * 16))                    'Give TextBoxUVL the UV level

TextBoxOVL = ((Mid$(Data, 109, 3) * 0.0015 * 16))                    'Give TextBoxOVL the OV level
```

The following code is a basic mathematical calculation of the stack voltage by adding all the individual cells together.

```
TextBoxStack = ((Mid$(Data, 2, 4) * 0.0015) + (Mid$(Data, 6, 4) * 0.0015) + (Mid$(Data, 10, 4) * 0.0015) + (Mid$(Data, 14, 4) * 0.0015) + (Mid$(Data, 18, 4) * 0.0015) + (Mid$(Data, 22, 4) * 0.0015) + (Mid$(Data, 26, 4) * 0.0015) + (Mid$(Data, 30, 4) * 0.0015) + (Mid$(Data, 34, 4) * 0.0015) + (Mid$(Data, 38, 4) * 0.0015) + (Mid$(Data, 42, 4) * 0.0015) + (Mid$(Data, 46, 4) * 0.0015))

                                                                     'Adds the first 12 Cells values

TextBoxStack = TextBoxStack + ((Mid$(Data, 50, 4) * 0.0015) + (Mid$(Data, 54, 4) * 0.0015) + (Mid$(Data, 58, 4) * 0.0015) + (Mid$(Data, 62, 4) * 0.0015) + (Mid$(Data, 66, 4) * 0.0015) + (Mid$(Data, 70, 4) * 0.0015) + (Mid$(Data, 74, 4) * 0.0015) + (Mid$(Data, 78, 4) * 0.0015) + (Mid$(Data, 82, 4) * 0.0015) + (Mid$(Data, 86, 4) * 0.0015) + (Mid$(Data, 90, 4) * 0.0015) + (Mid$(Data, 94, 4) * 0.0015))

                                                                     'Adds the second 12 Cells values tot the
                                                                     'first and displays it in TextBoxStack
```

If the LOG DATA toggle button is pressed the values of the individual cells will be logged onto an Excel sheet for later use and for graphical display. “a” is incremented by one every time this code is executed so that the row moves down one, while the column remains the same. Just two cell values are show here.

```
If ToggleButton1 = True Then                                         'If LOG DATA button is pushed

                                                                     'then do the following

  a = a + 1                                                          'Increment A by one every time

  Sheet2.Cells(a, 1) = TextBoxCell1.Value                            'Display Cell 1's value in column 1

  Sheet2.Cells(a, 2) = TextBoxCell2.Value                            'Do for all 24 Cells
```

The following code will check if the Com port is open or closed when the START MEASUREMENT button is pressed and display a message accordingly.

```
Private Sub CommandButtonOpenPort_Click()                            'If START MEASUREMENT button is
                                                                     'pushed

If Worksheets("Display").MSComm1.PortOpen = True Then                'Check Comm Port status

  TextBoxError = "Com1 Already Opened"                               'If open, then display

Else

TextBoxError = "Com1 Open"                                           'Else, display


 If Worksheets("Display").MSComm1.PortOpen = False Then              'If not open then do following
```

If the Com port was not open, the following code will use the following as setting and open the Com port.

```
  Worksheets("Display").MSComm1.Settings = "38400,N,8,1"             'Set Com. Port settings

  Worksheets("Display").MSComm1.RThreshold = 112                     'Set threshold

  Worksheets("Display").MSComm1.InBufferSize = 4096                  'Set max buffer size

  Worksheets("Display").MSComm1.PortOpen = True                      'Open Com Port
```

The next section of the code is used to close the Com port as well as display the status.

```
Private Sub CommandButtonClosePort_Click()                    'If STOP MEASUREMENT button is
                                                              'pushed
  a = 1                                                       'Equate A to 1
  If Worksheets("Display").MSComm1.PortOpen = False Then      'Check Comm Port status
    TextBoxError = "Com1 Already Closed"                      'If closed, then display
  Else
    TextBoxError = "Com1 Close"                               'Else display
    Worksheets("Display").MSComm1.PortOpen = False            'Close Comm Port
```

If the CLEAR GRAPH DATA is pressed the values on Sheet2 are cleared and logging and graphs begin from the beginning.

```
Private Sub CommandButtonCLRDATA_Click()                      'If CLEAR button is pushed


  MSComm1.PortOpen = False                                    'Close Comm Port
  Sheet2.Cells = Clear                                        'Clear all cells on Sheet2
  a = 1                                                       'Equate A to 1
  MSComm1.PortOpen = True                                     'Open Comm Port
```

8 Chapter Eight: Results

Due to the nature of the IC, simulations cannot be done and only actual results will be displayed here. For testing the initial values and settings a communication program called Serial Communicator was used. This allowed programming of the LTC while viewing the data in legible form.

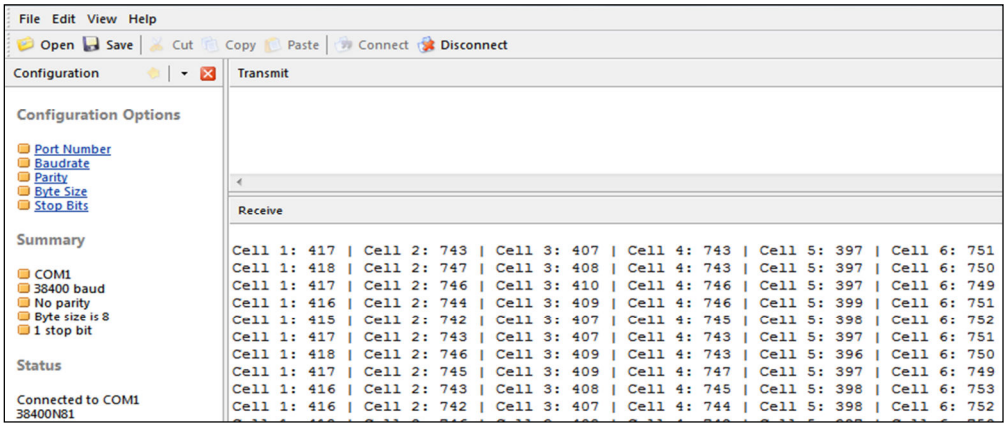

Figure 65: Serial Communicator showing values of cell 1 through 12 (with 6 cell values visible).

Figure 65 shows the data that was produced while doing initial tests on the IC. This specific figure showed that the serial communication settings were setup correctly and that the IC did communicate with the microcontroller. The program proved to be a useful tool for checking initial data and at later stages, where data that was sent to the GUI.

8.1 Cell Measurement Results with Cell Simulator

8.1.1 Testing Criteria

The testing criteria were based on the system specifications. If all the parameters were met or exceeded then acceptable results were obtained. Three different cells in a stack of 12 cells were tested to check measured accuracy against actual accuracy.

8.1.2 Measurement and supply devices

For voltage measurements the Agilent 34401A digital multimeter with two wires was used. Voltage accuracies of ±0.004% (0.0035% reading + 0.0005% range) could thus be obtained. The multimeter has an input impedance of 10MΩ when measuring DC values.

The power supply used was the DF1731SB 3A. In constant voltage mode, regulation was specified as 100mV + 500uV. All actual readings were rounded to the nearest 1.5mV.

8.1.3 Testing Procedure

The first test was to determine how accurate the LTC was over the whole operating voltage range. A full sweep of the voltage range was done by setting the voltage, measured using the Agilent

multimeter and finally comparing the measured values to the output values of the LTC. It was known that each increment in the ADC value represented 1.5mV. This was done for cells 1, 5 and 11 at 200mV increments. Passive equalising of the LTC was disabled for all tests to ensure high accuracy.

8.1.4 Test Results for Cell 1

The graph in figure 66 shows the actual results from the LTC minus the actual voltage measurement.

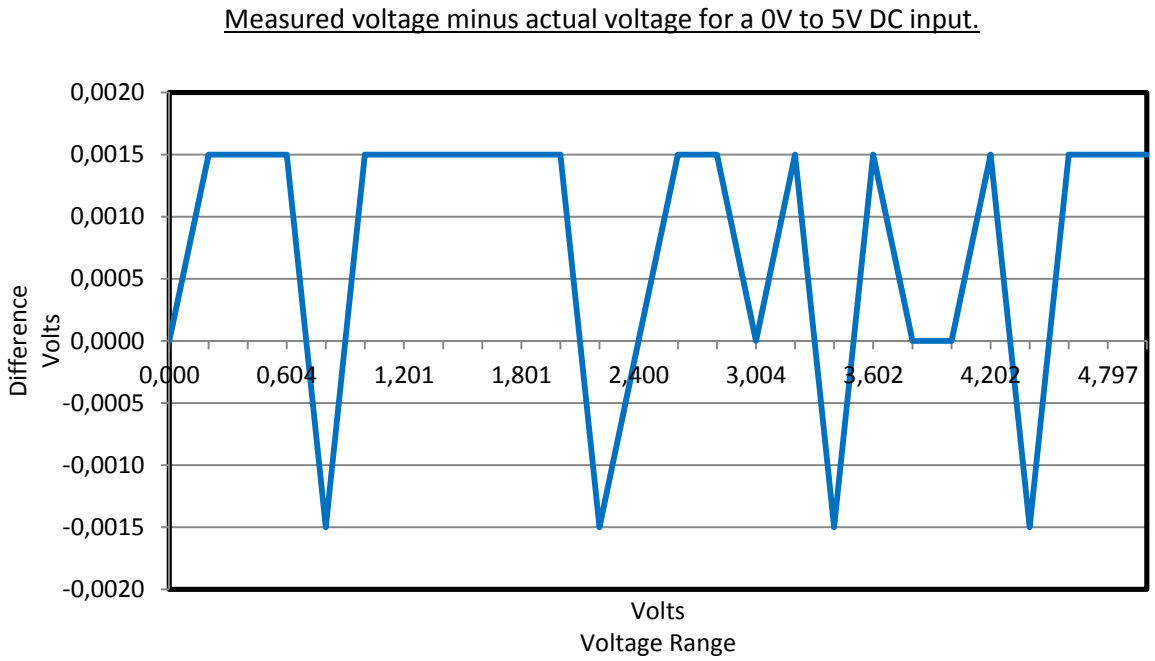

Figure 66: Difference in measured voltage minus actual voltage for cell 1.

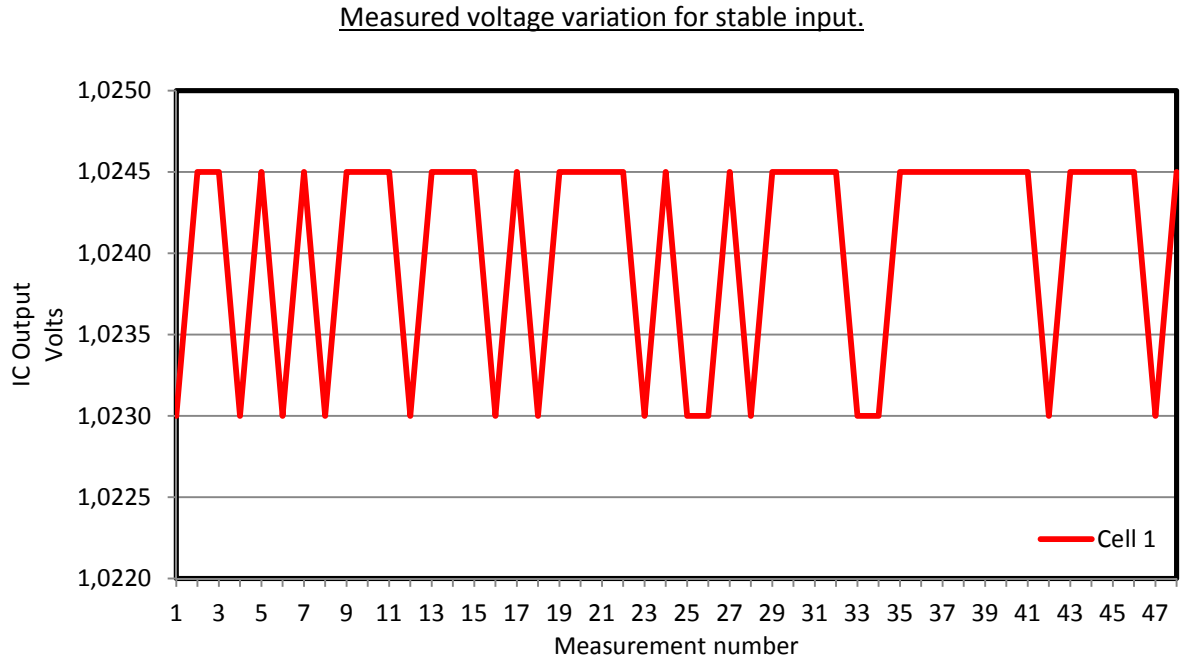

Figure 67: Measured voltage variation with stable input voltage for cell 1.

The results (figure 66 and 67) show that there is a maximum difference of 1.5mV up or down from the actual voltage. This is well below the system specification and is considered accurate.

8.1.5 Test Results for Cell 5

Measured voltage minus actual voltage for a 0V to 5V DC input.

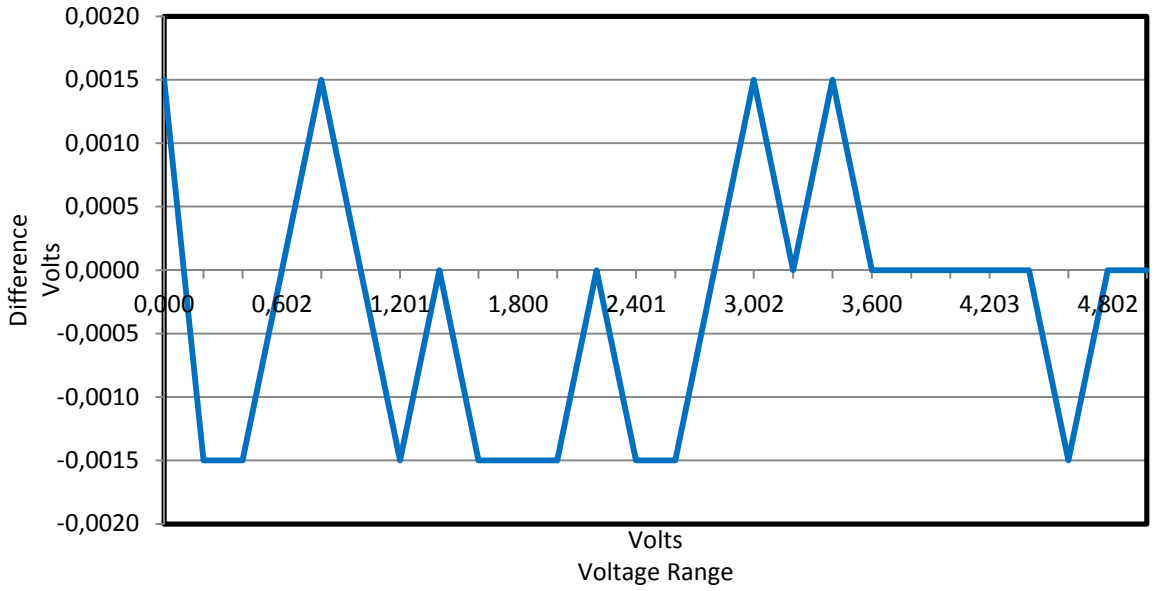

Figure 68: Difference in measured voltage minus actual voltage for cell 5.

Measured voltage variation for stable input.

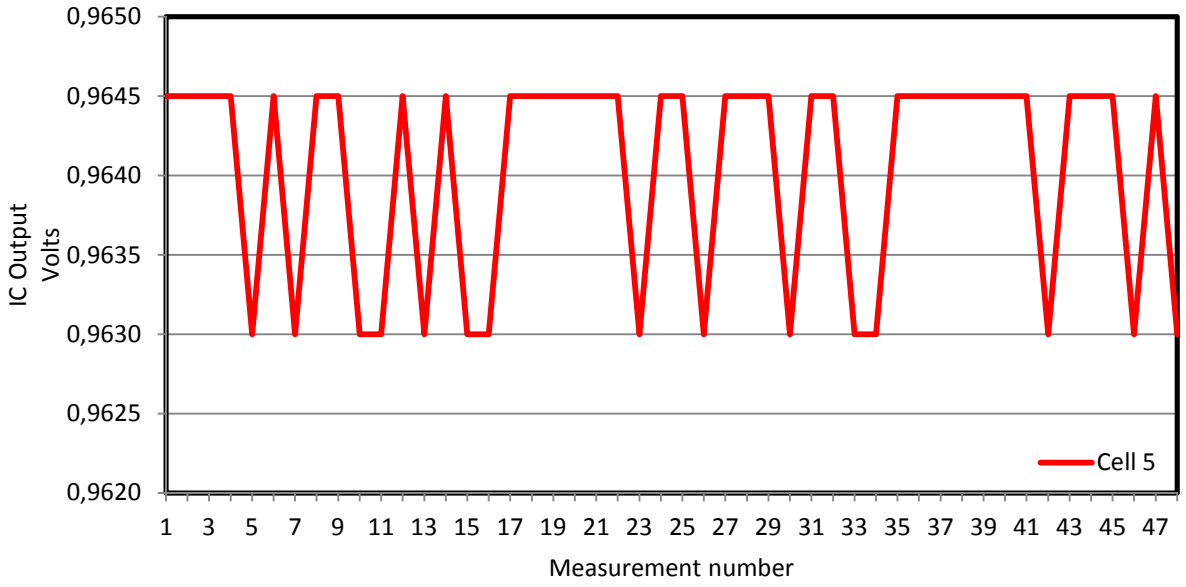

Figure 69: Measured voltage variation with stable input voltage for cell 5.

The results (figure 68 and 69) are similar to the measurements of cell 1 in that a maximum difference of 1.5mV up or down from the actual voltage is measured.

8.1.6 Test Results for Cell 11

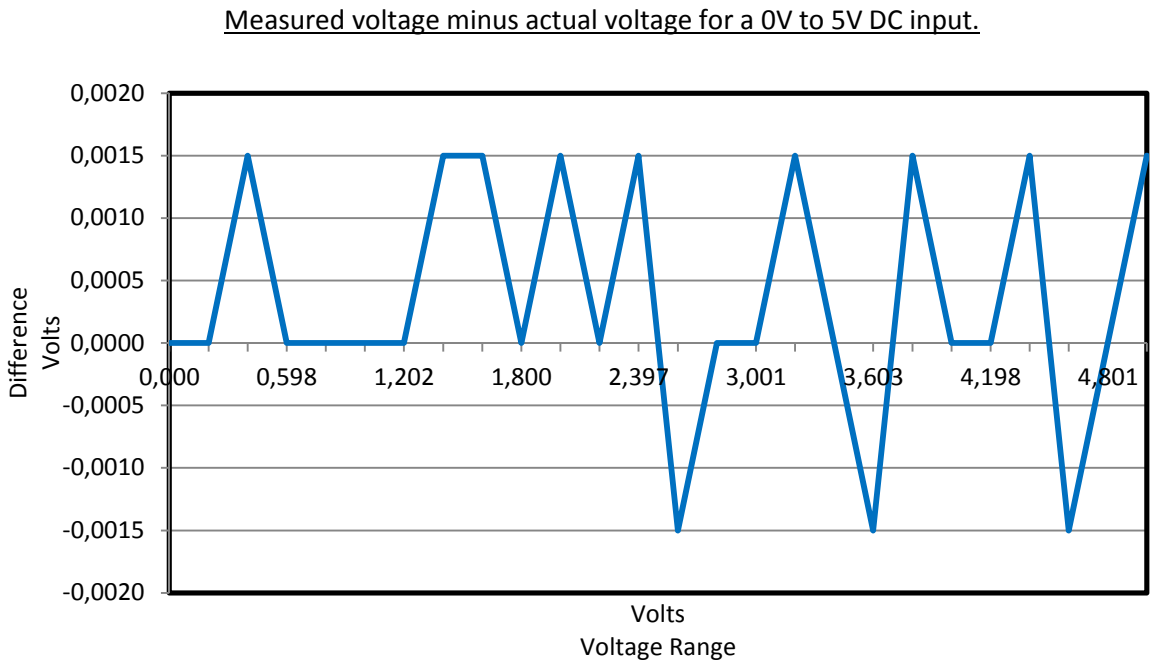

Figure 70: Difference in measured voltage minus actual voltage for cell 11.

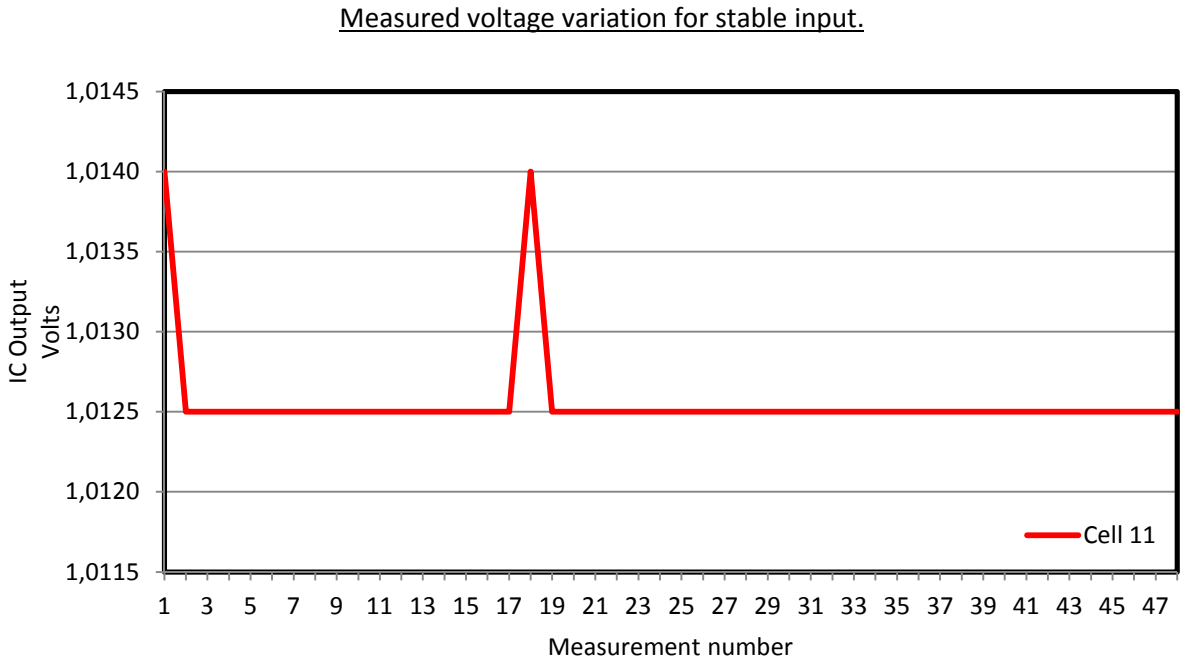

Figure 71: Measured voltage variation with stable input voltage for cell 11.

8.1.7 Conclusion

Excellent 0V to 5V results, figure 66, 68 and 70, were obtained using the LTC IC. Equally impressive were the stable results, figure 67, 69 and 71. Measurements were taken right after the filter, so that what was measured with the IC was what was measured with the voltmeter.

The IC specifies that a maximum input voltage of 5V per cell should not be exceeded, but while measuring it was noticed that a 5V measurement corresponds to a decimal value of 3334 for the LTCs 12 bit ADC. Out of a maximum possible value of 4095, this is only 81.42% of the full scale measurement. The maximum that was reached while testing was 3390, which corresponds to a voltage of 5.085V and was measured as such.

This shows that the device should be able to theoretically measure 6.144V, although this was not tested. This safety margin is possibly for overvoltage conditions on the stack that might occur as no cell chemistry has a voltage this high.

Having the ADC measure from 0V to 5V showed that the LTC is able to measure any cell chemistry as all the possible chemistry voltages fall into this range. It also showed that the accuracy over the 0V to 5V range was linear.

It was noted that while the input remained stable, the maximum difference in measurement was 1.5mV and only in one direction from actual measurement. Where the input was ramped from 0V to 5V the value changed to 1.5mV above and below the actual value. This was attributed to the design of the cell simulator and the power supply.

8.1.8 Test Results for all 12 Cells

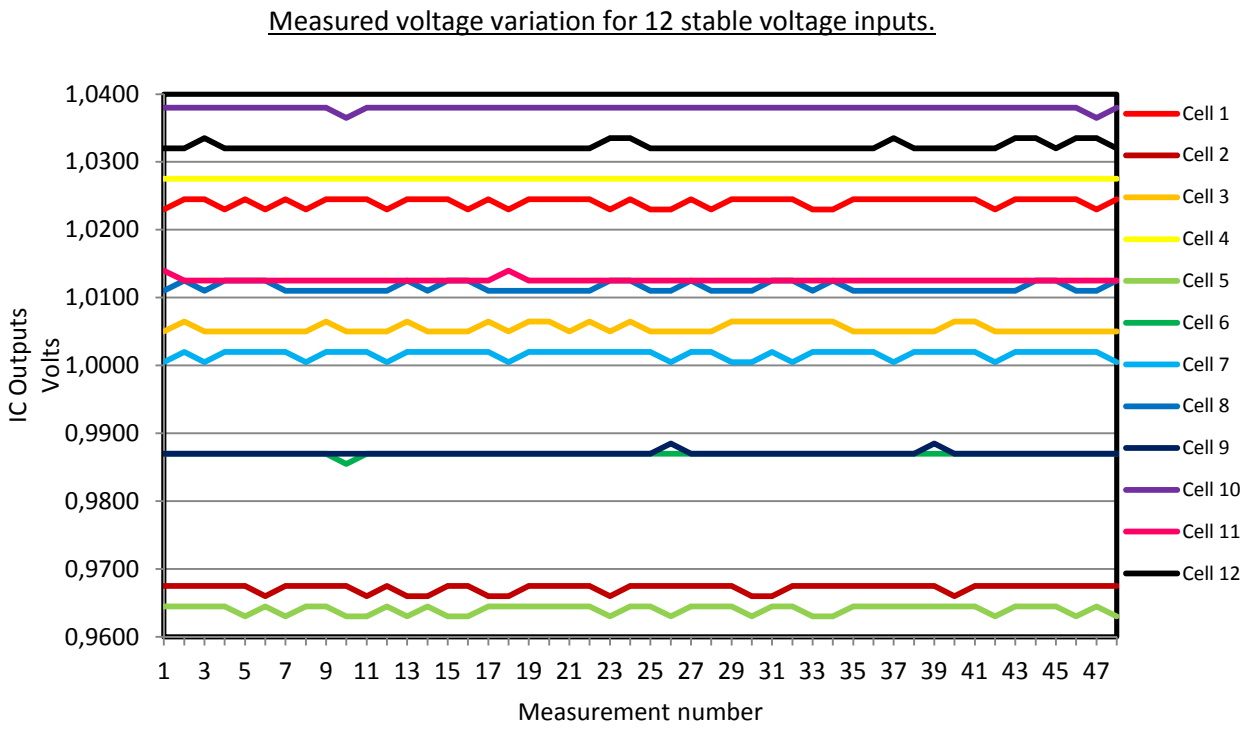

Figure 72: 48 Measurements of arbitrary voltages created with the cell simulator and logged with the bottom most IC.

Figure 72 shows 12 individual cell voltages on the same graph. They were generated with the cell simulator and logged by the GUI. The voltages were set arbitrarily and not according to a specific chemistry. This showed that even though variation occurs, it is well within system specification.

8.2 Cell Measurement Results with 24 Cell Lead Acid Stack

8.2.1 Test Results for Cell 1

Since excellent results were obtained with the cell simulator a 24 cell lead acid stack was obtained and tests were done in exactly the same way as they were for the cell simulator to see if accuracy would be similar. Only measurements for a single channel were captured since the accuracy for different channels was proved in the previous test.

The graph in figure 73 shows the variation from the stable voltage of the cell. The actual voltage was measured as 2.013V. The stack was in a discharged state when the first tests were done and is evident in the voltage measured by on channel 1 of the LTC IC.

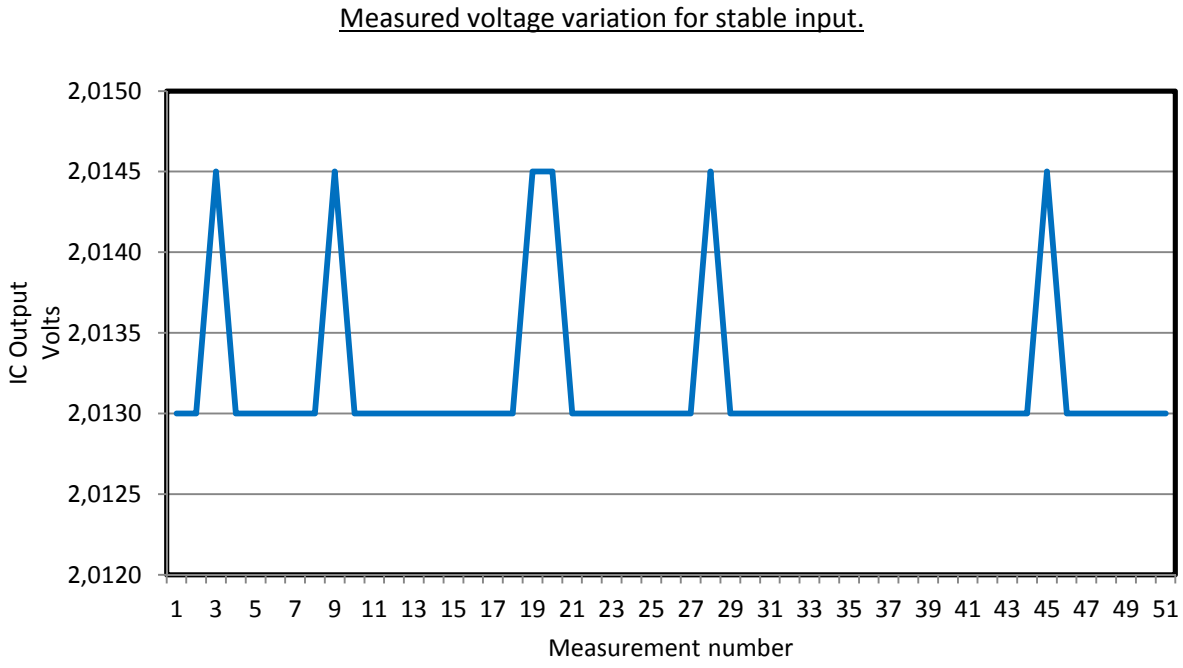

Figure 73: Measured voltage variation with stable input voltage for cell 1.

Results are similar to the results obtained for the cell simulator. Small voltage fluctuations exist in the measurement. Since the net change is never more than 1.5mV the results can be considered accurate.

This also proves that the cell simulator design operated like a Lead Acid battery in steady state.

8.3 Cell Measurement Results while Charging

8.3.1 Test Results for Cell 1

The next set of tests was used to determine the effects charging the cells along with the passive equalising would have on the cell measurements. For this test the prototype board was supplied from an external supply and not the cells. The charger that was used was a constant voltage switch mode power supply.

The graph in figure 74 shows the actual results as was captured from the cell 1 input of the LTC. The results are not as stable as previous measurements, but still within system specifications. Since results were obtained from the GUI, the results were rounded to the nearest millivolt. Note that the graph is drawn differently because of the different answers to the rounding of 1.5mV and 3mV and multiples of them.

The difference between the uncharged cell (figure 73) and the charged cell (figure 74) can clearly be seen. Figure 76 will later display the final voltage for cell 1 as the floating voltage is maintained.

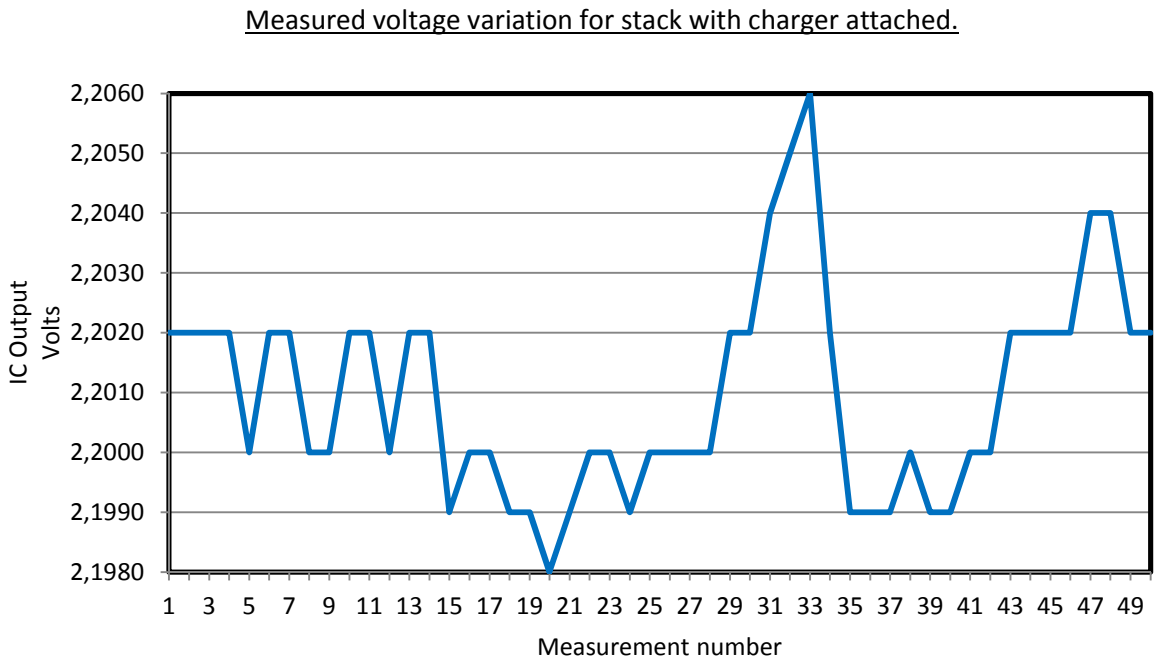

Figure 74: Measured voltage variation with stable input voltage for cell 1.

Voltage variations are from the discharge switches that try to balance cells. The inherent ripple of a switch mode power supply could not cause the variation because of the delta sigma's inherent filtering as well as the RC filter built onto the prototype board.

Since this is still within system specification, it was not deemed a problem even though it should be noted when using.

8.4 Heat Test

8.4.1 Heat Tests on Bottom LTC IC

Heat tests were done in exactly the same way as the DC stable voltage tests were done. An oven with thermal control was used to slowly increase the heat to 50°C above ambient.

This was done because the devices, under normal operation were just above 31°C and 34°C respectively. Tests were done with the cell simulator as it was easier and safer to fit into the oven than the lead acid cell stack.

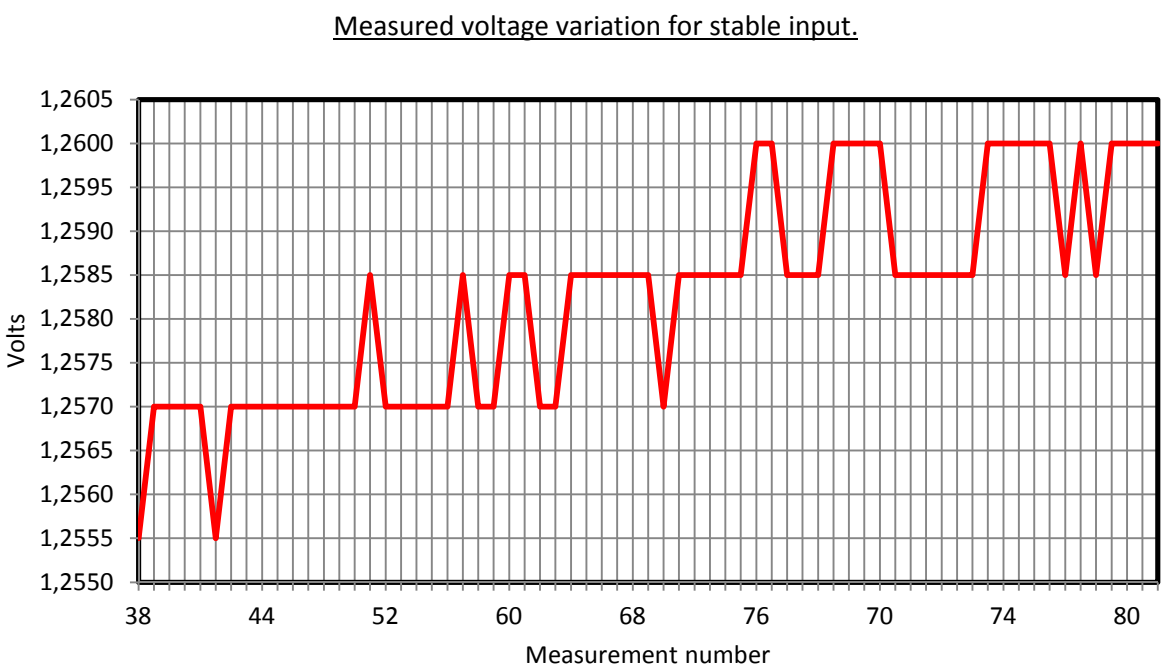

Figure 75: Voltage of cell 5 with an increase in chip temperature from 37°C to 80°C.

The LCT IC heat results are recorded onto the graph (figure 75). It must be noted that the supply used was considered accurate and stable enough and as such the voltage that goes to the LTC was not measured, as in the previous experiment, with an external voltmeter. The results are that of the ADC and the internal temperature ADC of the LTC. This was not deemed a problem as any rise in temperature would have this effect on the measurements.

From the graph above it can be seen that the higher the temperature went the higher the measured value went. The initial value was 1.255V and changed to a maximum of 1.260V. This is still within the system specification of <10mV accuracy and that is after a total temperature change of more than 40°C. Linearity for temperatures above 85°C is not guaranteed.

Only temperature rise was taken into consideration as temperature decrease below 0°C is extremely unlikely.

9 Chapter Nine: Graphical User Interface

Figure 76 shows all twenty four cell voltages, among other things, displayed on the GUI. The GUI was designed with a simple view in mind, while taking most of the calculation work away from the microcontroller. The microcontroller sends only the necessary data to the GUI and the data is calculated by the superior processor of the PC.

9.1 GUI

The GUI displays the twenty four cell voltages along with the stack voltage that it calculates from the twenty four measurements. Start and stop buttons are included to either start or stop data acquisition. The GUI is updated instantaneously and the update rate is controlled by how fast the microcontroller sends data to the PC.

If the log data toggle button is in the on position, the cell voltages will be logged for later retrieval. A maximum of 32000 points can be logged at any one instance in Excel. Over and under voltage levels are obtained from the microcontroller and can only be changed there.

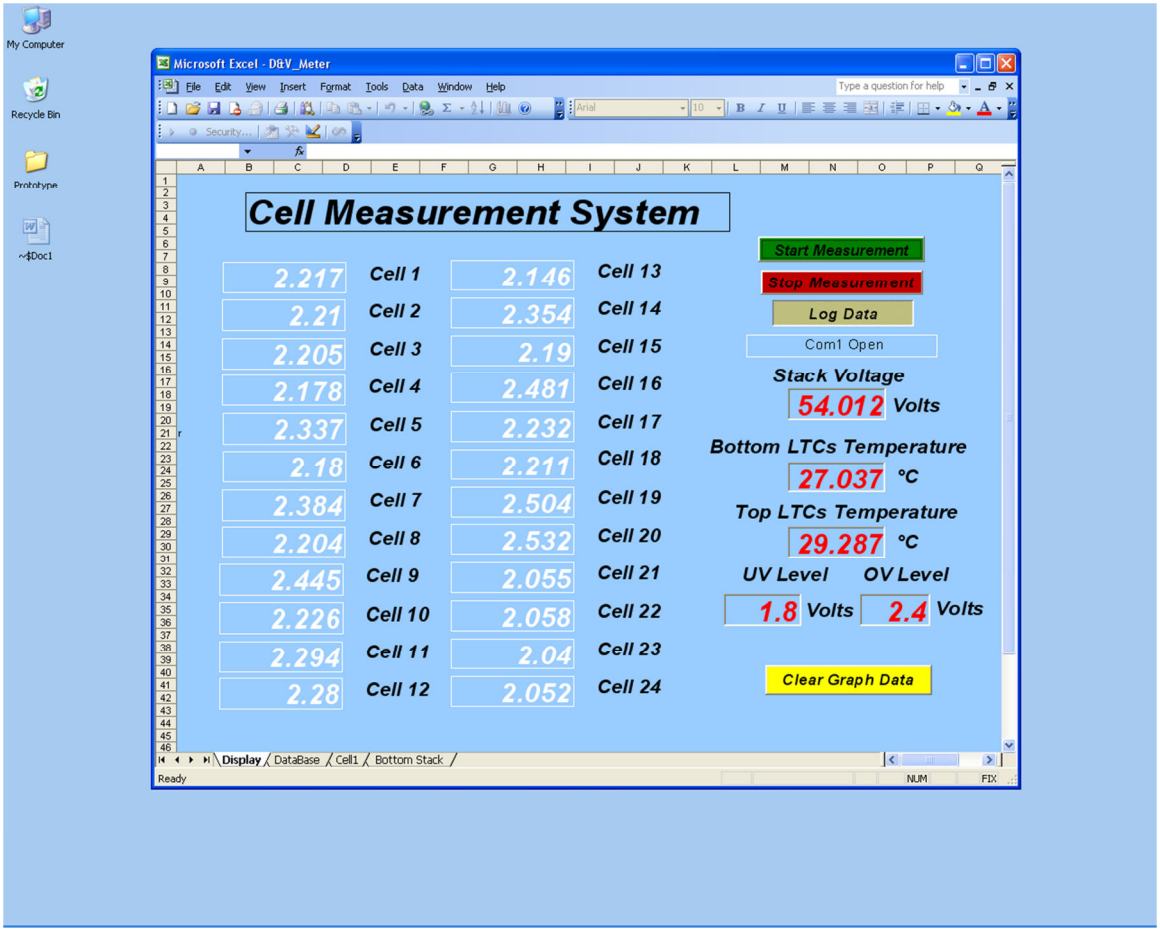

Figure 76: Screenshot of the GUI created to view cell voltages, chip parameters and additional information.

The clear graph data button clears the stored data and resets the graphs. This would be used if some error occurred or if new measurements are required.

The stack voltage is displayed along with the temperature of the two LTC ICs. All the information that is on screen can be recorded into a file, although only the voltage is currently being recorded.

9.2 GUI Results and Explanations

Figure 77 shows the actual problem that this research is meant to detect. This occurs when a stack is charged. The graph clearly shows why it is necessary to measure the individual voltages so that passive or active balancing can be done.

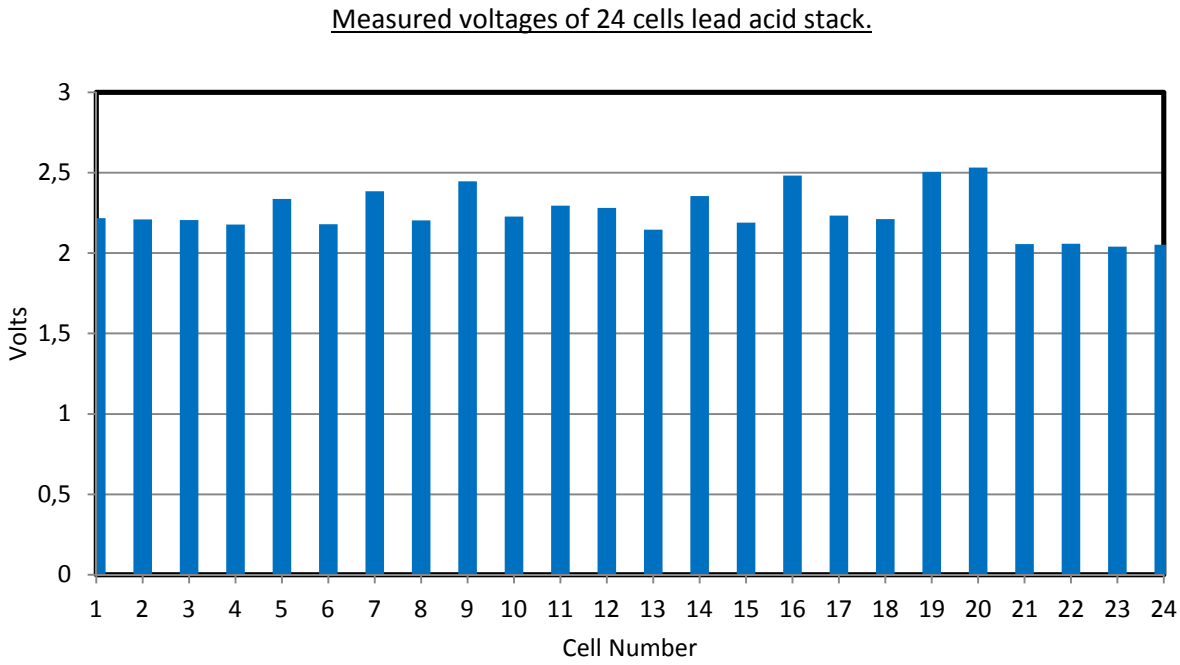

Figure 77: Actual voltage of 24 individual lead acid cells.

The required voltage level for a lead acid cell at 100% SOC is considered to be 2.105V and the ideal charging voltage is considered to be 2.25V. Out of the 24 lead acid cells, only 13 are in a safe voltage range where they won't over or under charge. This can be seen in figure 78 where viewing accuracy is increased by showing voltages from 2V to 2.6V. The other 11 cells are in a position where their plates will either sulphate over time because of insufficient charging voltage or they will suffer chemical loss from excessive gassing and heat generated by being subjected to higher voltages.

High accuracy measured voltages of 24 cells lead acid stack.

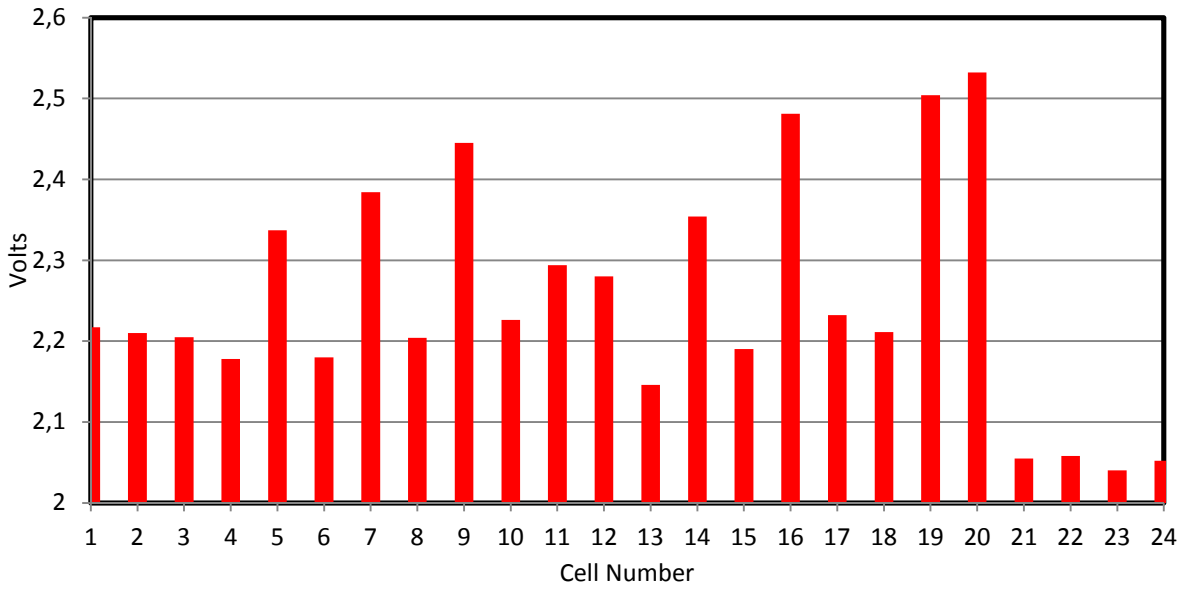

Figure 78: 24 Lead acid cells voltages displayed between 2V and 2.6V.

The charger was set to deliver 2.25V per cell, which equates to 54V. This can be seen in figure 76. However, because of the inherent cell imbalance and differences their voltages are not equal and hence vary. This variation will increase with time and even at the current stage damages to cells will occur if charging is permitted.

Variations between measured voltages of 24 cells lead acid stack and ideal.

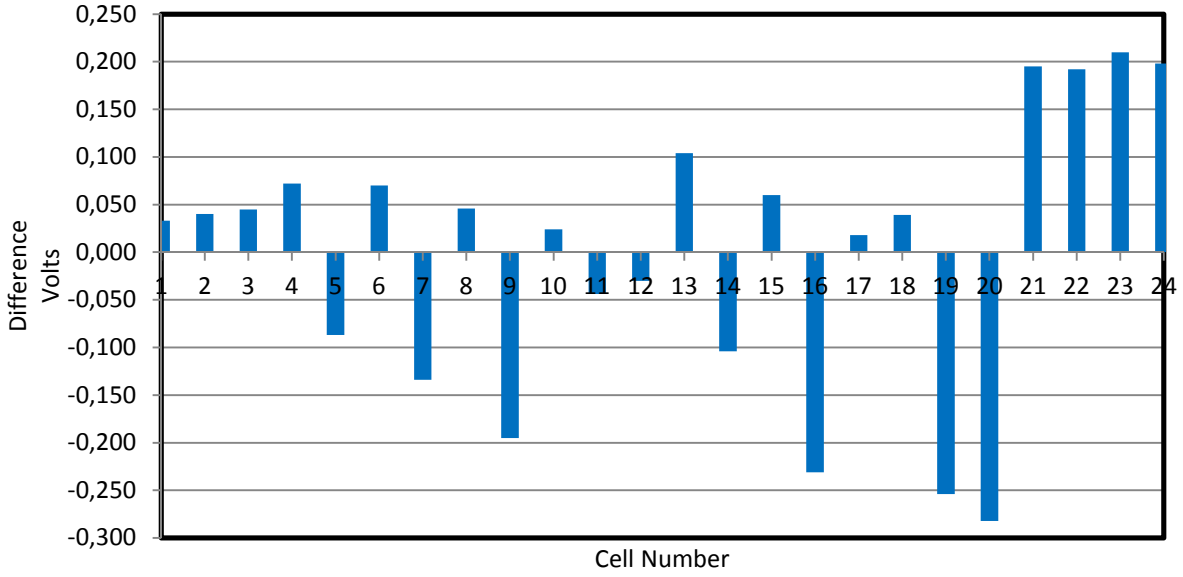

Figure 79: Voltage variation from ideal of 2.25V for a 24 cell stack.

Figure 79 show the deviation of each cell from the ideal charging cell voltage of 2.25V. The biggest difference for this stack is at cell 20 which is 282mV above the ideal. This equates to a voltage of 2.532V. If charging is permitted at this level the cell will fail prematurely as it has an elevated temperature and a loss of internal chemicals.

At the low end cell 23 has a voltage of 210mV less than 2.25V. This equates to a voltage of 2.04V as can be seen on the GUI. At these low voltage levels the plates of the cells will sulphate, making them unusable over time.

Having this information available there are three things that can be done:

1. When the stack is charged from a state where all the cells are below the ideal charging level of 2.25V, the stack should only be charged until any cell reaches 2.25V and then the charging should be stopped. This will undercharge most cells causing sulfation of the plates and future premature cell failure. When the cells are being discharged and the lowest cells voltage reaches 1.8V the stack should be disconnected from the load. Since most of the cells were undercharged from the previous charge cycle this would occur much sooner than if all the cells were equal. Hence stack operating time is shortened.
2. Knowing that cell voltages are not equal they can be replaced with cells with similar voltage levels at the same SOC. This is expensive, difficult and time consuming and will never work in reality.
3. Balancing can be incorporated to balance cells during the charging phase (active balancing could also be used during the discharge phase). Both passive and active balancing will ensure longer stack operation and cell lifetime at the cost of consuming some power.

9.3 Conclusion

The GUI gives information on the 24 lead acid cells on an accessible interface. The individual cell information tells the users useful information on the stack. Even though alarms could be given through the GUI, hardware should have resolved any unfavourable condition before unnecessary damage is done to the equipment, cell stack or user.

Using the system will allow the successful controlled operation of an active cell balancer and the following benefits will thus be obtained:

1. Longer stack operation.
2. Longer cell lifetime.

10 Chapter Ten: Conclusion

Table 10: Actual specification for a 12 cell measurement system, taken for worst case scenarios.

Topology	Accuracy	Operating Temp	Additional Components	System Power Consumption
ISO120	0.01%	-55°C to 125°C	One per cell, isolated power supply	60mA
4N25	Unknown	-55°C to 100°C	One per cell, isolated power supply	180mA
IL300	0.01%	-55°C to 100°C	One per cell, isolated power supply	120mA
Isolation Transformer	Unknown	-40°C to 125°C, based on components	One per cell, isolated power supply	Unknown
DC to DC Converter	0.001%	-40°C to 125°C, based on components	One per cell, isolated power supply	13mA
LTC6802-1	0.25% over full range	-40°C to 85°C	None	8mA
MAX11068	0.25% over full range	-40°C to 105°C	None	10mA
ATA6870	0.2% over full range	-40°C to 85°C	Two ICs	10mA

Table 10 and 11 shows an overview of required specifications against actual specifications. From these tables it can be seen that results were well below required specification. Only current consumption was higher and could be minimized if a lower powered microcontroller and a RS-232 IC with power down capability was used.

Table 11: Table showing desired specification against actual specifications of the complete system.

Parameter	Required Specification	Actual Active Mode	Actual Sleep Mode
Current Consumption	< 1mA	8mA	3mA
Voltage Measurement Range	0V to 5V	0V to 5V	N/A
Overvoltage Range	7V	6.144V	N/A
Voltage Measurement Accuracy	< 0.25%	0.12% worst case	N/A
Main Isolation Barrier, if Isolated	2kV to 5kV	Limited to IC	Limited to IC
All Cell Measurement Time	< 100msec	13ms	N/A
Measurement Accuracy	< 10mV	<6mV Worst Case	N/A

10.1 Discussion of Problems Encountered

10.1.1 Isolation amplifier

Although excellent results can be obtained using isolation amplifiers, the large amount of isolation amplifiers necessary to produce a system that will fit the specifications makes it uneconomical. The size and complexity in connecting them further ads to the unsuitability of isolation amplifiers.

10.1.2 Optocoupler

Large current draw and unacceptable performance makes optocouplers unsuitable as possible accurate voltage measurement devices. Possible performance increase can be obtained from creating a complex circuit, which would further increase current draw, cost and board space.

10.1.3 DC to DC Converter

Excellent performance was obtained using the DC to DC converter. Size and cost did however not allow it to be used as an accurate voltage measurement device in a system. Decreased ratings of chosen components could have minimized size and cost, but not to the extent that would allow it to become feasible. If size could be minimized the same integration problems would be faced as was the case for the isolation amplifiers.

10.1.4 Transformer Isolation

This simulation circuit performed well, but device differences make it impossible to ensure accuracy. The transformer isolation design would also not allow a small PCB layout. Again the same problem facing integration of isolation amplifiers and the DC to DC converter would be faced here.

10.1.5 Battery management ICs

These ICs solved the problems facing the other topologies with regard to space, power consumption, integration and accuracy. They allow accurate measurements in a very small PCB space with the ability of stacking multiple ICs on top of one another.

10.1.6 LTC6802-1

The LTC was an overall good battery management ICs and the only one that was built onto PCB and tested. No problems were encountered and accurate measurements were taken.

The top IC in the stack was about 3°C warmer than the bottom one. This is due to the operation of the ICs when communicating with each other. In communication mode between ICs, signals are sent by current rather than voltage. When the lower device wants to send a logic high, the lower device sinks a smaller current. To send a logic low, the lower device sinks a higher current. The same is true for the top device. To send a logic high the top device sources a higher current to the bottom device

and to send a logic low the higher device sources a smaller current to the lower device (Linear Technology Corporation, 2009).

Because of the programming algorithm, the lower device passes the configuration data to the top device; thereafter only 0s are sent while retrieving information. This leads to the top device having to supply current for longer periods which heats up the IC. The difference in temperature is not as significant as to cause problems.

10.1.7 System Design

Since the individual components that make up the system were well known by experimentation with the development board no system problems were encountered.

10.1.7.1 Current Consumption

The actual current consumption was much higher than initially anticipated. This is higher than the required specification and the main cause for this was the MAX232 IC. Since the IC did not have a low power mode, it was always in the active mode and had constant power drain. ICs that have a shutdown pin do exist and using it will allow a 4mA reduction in current.

The second contributing factor to the higher current consumption was the microcontroller. This was only when the microcontroller was in active mode and was for a very short time as was expected.

10.1.7.2 PCB Design

The prototype PCB is bigger than what is needed for the components. The layout of such a large PCB was done to allow easy access to all the measurement points. Since this was the prototype board it was known that many measurements would be taken and compared to results obtained through IC measurements.

Overall the PCB performed well as did the cell simulator. At no time during any of the test did anything on the board blow, even though the system was subjected to a wide variety of voltages and heating.

10.1.7.3 C Code

The programmable code for the LTC was time consuming and better preparation could have resulted in a shorter development time. The flowcharts section was done after struggling with the initial programming and not using flowcharts. Use of flowcharts proved to be necessary.

Problems that were encountered were the following:

1. Voltage comparison algorithm.
2. Discharge switch algorithm.
3. Initial values for the EEPROM.

Solutions to all these problems were found and implemented.

The final summarised description of the code is as follows.

Configuration data is sent to the LTC ICs. ADC measurements of all the cell voltages are taken and sent back to the microcontroller along with ADC temperature measurements. The new values of the cells voltage is compared against old values that were stored in the microcontrollers EEPROM. If the values are either higher or lower, the new values are written back to the EEPROM. The cells are compared against one another and if any difference is encountered the appropriate discharge switches are activated. The system is put into sleep mode for two seconds while discharging of overcharged cells is permitted. The system is reset and the process begins again.

This led to a system that worked well and that was stable over time.

10.1.7.4 Visual Basic

This was overall an easy program to learn and to code in. Major problems were encountered in the beginning with the MSCOMM file. This was due to the operating system and the version of Office.

Microsoft Windows 7 Ultimate and Microsoft Office 2007 were used for the initial system setup. After initial problems with the Windows 7 OS and Office it was tried on Microsoft XP Professional and Office 2003. When the file installation procedures were followed it worked immediately.

Exact reasons for the MSCOMM not working in Microsoft Windows 7 Ultimate and Office 2007 are unknown.

The final GUI worked well, but as data began to build up the system began to slow down. At about 10,000 points the system began to suffer. The data acquisition system that was used was a Pentium 4, 2.8GHz HT with 2GB of memory.

The Excel environment tended to be unstable as results began to build up. For initial testing the GUI was good, but a more stable program needs to be created if larger cell stacks need to be measured.

10.2 Future Design

10.2.1 Microcontroller

The microcontroller used was initially thought to have enough program memory. This was however not the case. Some functions could not be included because, the code as is, took up exactly 80% of the program memory. As these devices are pin to pin compatible a microcontroller with a larger program memory could have been used, but was not available at the time test were done. Additional features that could have been included are described in section 10.2.3.

The microcontroller was a good controller to test the initial code for the LTC IC and its features.

10.2.2 Microcontroller or FPGA

Recent FPGAs proclaim that a system on programmable chip (SOPC) can deliver the same and even better performance compared to that of a microcontroller. If a FPGA could be low cost, low power and programmable it could be a replacement for the controller.

This would have to be investigated and tested before any such statements can be considered true.

Features that manufacturers claim (Altera in general) are that you only program what you need. A microcontroller has many peripherals that are never used, thus only programming what is needed leaves the rest of the pins open for extra functionality. Having a SOPC also has the advantage of having a FPGA in the system. FPGAs are known for their calculation capability and speed. With their claimed low cost and low power consumption it seems like the perfect future solution.

10.2.3 Programmable Components

10.2.3.1 Cycle Redundancy Check Algorithm

The LTC offers an eight bit cycle redundancy check (CRC) code that is transmitted at the end of each serial transfer. This can be used to compare against a calculated CRC from the received data. If these two values, the sent CRC and the calculated CRC, match, the data was sent correctly.

Implementation of this code would allow for reliable data transfer, especially in noisy environments, or where many LTC ICs are stacked.

10.2.3.2 Charge and Discharge Algorithm

By having an algorithm that counts the discharge cycles, as well as how deep the discharge cycles are, it would be possible to calculate the remaining lifetime of the battery.

10.2.3.3 Passive Balancing

Passive balancing should only be enabled when the stack is charging to ensure that unnecessary discharge does not occur.

10.2.4 Isolation

The system, as was designed, featured no isolation other than that given by the ICs. A simple way of creating isolation is to put an isolation module between the communication device and the PC. This will allow worst case destruction of the measurement board with no damage being done to the PC or operator.

10.2.5 Wireless Communication

This is another form of isolation. A wireless communication link would isolate the measurement device from the display device.

This has however another benefit in that the system becomes portable. The system is not bound to a stationary point and seeing as that it provides backup to another system it will be able to transmit as long as the stack is operational.

Many different types of wireless communication can be used namely:

1. Bluetooth
2. Wi-Fi
3. Zigbee

The choice on which to choose will be application specific as the range of these is very different and they require different operating costs.

Furthermore these devices do not have to connect directly. If the system can be connected to the internet the results can be viewed from any place in the world.

10.2.6 Integration

The previous section described that the device does not necessarily have to be connected to a PC. Many other forms of devices we carry around every day will be able to display this data. A cellular phone, iPod and a Blackberry can be used and the applications seem endless. All these can connect to the internet as well as Wi-Fi and Bluetooth. This broadens the scope of integration and application drastically.

10.2.7 Power Measurement

By reading Daniel O' Connells thesis (O' Connell, 2009) it was obvious that to further increase the usefulness of accurate voltage readings would be to include intelligent current metering. With accurate voltage and current readings certain power parameters could be displayed. These extra parameters could only be obtained if accurate readings were obtained in a specific time. Special attention was given in his thesis to this accuracy and what factors reduce it. These included sampling rate and thermal drift.

He created a system that could measure two channels at 500Hz with 16-bit delta sigma ADCs. By accurately measuring current, voltage and time and performing math algorithms on the results the following could be displayed:

1. Power.
2. Charge (*Ah*).
3. Energy (*Wh/kWh).*
4. Energy-cost.
5. Voltage-peak.
6. Current-peak.
7. Power peak.

By following his principals and incorporating it into the BMS will open up a much wider variety of information available to the user about the stack's operation as well as the load.

10.3 Conclusion and Closing Remarks

Conclusions in respect of the following:

1. Objectives of Research.
2. Delineation of Research.

10.3.1 Objectives of Research

10.3.1.1 Identify a suitable method to measure individual cell voltages in a string of twenty four cells.

Isolation amplifiers, Optocouplers, transformer isolation, DC-to-DC converters and Battery Monitoring ICs were simulated, tested and evaluated. All the methods worked adequately except the optocouplers. They were either not economically viable or connecting them in large systems were complicated.

The BMS ICs provided economical, compact and easy system integration and was therefore chosen to build a system with.

10.3.1.2 Build a prototype and do initial tests to check the accuracy of the chosen design.

A prototype was built and tests were conducted. The first tests were communication tests to see if the LTC ICs responded to commands given and with the PC. This was checked with a program called Serial Communicator. Tests were successful and different settings could be programmed to the controller through the LTCs onboard SPI port.

A cell simulator was built and cell voltages were simulated. Once successful simulations were done a GUI was created along with a protocol to accept information from the microcontroller. Once communications to the GUI was correctly implemented further modifications were done to the code on the controller to include functionality.

10.3.1.3 Initial tests will be done on a twenty four cell stack, but should be scalable to any cell count.

After multiple tests on the bottom LTC IC, the second board was connected by rearranging two resistors that was designed for this purpose. This allowed the addition of another LTC IC and increased the measureable cell count to 24. Increased scaling is possible by including component arrangements in the design and can be scaled up to a stack voltage of about 1000V.

10.3.1.4 Integrate to a PC and log data through a graphical user interface (GUI).

After both LTC ICs were connected, the GUI was operational and the extra functionality coding was included. The system could log the values of 24 cells as well as display other stack parameters. The cell simulator was replaced at this point with a lead acid stack of 24 cells and tests were done.

10.3.1.5 Investigate wireless capabilities to increase portability of the system.

This was originally included for investigation and it was later decided to exclude it from the main research. Wireless capabilities should be definite future addition to any modern design and it was discussed in the future research.

10.3.2 Delineation of the Research

10.3.2.1 Measure, display and log the individual voltages of a twenty four series cell stack.

An accurate voltage monitoring system was designed. The system could log accurate voltage measurements to a PC.

The prototype system was proof of right component choice (with regard to the LTC6802-1) and principal. It showed that compact integration could be done between a series cell stack and a PC.

11 References

Alessandrello, A, Brofferio, C, Bucci, C, Camin, D, Cremonesi, O, Giulianai, A, Nucciotti, A, Pavan, MPG & Previtali, E 1998, 'A linear, low noise, low power optocoupler amplifier for bolometric detectors', *Nuclear Instruments and Methods in Physics Research*, vol 409, no. 1, pp. 343-345.

Andrea, D 2009, *Comparison of Integrated Circuits for Battery Management Systems for Li-Ion batteries*, viewed 29 May 2009, <http://liionbms.com/php/wp_bms_chips.php>.

Aponte, MA, Andujar, M & Schroder, E 1996, 'Demand for higher performance and functional properties of batteries', in JC Salamone (ed.), *Polymeric materials encyclopedia, volume 1*, 1st edn, CRC Press, Inc, Florida.

Atmel 2002, *ATmega16, 8-bit microcontroller with 16K in system programable flash*, viewed 25 August 2009, <www.atmel.com/atmel/acrobat/doc2466.pdf>.

Atmel 2009, *Li-Ion, NiMH battery measuring, charge balancing and power supply circuit*, California, viewed 9 June 2009, <www.atmel.com/dyn/resources/prod_documents/doc9019.pdf>.

Band, A & Unguris, J 1997, 'Optically isolated current to voltage converter for an electron optics system', Published Paper. National Institute of Standards and Technology. Maryland.

Bergveld, JH 2002, 'Battery management systems, design by modelling', PhD Thesis, Department of Electrical Engineering, University of Twente, ISBN 90-74445-51-9, Kluwer Academic Publishers, Netherlands.

Burr Brown 1992, *Precision low cost isolation amplifier ISO120*, Arizona, viewed August 16 2009, <http://focus.ti.com/general/docs/lit/getliterature.tsp?genericPartNumber=iso121&fileType=pdf>.

Camina, D, Grassia, V, Levatia, F & Scherini, V 2004, 'Galvanically isolated and linear transmission of analog current-signals using the optically coupled current-mirror architecture', *Nuclear instruments & methods in physics research*, vol 535, no. 1, pp. 485-490.

Chen, Y & Evans, JW 1995, 'Thermal management of Lithium/Polymer-electrolyte batteries for electric vehicle application', *The Electrochemical Society*, The Electrochemical Society, Inc, New Jersey.

De Vries, I 2008, 'Design considerations of active battery cell balancers', Unpublished Paper, Department of Electrical Engineering, Cape Peninsula University of Technology, Cape Town.

Del Rio, A & Cao y Paz, A 2002, 'AC power line measurement using linear optocoupler', *EDN*, 1 October 2002, pp. 76-78.

Endo, M & Kim, Y 2003, 'Applications of Advanced Carbon Materials to the Lithium Ion Secondary Battery', in M Endo, A Oya, Y Tanabe (eds.), *Carbon Alloys: Novel Convepts to Develop Carbon Science and Technology*, 1st edn, Elsevier Science Ltd, Oxford.

Ibanez, J & Dixon, JW 2004, 'Monitoring battery system for electric vehicle based on "one wire" technology', Unpublished Paper, Department of Electrical Engineering, Pontificia Universidad Católica, Chile.

Kernighan, B & Ritchie, D 1988, *The C Programming Language*, 2nd edn, Prentice Hall, New Jersey.

Kultgen, L 2009, 'Managing high-voltage lithium-ion batteries in HEVs', *EDN*, 4 September 2009, pp. 45-52.

LEM 2010, *Sentinel - comprehensive monitoring for critical battery systems*, viewed 08 May 2011, <http://www.lem.com/images/stories/files/Products/P1_7_5_sentinel/Sentinel_D27100_E_280208.pdf>.

Linear Technology Corporation 2009, *Multicell battery stack monitor*, California, viewed 9 June 2009, <http://cds.linear.com/docs/Datasheet/6801p.pdf>.

Mathivanan, N 2007, 'Data Aquisition Using Serial Interfaces', in *PC-Based Instrumentation: Concepts and Practice*, 1st edn, Asoke K. Ghosh, New Delhi.

Maxim 2008, *12 Channel, high voltage sensor with smart data acquisition interface*, Maxim, viewed 9 June 2009, <http://www.maxim-ic.com/quick_view2.cfm/qv_pk/5523>.

Meyrath, TP 2005, 'Precision analog optocoupler', Unpublished Paper, Center for Nonlinear Dynamics, University of Texas, Austin.

Microchip 2003, *PIC16F877A enhanced flash microcontrollers*, viewed August 25 2009, <http://ww1.microchip.com/downloads/en/devicedoc/39582b.pdf>.

Microchip Technology Incorporated 2007, *Portable Power Management Seminar*, viewed 23 August 2009, <http://www.microchip.com.tw/Workshop%20ZIP%20files/Portable_Power.pdf>.

Morrison, R 1997, 'Analog Circuits', in *Grounding and Shielding, Circuits and Inteference*, 5th edn, John Wiley and Sons, Inc, New Jersey.

Mulder, G, Coenen, P, De Ridder, F & Martens, A, 'An advanced cell voltage monitoring device for fuel cell control', Belgium, Published Paper. Vlaamse Instelling voor Technologisch Onderzoek. Belgium.

Network Service Group LLC 2010, *Tortoise battery monitoring system*, viewed 9 March 2011, <http://www.networkservicesgrp.com/tortoise.pdf>.

O' Connell, D 2009, 'Digital energy metering for electrical energy management', A thesis in fulfilment of the requirements for the Magister Technologiae degree in Electrical Engineering, Department of Electrical Engineering, Cape Peninsual University of Technology, Cape Town.

O'Loughlin, M 2009, 'A way to reduce costs in automotive power supplies', *Power Systems Design*, Desember 2009, pp. 36-39.

On-Line Monitoring Inc. 2010, *Battery monitoring systems*, viewed 9 March 2011, <http://www.on-lineinc.com/battery.pdf>.

Perez, R 1993, 'Lead Acid battery state of charge versus voltage', *Home Power #36*, August/September 1993, pp. 66-69.

Philips 2003, *I2C manual*, viewed 19 July 2009, <www.nxp.com/documents/application_note/AN10216.pdf>.

Power Shield 2010, *Permanent monitoring for lead-acid and Ni-Cad batteries.*, viewed 9 March 2011, <http://www.powershield.com/upload/downloads/PowerShield%20Sentinel%20Datasheet.pdf>.

Ratnakumar, BV & Smart, MC 2007, 'Rechargeable Batteries', in M Broussely, G Pistoia (eds.), *Industrial Applications of Batteries: From Cars to Aerospace and Energy Storage*, 1st edn, Elsevier B.V., Netherland.

Raynus, R & Freidline, E 2006, 'Microcontroller simplifies battery-state-of-charge measurement', *EDN*, 4 April 2006, pp. 96-98.

Stevens, J & Corey, G 1998, 'A study of lead acid battery efficiency near top of charge and the impact on PV system design', Unpublished Paper, Department of Battery Analysis and Evaluation, Sandia National Laboratories, New Mexico.

Texas Instruments 2009, *MSP430F2254 mixed signal microcontroller*, Dallas, viewed 25 August 2009, <http://www.ti.com/lit/gpn/msp430f2254>.

Van Overschee, P 1998, 'Galvanic Isolation', in JFM van Impe, PA Vanrolleghem, D Inserentant (eds.), *Advanced Instrumentation, Data Interpretation, and Control of Biotechnological Processes*, Kluwer Academic Publishers, Norwell.

Van Putten, AFP 1996, 'A Centralized Data-Acquisition System', in *Electronic Measurement Systems: Theory and Practise*, 1st edn, Taylor & Francis Group, LLC, Great Britain.

Wagener, J 2009, 'Investigation and Design of a magnetically isolated DC-DC converter for use in cell balancing', A thesis in fulfilment of the requirements for the Magister Technologiae degree in Electrical Engineering, Department of Electrical Engineering, Cape Peninsula University of Technology, Cape Town.

Williams, T 2005, 'Batteries', in *The circuit designer's companion*, 2nd edn, Newnes, Great Brittain.

Williams, M & Thoren, J 2008, 'Novel measurement circuit eases battery-stack-cell design', *EDN*, 1 October 2008, pp. 53-63.

Wright, R 2006, 'Experemental analysis of a battery thermal management system: implications and applications for industrial and commercial projects', *Journal of Industrial Technology*, vol 22, no. 2, pp. 1-6.

Yes Telecom 1996, *Caller ID, MSCOMM32.OCX MSComm Control*, viewed 14 November 2009, <http://www.yes-tele.com/mscomm.html>.

12 Appendices

12.1 Appendix A: BMS Chip Comparisons

Table 12. Table of comparison between the LTC, MAX and ATA BMS ICs (Andrea, 2009).

		Linear Technology	Maxim	Atmel
General	**Part Number**	LTC6802-1 / LTC6802-2	MAX11068	ATA6870
	Best application	Any size pack, 24 V to 800 V	Any size pack, 8 V to 1.4 kV	Medium pack, 10 V to 355 V
	Development effort required	Medium	Medium	Medium
	Control sophistication	Not included: user must develop	Not included: user must develop	Not included: user must develop
	Availability	Good	Poor	Poor
Cells	**Topology**	Modular: 1 board every 12 cells	Modular: 1 board every 12 cells	Modular: 1 board every 6 cells
	Series cells, min	4	4	3
	Series cells, max / bank (No isolators)	LTC6802-1: 192 LTC6802-2: 12	372	96
	Series cells, max total (With isolators)	Unlimited	Unlimited	Unlimited
Balancing	**On chip, passive**	Yes (barely OK)	Requires external resistor	-
	External, passive	Yes, with separate drive pins	Yes, with shared pins (1)	Yes, with separate drive pins
	External, active	-	-	-
Current Drain	**Standby drain [uA]**	50	1	10
	Operating drain [mA]	1.5	2	15
	Balance drain, @ 3.6 V [mA]	internal: 330 external possible	~250 mA (ext res)	external

Readings	**Voltage measurement accuracy @ 3.6 V, 25 C [mV]**	8	15	7
	Cell voltage, min	0	0	0
	Cell voltage, max	5	5	5
	Temperature measurement	2 every 12 cells	2 for 12 cells	2 every 6 cells
	Readings rate	13 ms	-	8 ms
	Data rate	up to 1 MHz	10 kHz ~ 1 MHz	up to 250 kHz
Communi cations	**Between adjacent boards: wires**	3, current sources (LTC6801-1) / logic levels (LTC6801-2)	4, capacitively coupled	8, current sources
	Between banks and controller: wires	3, non-isolated	4, non-isolated	8, non-isolated
	Internal	SPI	I2C (SMB)	SPI
	Out of controller	N.A.	N.A.	N.A.
Reliability	**EMI and noise immunity**	Very poor (6801-1) / Medium (6801-2)	Unknown	Unknown
	Reversed polarity connection	Destroys chip	Unknown	Unknown
	Open connection, B+	Destroys chip	Destroys balancing MOSFET	Unknown
	Open power connection between cells, same bank	Destroys chip	Unknown	Unknown
	Open power connection between cells, different banks	No damage	No damage	Unknown
	Open connection, others	Loss of data on that cell	Loss of data on that cell	Loss of data on that cell
	Misconnection of comm wires between adjacent sections	If more negative, destroys chip. If more positive, no damage	-	-

Cost	**Controller, per system [$, in 100s]**	~100	~100	~100
	ICs only / per cell [$, in 10000s]	~1	Unknown	~3
	Parts only / per cell [$, in 10000s]	~1.5	~1.5	~1.5
Notes		6802-1: for direct, daisy-chain data bus connection between ICs / 6802-2: through isolators	See note (2), below. Add reliability with a MAX11080 Overvoltage/Undervoltage 12-Cell Monitor (not yet available)	The ATA6871 is similar, but for simple go/no-go BMSs (no voltage measurements)

12.2 Appendix B: Development Board

Figure 80 and 81 show the actual PCB design that was used for all the development tests. It included 8 LEDs, 8 ADC potentiometers, RS-232 communications, ISP port, LCD display and 3 push buttons. Refer to section Chapter 4 for the schematic.

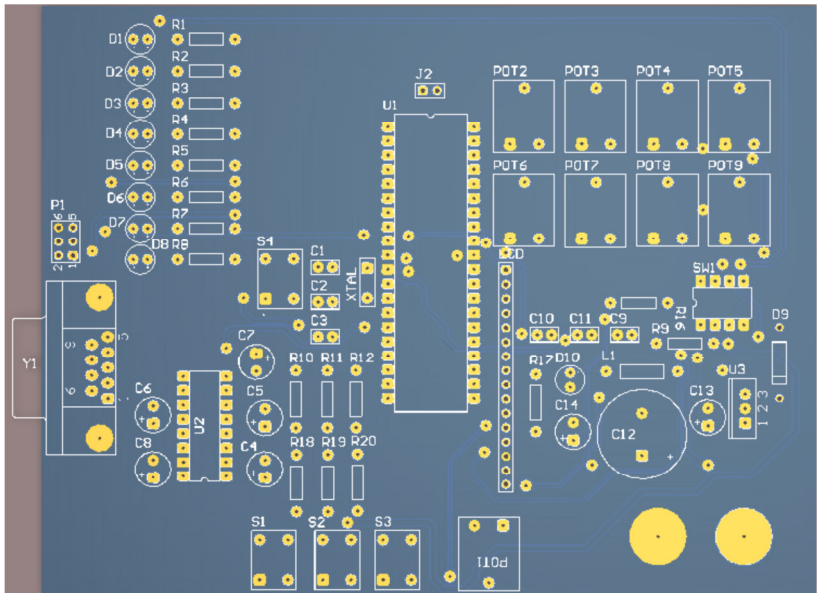

Figure 80: 2D PCB of development board topside.

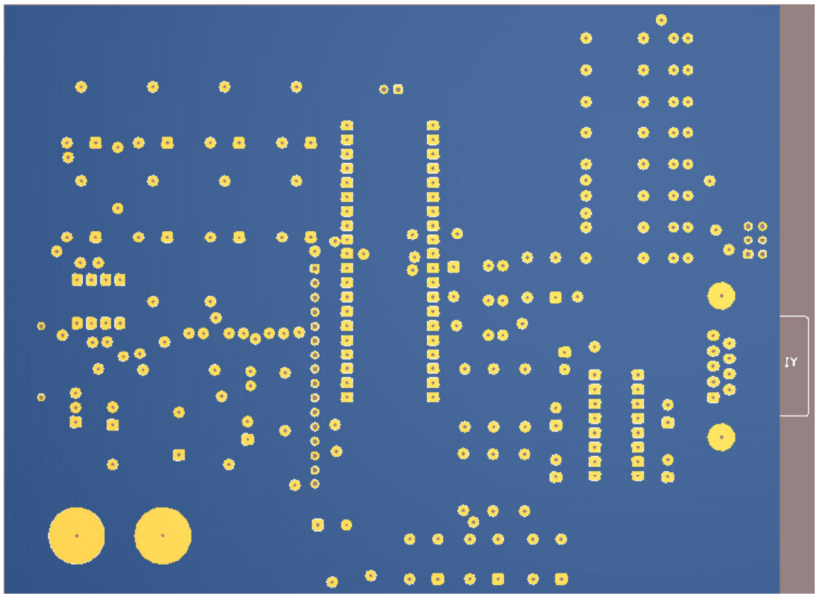

Figure 81: 2D PCB of development board bottom.

12.3 Appendix C: Prototype Board

Figure 82 and 83 show the top and bottom of the development board. The layout was chosen to be spacious for the prototype to allow easy access to measuring points.

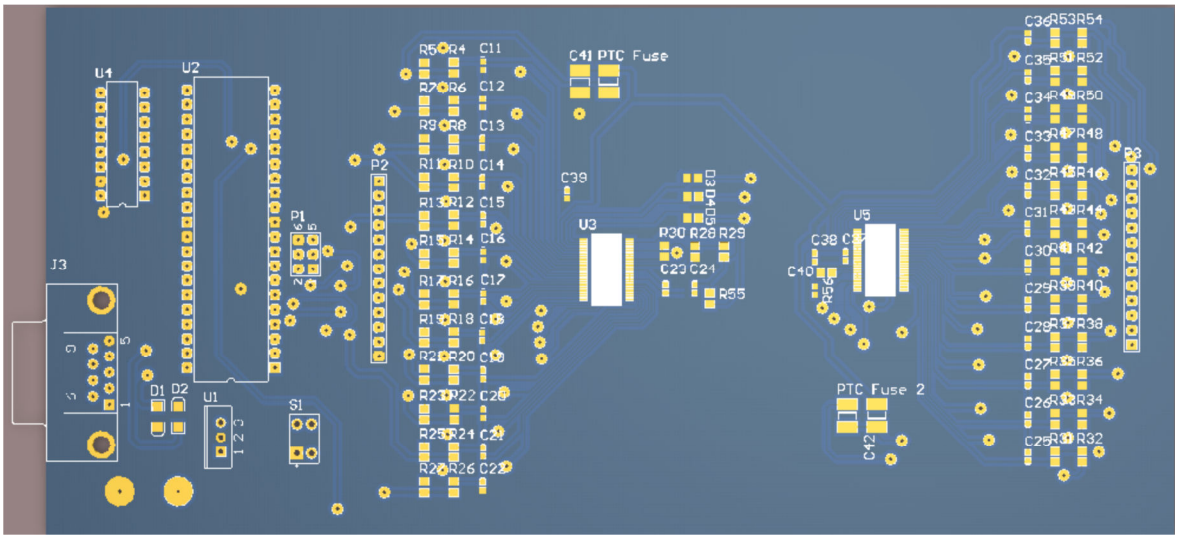

Figure 82: 2D Top view of PCB

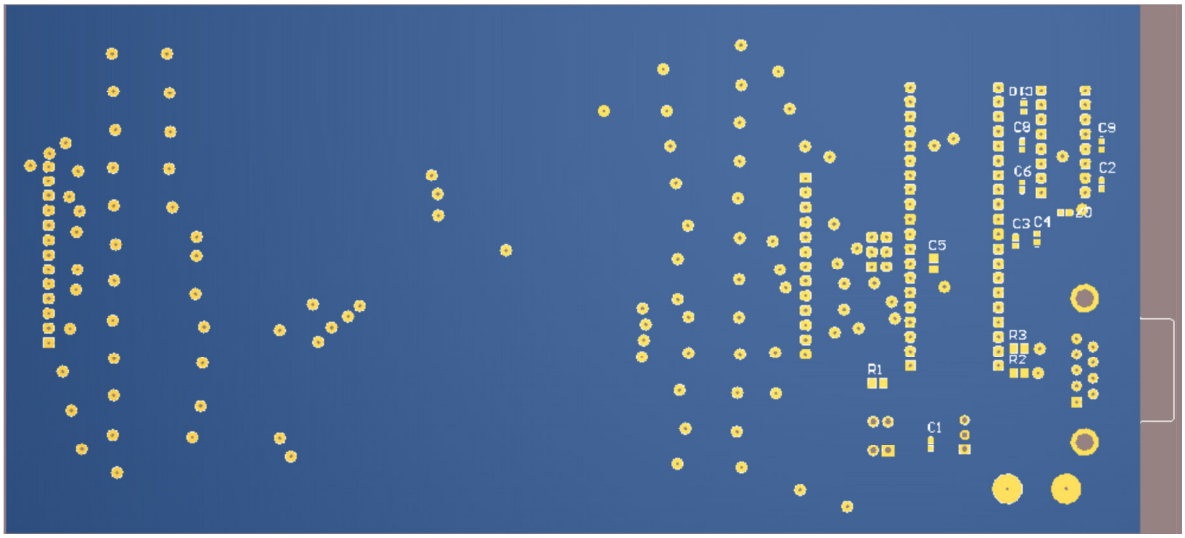

Figure 83: 2D Bottom view of PCB

Figure 84 shows the simulator board. Layout of this board was done with the prototype board in mind. Two cell simulator boards had to fit onto the prototype and it was designed to fit.

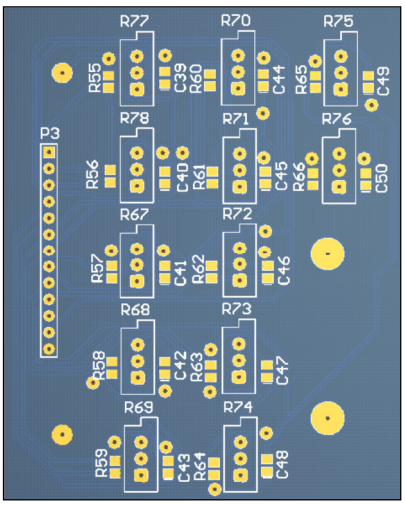

Figure 84: Cell simulator board used to simulate twelve cells of different chemistries.

Figure 85 was used to check the size and placement of all the boards before they were made. If any changes were required they could be done before manufacturing was done.

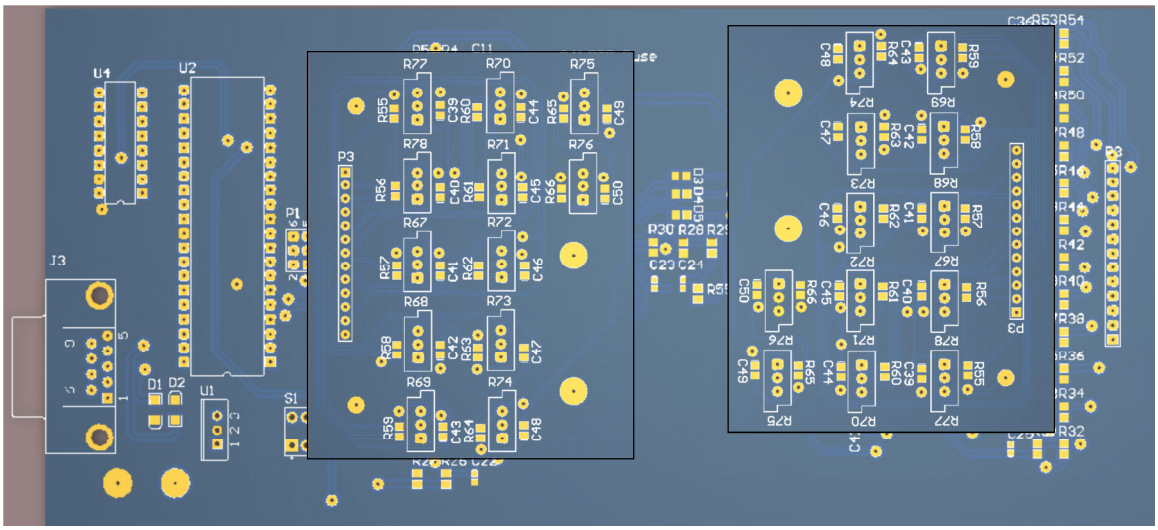

Figure 85: Prototype board with the cell simulators placed in appropriate places as it would be used when testing is done.

12.4 Appendix D: Programming Code

12.4.1 C Code

```
/*
LTC6802.c
Control Program for LTC6802-1 Development Board
Nick Prinsloo
204112923
Cape Peninsula University of Technology 2009
Rev 4.00

Rev Dates and Changes Made:
1.00    18/10/2009          Created
2.00    25/10/2009          Added the second LTC IC
2.50    06/12/2009          Changed data to allow for communication with a GUI
3.00    19/12/2009          Changed watchdog timer settings and start loop
4.00    21/12/2009          Added cell discharge switches along with Min and Max voltage storage in EEPROM
*/

//================= DEFINES ============================

/* Extra variables defined */
#define F_CPU                   8000000                 // Define overall CPU frequency
#define FOSC                    8000000                 // Define CPU frequency for USART
#define BAUD9600                51                      // Register value for 9600 Baud
#define BAUD19200               25                      // Register value for 19200 Baud
#define BAUD38400               12                      // Register value for 38400 Baud
#define BAUD9600X2              103                     // Register value for 9600 Baud x 2
#define BAUD19200X2             51                      // Register value for 19200 Baud x 2
#define BAUD38400X2             25                      // Register value for 38400 Baud x 2

#define SS              PB4                             // SS = PORTB4
#define MOSI            PB5                             // MOSI = PORTB5
#define MISO            PB6                             // MISO = PORTB6
#define SCK             PB7                             // SCK = PORTB7

#define STATUS          PORTA
#define RUNNING         PORTA1                          // Blue LED
#define STOP            PORTA0                          // Red LED

// REGISTER VALUES FOR LTC6802-1 CHIP

#define WRCFG           0x01                            // Write configuration register group
#define RDCFG           0x02                            // Read configuration register group
#define RDCV            0x04                            // Read cell voltage register group
#define RDFLG           0x06                            // Read flag register group
#define RDTMP           0x08                            // Read temperature register group
#define STCVAD          0x10                            // Start cell voltage AD conversion
#define CST1            0x1E                            // Produce 0x555 value, decimal 1365
#define CST2            0x1F                            // Produce 0xAAA value, decimal
#define STOWAD          0x20                            // Start open wire AD conversion
#define STTMPAD         0x30                            // Start temperature AD conversion
#define PLADC           0x40                            // Poll AD converter status
#define PLINT           0x50                            // Poll interrupt status
#define STCVDC          0x60                            // Start cell voltage AD conversion
                                                        // with discharge permitted
#define STOWDC          0x70                            // Start open wire AD conversion
                                                        // with discharge permitted

// END LTC6802-1 REGISTER DEFINITIONS


/* Initialisations and Includes */
#include <avr\io.h>                                     // Basic input and output declaration
#include <avr\interrupt.h>                              // Interrupt declarations
#include <avr\sleep.h>                                  // Sleep mode declarations
#include <avr\wdt.h>                                    // Watch Dog Timer mode declaration
```

```
#include <util\delay.h>                                  // Delay function declaration
#include <stdlib.h>                                      // Include some conversion functions
#include <avr\eeprom.h>                                  // Include EEPROM handling functions

//================= FUNCTIONS ========================

/* Setting the BAUD rate */
void USART_Int(void)
{

        UBRRH = (unsigned char)(BAUD38400>>8);           // Set baud rate high register
        UBRRL = (unsigned char)BAUD38400;                // Set baud rate low register

        UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);        // Set data to 8 bits, one stop bit
                                                         // and no parity bits
}

/* USART Enabled */
void USART_Enable(void)
{

        UCSRB = (1<<TXEN)|(1<<RXEN);                     // Enable receiver

}

/* USART Disabled */
void USART_Disable(void)
{

        UCSRB = (0<<TXEN)|(0<<RXEN);                     // Disable receiver

}

/* Transmit data through USART if USART is not busy */
void USART_Transmit( long data )
{

        while ( !( UCSRA & (1<<UDRE)) )                  // Wait for empty transmit buffer
        ;

        UDR = data;                                      // Put data into buffer, sends the data

}

// Transmit string through USART
void USART_TransmitString( char *data)
{

        while (*data) USART_Transmit(*data++);           // Send string through USART port

}

// Enable SPI interface
void SPI_Master_Enable( void )
{

        DDRB = (1<<MOSI)|(1<<SCK)|(1<<SS);               // Set MOSI and SCK output
                                                         // all others input
        SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);             // Enable SPI, Master, set clock
                                                         // rate F_CPU/4 i.e. 1MHz
        SPSR |= (1<<SPI2X);                              // Double rate
        SPCR |= (1<<CPOL)|(1<<CPHA);                     // Set clock phase and polarity

        PORTB |= (1<<SS);                                // Pull slave select high

}

// Send Data via SPI
void SPI_MasterTransmit(char cData)
{
```

```
        SPDR = cData;                                   // Load data into transfer register

        while(!(SPSR & (1<<SPIF)))                      // Wait for SPI transfer to complete
        ;

}

void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{

        while(EECR & (1<<EEWE))                         // Wait for previous write to complete
        ;

        EEAR = uiAddress;                               // Set address
        EEDR = ucData;                                  // Set data to be written

        EECR |= (1<<EEMWE);                             // Set master write enable bit

        EECR |= (1<<EEWE);                              // Set write enable bit and write
}

unsigned char EEPROM_read(unsigned int uiAddress)
{

        while(EECR & (1<<EEWE))                         // Check if a write is in progress
        ;

        EEAR = uiAddress;                               // Set address

        EECR |= (1<<EERE);                              // Set read enable bit

        return EEDR;                                    // Return value to the caller
}

//================ MAIN PROGRAM ====================

int main( )
{
                                                        // Set config values for CFG registers
//      BOTTOM CHIP CONFIG REGISTERS
        char CFG1R0 = 0b01100001;                       // Watchdog and CDC settings
        char CFG1R1 = 0b00000000;                       // Discharge switches
        char CFG1R2 = 0b00000000;                       // Discharge switches and masking
        char CFG1R3 = 0b00000000;                       // Masking interrupts
        char CFG1R4 = 0b01100100;                       // 2.4V Undervoltage
        char CFG1R5 = 0b10010110;                       // 3.6V Overvoltage

//      HIGH CHIP CONFIG REGISTERS
        char CFG2R0 = 0b01100001;                       // Watchdog and CDC settings
        char CFG2R1 = 0b00000000;                       // Discharge switches
        char CFG2R2 = 0b00000000;                       // Discharge switches and masking
        char CFG2R3 = 0b00000000;                       // Masking interrupts
        char CFG2R4 = 0b01100100;                       // 2.4V Undervoltage
        char CFG2R5 = 0b10010110;                       // 3.6V Overvoltage

        char Dummy = 0b01010101;                        // Dummy variable for SPI reads

//      CHIP1
        char ADC1, ADC2, ADC3, ADC4, ADC5, ADC6, ADC7, ADC8;
        char ADC9, ADC10, ADC11, ADC12, ADC13, ADC14;
        char ADC15, ADC16, ADC17, ADC18;
        char PECVOLT, PECTEMP, PECFLG1;
        char TMPR0, TMPR1, TMPR2, TMPR3, TMPR4;
        char FLGR0, FLGR1, FLGR2;

//      CHIP2
        char ADC21, ADC22, ADC23, ADC24, ADC25, ADC26, ADC27, ADC28;
        char ADC29, ADC210, ADC211, ADC212, ADC213, ADC214;
        char ADC215, ADC216, ADC217, ADC218;
        char PECVOLT2, PECTEMP2, PECFLG2;
        char TMPR20, TMPR21, TMPR22, TMPR23, TMPR24;
```

```
        char FLGR20, FLGR21, FLGR22;

//      STORED CELL VALUES
        unsigned char HighHCell1, HighLCell1, LowHCell1, LowLCell1;
        unsigned char HighHCell2, HighLCell2, LowHCell2, LowLCell2;
        unsigned char HighHCell3, HighLCell3, LowHCell3, LowLCell3;
        unsigned char HighHCell4, HighLCell4, LowHCell4, LowLCell4;
        unsigned char HighHCell5, HighLCell5, LowHCell5, LowLCell5;
        unsigned char HighHCell6, HighLCell6, LowHCell6, LowLCell6;
        unsigned char HighHCell7, HighLCell7, LowHCell7, LowLCell7;
        unsigned char HighHCell8, HighLCell8, LowHCell8, LowLCell8;
        unsigned char HighHCell9, HighLCell9, LowHCell9, LowLCell9;
        unsigned char HighHCell10, HighLCell10, LowHCell10, LowLCell10;
        unsigned char HighHCell11, HighLCell11, LowHCell11, LowLCell11;
        unsigned char HighHCell12, HighLCell12, LowHCell12, LowLCell12;
        unsigned char HighHCell13, HighLCell13, LowHCell13, LowLCell13;
        unsigned char HighHCell14, HighLCell14, LowHCell14, LowLCell14;
        unsigned char HighHCell15, HighLCell15, LowHCell15, LowLCell15;
        unsigned char HighHCell16, HighLCell16, LowHCell16, LowLCell16;
        unsigned char HighHCell17, HighLCell17, LowHCell17, LowLCell17;
        unsigned char HighHCell18, HighLCell18, LowHCell18, LowLCell18;
        unsigned char HighHCell19, HighLCell19, LowHCell19, LowLCell19;
        unsigned char HighHCell20, HighLCell20, LowHCell20, LowLCell20;
        unsigned char HighHCell21, HighLCell21, LowHCell21, LowLCell21;
        unsigned char HighHCell22, HighLCell22, LowHCell22, LowLCell22;
        unsigned char HighHCell23, HighLCell23, LowHCell23, LowLCell23;
        unsigned char HighHCell24, HighLCell24, LowHCell24, LowLCell24;

        _delay_us(10);                                  // Short delay before program starts

        wdt_enable(WDTO_2S);                            // Setup watch dog timer for 2sec reset

        DDRA = 0xFF;                                    // Set PORTA all outputs for status LEDs

        STATUS = (1<<RUNNING);                          // Enable blue LED

        sei();
        // Enable system interrupts

        SPI_Master_Enable();                            // Enable master SPI mode

// START Send Config                                    // Send a command to the LTC Chips

//      CHIP HIGH
        PORTB &= ~(1<<SS);                              // Pull slave select low

        SPI_MasterTransmit(WRCFG);                      // Send command address
                                                        // Send config registers, they are the
                                                        // different for all the LTC Chips
        SPI_MasterTransmit(CFG1R0);                     // Send config register 0
        SPI_MasterTransmit(CFG1R1);                     // Send config register 1
        SPI_MasterTransmit(CFG1R2);                     // Send config register 2
        SPI_MasterTransmit(CFG1R3);                     // Send config register 3
        SPI_MasterTransmit(CFG1R4);                     // Send config register 4
        SPI_MasterTransmit(CFG1R5);                     // Send config register 5

//      CHIP LOW
        SPI_MasterTransmit(CFG2R0);                     // Send config register 0
        SPI_MasterTransmit(CFG2R1);                     // Send config register 1
        SPI_MasterTransmit(CFG2R2);                     // Send config register 2
        SPI_MasterTransmit(CFG2R3);                     // Send config register 3
        SPI_MasterTransmit(CFG2R4);                     // Send config register 4
        SPI_MasterTransmit(CFG2R5);                     // Send config register 5

        PORTB |= (1<<SS);                               // Pull slave select high

// END Send Config

//      READ STORED CELL VALUES                         // Read values out of the EEPROM
        HighLCell1 = EEPROM_read(1);                    // from address 1 to 96 and store
        HighHCell1 = EEPROM_read(2);                    // in the appropriate variable
```

```
LowLCell1 = EEPROM_read(3);
LowHCell1 = EEPROM_read(4);

HighLCell2 = EEPROM_read(5);
HighHCell2 = EEPROM_read(6);
LowLCell2 = EEPROM_read(7);
LowHCell2 = EEPROM_read(8);

HighLCell3 = EEPROM_read(9);
HighHCell3 = EEPROM_read(10);
LowLCell3 = EEPROM_read(11);
LowHCell3 = EEPROM_read(12);

HighLCell4 = EEPROM_read(13);
HighHCell4 = EEPROM_read(14);
LowLCell4 = EEPROM_read(15);
LowHCell4 = EEPROM_read(16);

HighLCell5 = EEPROM_read(17);
HighHCell5 = EEPROM_read(18);
LowLCell5 = EEPROM_read(19);
LowHCell5 = EEPROM_read(20);

HighLCell6 = EEPROM_read(21);
HighHCell6 = EEPROM_read(22);
LowLCell6 = EEPROM_read(23);
LowHCell6 = EEPROM_read(24);

HighLCell7 = EEPROM_read(25);
HighHCell7 = EEPROM_read(26);
LowLCell7 = EEPROM_read(27);
LowHCell7 = EEPROM_read(28);

HighLCell8 = EEPROM_read(29);
HighHCell8 = EEPROM_read(30);
LowLCell8 = EEPROM_read(31);
LowHCell8 = EEPROM_read(32);

HighLCell9 = EEPROM_read(33);
HighHCell9 = EEPROM_read(34);
LowLCell9 = EEPROM_read(35);
LowHCell9 = EEPROM_read(36);

HighLCell10 = EEPROM_read(37);
HighHCell10 = EEPROM_read(38);
LowLCell10 = EEPROM_read(39);
LowHCell10 = EEPROM_read(40);

HighLCell11 = EEPROM_read(41);
HighHCell11 = EEPROM_read(42);
LowLCell11 = EEPROM_read(43);
LowHCell11 = EEPROM_read(44);

HighLCell12 = EEPROM_read(45);
HighHCell12 = EEPROM_read(46);
LowLCell12 = EEPROM_read(47);
LowHCell12 = EEPROM_read(48);

HighLCell13 = EEPROM_read(49);
HighHCell13 = EEPROM_read(50);
LowLCell13 = EEPROM_read(51);
LowHCell13 = EEPROM_read(52);

HighLCell14 = EEPROM_read(53);
HighHCell14 = EEPROM_read(54);
LowLCell14 = EEPROM_read(55);
LowHCell14 = EEPROM_read(56);

HighLCell15 = EEPROM_read(57);
HighHCell15 = EEPROM_read(58);
LowLCell15 = EEPROM_read(59);
```

```
        LowHCell15 = EEPROM_read(60);

        HighLCell16 = EEPROM_read(61);
        HighHCell16 = EEPROM_read(62);
        LowLCell16 = EEPROM_read(63);
        LowHCell16 = EEPROM_read(64);

        HighLCell17 = EEPROM_read(65);
        HighHCell17 = EEPROM_read(66);
        LowLCell17 = EEPROM_read(67);
        LowHCell17 = EEPROM_read(68);

        HighLCell18 = EEPROM_read(69);
        HighHCell18 = EEPROM_read(70);
        LowLCell18 = EEPROM_read(71);
        LowHCell18 = EEPROM_read(72);

        HighLCell19 = EEPROM_read(73);
        HighHCell19 = EEPROM_read(74);
        LowLCell19 = EEPROM_read(75);
        LowHCell19 = EEPROM_read(76);

        HighLCell20 = EEPROM_read(77);
        HighHCell20 = EEPROM_read(78);
        LowLCell20 = EEPROM_read(79);
        LowHCell20 = EEPROM_read(80);

        HighLCell21 = EEPROM_read(81);
        HighHCell21 = EEPROM_read(82);
        LowLCell21 = EEPROM_read(83);
        LowHCell21 = EEPROM_read(84);

        HighLCell22 = EEPROM_read(85);
        HighHCell22 = EEPROM_read(86);
        LowLCell22 = EEPROM_read(87);
        LowHCell22 = EEPROM_read(88);

        HighLCell23 = EEPROM_read(89);
        HighHCell23 = EEPROM_read(90);
        LowLCell23 = EEPROM_read(91);
        LowHCell23 = EEPROM_read(92);

        HighLCell24 = EEPROM_read(93);
        HighHCell24 = EEPROM_read(94);
        LowLCell24 = EEPROM_read(95);
        LowHCell24 = EEPROM_read(96);

// START Cell Voltage AD Conversions and Poll Status                    // Start ADC conversions

        PORTB &= ~(1<<SS);                                              // Pull slave select low

        SPI_MasterTransmit(STCVDC);                                     // Send start conversion command

        _delay_ms(18);                                                  // Wait for conversions to complete

        PORTB |= (1<<SS);                                               // Pull slave select high

// END Cell Voltage AD Conversions and Poll Status

// START Read Cell Voltage Registers                                    // Read ADC values into variables

        PORTB &= ~(1<<SS);                                              // Pull slave select low

        SPI_MasterTransmit(RDCV);                                       // Send read voltage command

//      CHIP LOW
        SPI_MasterTransmit(Dummy);                                      // Transmit dummy and receive data
        ADC1 = SPDR;
        SPI_MasterTransmit(Dummy);                                      // Same for all the registers
        ADC2 = SPDR;
        SPI_MasterTransmit(Dummy);
```

```
        ADC3 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC4 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC5 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC6 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC7 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC8 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC9 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC10 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC11 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC12 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC13 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC14 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC15 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC16 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC17 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC18 = SPDR;

//      PEC
        SPI_MasterTransmit(Dummy);
        PECVOLT = SPDR;

//      CHIP HIGH
        SPI_MasterTransmit(Dummy);
        ADC21 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC22 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC23 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC24 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC25 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC26 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC27 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC28 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC29 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC210 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC211 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC212 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC213 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC214 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC215 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC216 = SPDR;
        SPI_MasterTransmit(Dummy);
        ADC217 = SPDR;
```

```
        SPI_MasterTransmit(Dummy);
        ADC218 = SPDR;

//      PEC
        SPI_MasterTransmit(Dummy);
        PECVOLT2 = SPDR;


        PORTB |= (1<<SS);                       // Pull slave select high

// END Read Cell Voltage Registers

// START Temperature AD Conversions and Poll Status

        PORTB &= ~(1<<SS);                      // Pull slave select low

        SPI_MasterTransmit(STTMPAD);            // Send start conversion command

        _delay_ms(5);                           // Wait for conversions to complete

        PORTB |= (1<<SS);                       // Pull slave select high

// END Temperature AD Conversions and Poll Status

// START Read Temp Registers

        PORTB &= ~(1<<SS);                      // Pull slave select low

        SPI_MasterTransmit(RDTMP);              // Send read voltage command

//      CHIP LOW
        SPI_MasterTransmit(Dummy);              // Transmit dummy and receive data
        TMPR0 = SPDR;
        SPI_MasterTransmit(Dummy);              // Same for all the registers
        TMPR1 = SPDR;
        SPI_MasterTransmit(Dummy);
        TMPR2 = SPDR;
        SPI_MasterTransmit(Dummy);
        TMPR3 = SPDR;
        SPI_MasterTransmit(Dummy);
        TMPR4 = SPDR;

//      PEC
        SPI_MasterTransmit(Dummy);
        PECTEMP = SPDR;

//      CHIP HIGH
        SPI_MasterTransmit(Dummy);
        TMPR20 = SPDR;
        SPI_MasterTransmit(Dummy);
        TMPR21 = SPDR;
        SPI_MasterTransmit(Dummy);
        TMPR22 = SPDR;
        SPI_MasterTransmit(Dummy);
        TMPR23 = SPDR;
        SPI_MasterTransmit(Dummy);
        TMPR24 = SPDR;

//      PEC
        SPI_MasterTransmit(Dummy);
        PECTEMP2 = SPDR;

        PORTB |= (1<<SS);                       // Pull slave select high

// END Read Temp Registers

// START Read Flag Registers

        PORTB &= ~(1<<SS);                      // Pull slave select low

        SPI_MasterTransmit(RDFLG);              // Send read voltage command
```

```
//          CHIP LOW
            SPI_MasterTransmit(Dummy);                          // Transmit dummy and receive data
            FLGR0 = SPDR;
            SPI_MasterTransmit(Dummy);                          // Same for all registers
            FLGR1 = SPDR;
            SPI_MasterTransmit(Dummy);
            FLGR2 = SPDR;

//          PEC
            SPI_MasterTransmit(Dummy);
            PECFLG1 = SPDR;

//          CHIP HIGH
            SPI_MasterTransmit(Dummy);
            FLGR20 = SPDR;
            SPI_MasterTransmit(Dummy);
            FLGR21 = SPDR;
            SPI_MasterTransmit(Dummy);
            FLGR22 = SPDR;

//          PEC
            SPI_MasterTransmit(Dummy);    a
            PECFLG2 = SPDR;

            PORTB |= (1<<SS);                                   // Pull slave select high

// END Read Flag Registers


// START Cell Voltage Calculations

            unsigned char Temp1, Temp2, Temp3 = 0;              // Temp bit registers for voltage
            char buf[5];

//          CHIP1
            volatile uint16_t Cell1, Cell2, Cell3, Cell4, Cell5;
            volatile uint16_t Cell6, Cell7, Cell8, Cell9, Cell10;
            volatile uint16_t Cell11, Cell12;

            volatile uint16_t TEMPEX1, TEMPEX2, TEMPINT;        // Temp bit registers for temperature

//          CHIP2
            volatile uint16_t Cell21, Cell22, Cell23, Cell24, Cell25;
            volatile uint16_t Cell26, Cell27, Cell28, Cell29, Cell210;
            volatile uint16_t Cell211, Cell212;


            volatile uint16_t TEMPEX21, TEMPEX22, TEMPINT2;     // Temp bit registers for temperature

//          HIGH AND LOW VOLTAGE FROM EEPROM                    // Variables for Highest and Lowest voltage
            volatile uint16_t CellHV1, CellLV1, CellHV2, CellLV2;
            volatile uint16_t CellHV3, CellLV3, CellHV4, CellLV4;
            volatile uint16_t CellHV5, CellLV5, CellHV6, CellLV6;
            volatile uint16_t CellHV7, CellLV7, CellHV8, CellLV8;
            volatile uint16_t CellHV9, CellLV9, CellHV10, CellLV10;
            volatile uint16_t CellHV11, CellLV11, CellHV12, CellLV12;
            volatile uint16_t CellHV13, CellLV13, CellHV14, CellLV14;
            volatile uint16_t CellHV15, CellLV15, CellHV16, CellLV16;
            volatile uint16_t CellHV17, CellLV17, CellHV18, CellLV18;
            volatile uint16_t CellHV19, CellLV19, CellHV20, CellLV20;
            volatile uint16_t CellHV21, CellLV21, CellHV22, CellLV22;
            volatile uint16_t CellHV23, CellLV23, CellHV24, CellLV24;

//          CHIP LOW
//          CELL1
            Temp1 = ADC1;                                       // Move 8 bits to variable
            Temp2 = ADC2;                                       // Move 8 bits to variable
            Temp3 = (Temp2 & 0x0F);                             // Mask the 4 MSBs
            Cell1 = ( Temp3 << 8 ) + Temp1;
```

```
//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell1;                     // Move 8 bits to variable
            Temp2 = HighHCell1;                     // Move 8 bits to variable
            Temp3 = (Temp2 & 0x0F);                 // Mask the 4 MSBs
            CellHV1 = ( Temp3 << 8 ) + Temp1;
//          Low
            Temp1 = LowLCell1;                      // Move 8 bits to variable
            Temp2 = LowHCell1;                      // Move 8 bits to variable
            Temp3 = (Temp2 & 0x0F);                 // Mask the 4 MSBs
            CellLV1 = ( Temp3 << 8 ) + Temp1;

            if (Cell1>CellHV1)                      // Check if current value is higher
            {
            EEPROM_write(1, ADC1);
            EEPROM_write(2, ADC2);
            }

            if (Cell1<CellLV1)                      // Check if current value is lower
            {
            EEPROM_write(3, ADC1);
            EEPROM_write(4, ADC2);
            }

//          CELL2
            Temp1 = ADC2;                           // Comment above applies to all 24 cells
            Temp2 = ADC3;
            Temp3 = (Temp1 & 0xF0);
            Cell2 = ( Temp2 << 8 ) + Temp3;
            Cell2 = (Cell2 >> 4);

//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell2;
            Temp2 = HighHCell2;
            Temp3 = (Temp1 & 0xF0);
            CellHV2 = ( Temp2 << 8 ) + Temp3;
            CellHV2 = (CellHV2 >> 4);
//          Low
            Temp1 = LowLCell2;
            Temp2 = LowHCell2;
            Temp3 = (Temp1 & 0xF0);
            CellLV2 = ( Temp2 << 8 ) + Temp3;
            CellLV2 = (CellLV2 >> 4);

            if (Cell2>CellHV2)
            {
            EEPROM_write(5, ADC2);
            EEPROM_write(6, ADC3);
            }

            if (Cell2<CellLV2)
            {
            EEPROM_write(7, ADC2);
            EEPROM_write(8, ADC3);
            }

//          CELL3
            Temp1 = ADC4;
            Temp2 = ADC5;
            Temp3 = (Temp2 & 0x0F);
            Cell3 = ( Temp3 << 8 ) + Temp1;

//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell3;
            Temp2 = HighHCell3;
            Temp3 = (Temp2 & 0x0F);
            CellHV3 = ( Temp3 << 8 ) + Temp1;
//          Low
            Temp1 = LowLCell3;
```

```
        Temp2 = LowHCell3;
        Temp3 = (Temp2 & 0x0F);
        CellLV3 = ( Temp3 << 8 ) + Temp1;

        if (Cell3>CellHV3)
        {
        EEPROM_write(9, ADC4);
        EEPROM_write(10, ADC5);
        }

        if (Cell3<CellLV3)
        {
        EEPROM_write(11, ADC4);
        EEPROM_write(12, ADC5);
        }

//      CELL4
        Temp1 = ADC5;
        Temp2 = ADC6;
        Temp3 = (Temp1 & 0xF0);
        Cell4 = ( Temp2 << 8 ) + Temp3;
        Cell4 = (Cell4 >> 4);

//      Reconstruct High and Low values
//      High
        Temp1 = HighLCell4;
        Temp2 = HighHCell4;
        Temp3 = (Temp1 & 0xF0);
        CellHV4 = ( Temp2 << 8 ) + Temp3;
        CellHV4 = (CellHV4 >> 4);
//      Low
        Temp1 = LowLCell4;
        Temp2 = LowHCell4;
        Temp3 = (Temp1 & 0xF0);
        CellLV4 = ( Temp2 << 8 ) + Temp3;
        CellLV4 = (CellLV4 >> 4);

        if (Cell4>CellHV4)
        {
        EEPROM_write(13, ADC5);
        EEPROM_write(14, ADC6);
        }

        if (Cell4<CellLV4)
        {
        EEPROM_write(15, ADC5);
        EEPROM_write(16, ADC6);
        }

//      CELL5
        Temp1 = ADC7;
        Temp2 = ADC8;
        Temp3 = (Temp2 & 0x0F);
        Cell5 = ( Temp3 << 8 ) + Temp1;

//      Reconstruct High and Low values
//      High
        Temp1 = HighLCell5;
        Temp2 = HighHCell5;
        Temp3 = (Temp2 & 0x0F);
        CellHV5 = ( Temp3 << 8 ) + Temp1;
//      Low
        Temp1 = LowLCell5;
        Temp2 = LowHCell5;
        Temp3 = (Temp2 & 0x0F);
        CellLV5 = ( Temp3 << 8 ) + Temp1;

        if (Cell5>CellHV5)
        {
        EEPROM_write(17, ADC7);
        EEPROM_write(18, ADC8);
```

```
        }

        if (Cell5<CellLV5)
        {
        EEPROM_write(19, ADC7);
        EEPROM_write(20, ADC8);
        }

//      CELL6
        Temp1 = ADC8;
        Temp2 = ADC9;
        Temp3 = (Temp1 & 0xF0);
        Cell6 = ( Temp2 << 8 ) + Temp3;
        Cell6 = (Cell6 >> 4);

//      Reconstruct High and Low values
//      High
        Temp1 = HighLCell6;
        Temp2 = HighHCell6;
        Temp3 = (Temp1 & 0xF0);
        CellHV6 = ( Temp2 << 8 ) + Temp3;
        CellHV6 = (CellHV6 >> 4);
//      Low
        Temp1 = LowLCell6;
        Temp2 = LowHCell6;
        Temp3 = (Temp1 & 0xF0);
        CellLV6 = ( Temp2 << 8 ) + Temp3;
        CellLV6 = (CellLV6 >> 4);

        if (Cell6>CellHV6)
        {
        EEPROM_write(21, ADC8);
        EEPROM_write(22, ADC9);
        }

        if (Cell6<CellLV6)
        {
        EEPROM_write(23, ADC8);
        EEPROM_write(24, ADC9);
        }

//      CELL7
        Temp1 = ADC10;
        Temp2 = ADC11;
        Temp3 = (Temp2 & 0x0F);
        Cell7 = ( Temp3 << 8 ) + Temp1;

//      Reconstruct High and Low values
//      High
        Temp1 = HighLCell7;
        Temp2 = HighHCell7;
        Temp3 = (Temp2 & 0x0F);
        CellHV7 = ( Temp3 << 8 ) + Temp1;
//      Low
        Temp1 = LowLCell7;
        Temp2 = LowHCell7;
        Temp3 = (Temp2 & 0x0F);
        CellLV7 = ( Temp3 << 8 ) + Temp1;

        if (Cell7>CellHV7)
        {
        EEPROM_write(25, ADC10);
        EEPROM_write(26, ADC11);
        }

        if (Cell7<CellLV7)
        {
        EEPROM_write(27, ADC10);
        EEPROM_write(28, ADC11);
        }
```

```
//          CELL8
            Temp1 = ADC11;
            Temp2 = ADC12;
            Temp3 = (Temp1 & 0xF0);
            Cell8 = ( Temp2 << 8 ) + Temp3;
            Cell8 = (Cell8 >> 4);

//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell8;
            Temp2 = HighHCell8;
            Temp3 = (Temp1 & 0xF0);
            CellHV8 = ( Temp2 << 8 ) + Temp3;
            CellHV8 = (CellHV8 >> 4);
//          Low
            Temp1 = LowLCell8;
            Temp2 = LowHCell8;
            Temp3 = (Temp1 & 0xF0);
            CellLV8 = ( Temp2 << 8 ) + Temp3;
            CellLV8 = (CellLV8 >> 4);

            if (Cell8>CellHV8)
            {
            EEPROM_write(29, ADC11);
            EEPROM_write(30, ADC12);
            }

            if (Cell8<CellLV8)
            {
            EEPROM_write(31, ADC11);
            EEPROM_write(32, ADC12);
            }

//          CELL9
            Temp1 = ADC13;
            Temp2 = ADC14;
            Temp3 = (Temp2 & 0x0F);
            Cell9 = ( Temp3 << 8 ) + Temp1;

//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell9;
            Temp2 = HighHCell9;
            Temp3 = (Temp2 & 0x0F);
            CellHV9 = ( Temp3 << 8 ) + Temp1;
//          Low
            Temp1 = LowLCell9;
            Temp2 = LowHCell9;
            Temp3 = (Temp2 & 0x0F);
            CellLV9 = ( Temp3 << 8 ) + Temp1;

            if (Cell9>CellHV9)
            {
            EEPROM_write(33, ADC13);
            EEPROM_write(34, ADC14);
            }

            if (Cell9<CellLV9)
            {
            EEPROM_write(35, ADC13);
            EEPROM_write(36, ADC14);
            }

//          CELL10
            Temp1 = ADC14;
            Temp2 = ADC15;
            Temp3 = (Temp1 & 0xF0);
            Cell10 = ( Temp2 << 8 ) + Temp3;
            Cell10 = (Cell10 >> 4);

//          Reconstruct High and Low values
```

```
//          High
            Temp1 = HighLCell10;
            Temp2 = HighHCell10;
            Temp3 = (Temp1 & 0xF0);
            CellHV10 = ( Temp2 << 8 ) + Temp3;
            CellHV10 = (CellHV10 >> 4);
//          Low
            Temp1 = LowLCell10;
            Temp2 = LowHCell10;
            Temp3 = (Temp1 & 0xF0);
            CellLV10 = ( Temp2 << 8 ) + Temp3;
            CellLV10 = (CellLV10 >> 4);

            if (Cell10>CellHV10)
            {
            EEPROM_write(37, ADC14);
            EEPROM_write(38, ADC15);
            }

            if (Cell10<CellLV10)
            {
            EEPROM_write(39, ADC14);
            EEPROM_write(40, ADC15);
            }

//          CELL11
            Temp1 = ADC16;
            Temp2 = ADC17;
            Temp3 = (Temp2 & 0x0F);
            Cell11 = ( Temp3 << 8 ) + Temp1;

//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell11;
            Temp2 = HighHCell11;
            Temp3 = (Temp2 & 0x0F);
            CellHV11 = ( Temp3 << 8 ) + Temp1;
//          Low
            Temp1 = LowLCell11;
            Temp2 = LowHCell11;
            Temp3 = (Temp2 & 0x0F);
            CellLV11 = ( Temp3 << 8 ) + Temp1;

            if (Cell11>CellHV11)
            {
            EEPROM_write(41, ADC16);
            EEPROM_write(42, ADC17);
            }

            if (Cell11<CellLV11)
            {
            EEPROM_write(43, ADC16);
            EEPROM_write(44, ADC17);
            }

//          CELL12
            Temp1 = ADC17;
            Temp2 = ADC18;
            Temp3 = (Temp1 & 0xF0);
            Cell12 = ( Temp2 << 8 ) + Temp3;
            Cell12 = (Cell12 >> 4);

//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell12;
            Temp2 = HighHCell12;
            Temp3 = (Temp1 & 0xF0);
            CellHV12 = ( Temp2 << 8 ) + Temp3;
            CellHV12 = (CellHV12 >> 4);
//          Low
            Temp1 = LowLCell12;
```

```
        Temp2 = LowHCell12;
        Temp3 = (Temp1 & 0xF0);
        CellLV12 = ( Temp2 << 8 ) + Temp3;
        CellLV12 = (CellLV12 >> 4);

        if (Cell12>CellHV12)
        {
        EEPROM_write(45, ADC17);
        EEPROM_write(46, ADC18);
        }

        if (Cell12<CellLV12)
        {
        EEPROM_write(47, ADC17);
        EEPROM_write(48, ADC18);
        }

//      CHIP HIGH
//      CELL13
        Temp1 = ADC21;
        Temp2 = ADC22;
        Temp3 = (Temp2 & 0x0F);
        Cell21 = ( Temp3 << 8 ) + Temp1;

//      Reconstruct High and Low values
//      High
        Temp1 = HighLCell13;
        Temp2 = HighHCell13;
        Temp3 = (Temp2 & 0x0F);
        CellHV13 = ( Temp3 << 8 ) + Temp1;
//      Low
        Temp1 = LowLCell13;
        Temp2 = LowHCell13;
        Temp3 = (Temp2 & 0x0F);
        CellLV13 = ( Temp3 << 8 ) + Temp1;

        if (Cell21>CellHV13)
        {
        EEPROM_write(49, ADC21);
        EEPROM_write(50, ADC22);
        }

        if (Cell21<CellLV13)
        {
        EEPROM_write(51, ADC21);
        EEPROM_write(52, ADC22);
        }

//      CELL14
        Temp1 = ADC22;
        Temp2 = ADC23;
        Temp3 = (Temp1 & 0xF0);
        Cell22 = ( Temp2 << 8 ) + Temp3;
        Cell22 = (Cell22 >> 4);

//      Reconstruct High and Low values
//      High
        Temp1 = HighLCell14;
        Temp2 = HighHCell14;
        Temp3 = (Temp1 & 0xF0);
        CellHV14 = ( Temp2 << 8 ) + Temp3;
        CellHV14 = (CellHV14 >> 4);
//      Low
        Temp1 = LowLCell2;
        Temp2 = LowHCell2;
        Temp3 = (Temp1 & 0xF0);
        CellLV14 = ( Temp2 << 8 ) + Temp3;
        CellLV14 = (CellLV14 >> 4);
```

```
            if (Cell22>CellHV14)
            {
            EEPROM_write(53, ADC22);
            EEPROM_write(54, ADC23);
            }

            if (Cell22<CellLV14)
            {
            EEPROM_write(55, ADC22);
            EEPROM_write(56, ADC23);
            }

//          CELL15
            Temp1 = ADC24;
            Temp2 = ADC25;
            Temp3 = (Temp2 & 0x0F);
            Cell23 = ( Temp3 << 8 ) + Temp1;

//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell15;
            Temp2 = HighHCell15;
            Temp3 = (Temp2 & 0x0F);
            CellHV15 = ( Temp3 << 8 ) + Temp1;
//          Low
            Temp1 = LowLCell15;
            Temp2 = LowHCell15;
            Temp3 = (Temp2 & 0x0F);
            CellLV15 = ( Temp3 << 8 ) + Temp1;

            if (Cell23>CellHV15)
            {
            EEPROM_write(57, ADC24);
            EEPROM_write(58, ADC25);
            }

            if (Cell23<CellLV15)
            {
            EEPROM_write(59, ADC24);
            EEPROM_write(60, ADC25);
            }

//          CELL16
            Temp1 = ADC25;
            Temp2 = ADC26;
            Temp3 = (Temp1 & 0xF0);
            Cell24 = ( Temp2 << 8 ) + Temp3;
            Cell24 = (Cell24 >> 4);

//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell16;
            Temp2 = HighHCell16;
            Temp3 = (Temp1 & 0xF0);
            CellHV16 = ( Temp2 << 8 ) + Temp3;
            CellHV16 = (CellHV16 >> 4);
//          Low
            Temp1 = LowLCell16;
            Temp2 = LowHCell16;
            Temp3 = (Temp1 & 0xF0);
            CellLV16 = ( Temp2 << 8 ) + Temp3;
            CellLV16 = (CellLV16 >> 4);

            if (Cell24>CellHV16)
            {
            EEPROM_write(61, ADC25);
            EEPROM_write(62, ADC26);
            }
```

```
            if (Cell24<CellLV16)
            {
            EEPROM_write(63, ADC25);
            EEPROM_write(64, ADC26);
            }

//          CELL17
            Temp1 = ADC27;
            Temp2 = ADC28;
            Temp3 = (Temp2 & 0x0F);
            Cell25 = ( Temp3 << 8 ) + Temp1;

//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell17;
            Temp2 = HighHCell17;
            Temp3 = (Temp2 & 0x0F);
            CellHV17 = ( Temp3 << 8 ) + Temp1;
//          Low
            Temp1 = LowLCell17;
            Temp2 = LowHCell17;
            Temp3 = (Temp2 & 0x0F);
            CellLV17 = ( Temp3 << 8 ) + Temp1;

            if (Cell25>CellHV17)
            {
            EEPROM_write(65, ADC27);
            EEPROM_write(66, ADC28);
            }

            if (Cell25<CellLV17)
            {
            EEPROM_write(67, ADC27);
            EEPROM_write(68, ADC28);
            }

//          CELL18
            Temp1 = ADC28;
            Temp2 = ADC29;
            Temp3 = (Temp1 & 0xF0);
            Cell26 = ( Temp2 << 8 ) + Temp3;
            Cell26 = (Cell26 >> 4);

//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell18;
            Temp2 = HighHCell18;
            Temp3 = (Temp1 & 0xF0);
            CellHV18 = ( Temp2 << 8 ) + Temp3;
            CellHV18 = (CellHV18 >> 4);
//          Low
            Temp1 = LowLCell18;
            Temp2 = LowHCell18;
            Temp3 = (Temp1 & 0xF0);
            CellLV18 = ( Temp2 << 8 ) + Temp3;
            CellLV18 = (CellLV18 >> 4);

            if (Cell26>CellHV18)
            {
            EEPROM_write(69, ADC28);
            EEPROM_write(70, ADC29);
            }

            if (Cell26<CellLV18)
            {
            EEPROM_write(71, ADC28);
            EEPROM_write(72, ADC29);
            }
```

```
//          CELL19
            Temp1 = ADC210;
            Temp2 = ADC211;
            Temp3 = (Temp2 & 0x0F);
            Cell27 = ( Temp3 << 8 ) + Temp1;

//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell19;
            Temp2 = HighHCell19;
            Temp3 = (Temp2 & 0x0F);
            CellHV19 = ( Temp3 << 8 ) + Temp1;
//          Low
            Temp1 = LowLCell19;
            Temp2 = LowHCell19;
            Temp3 = (Temp2 & 0x0F);
            CellLV19 = ( Temp3 << 8 ) + Temp1;

            if (Cell27>CellHV19)
            {
            EEPROM_write(73, ADC210);
            EEPROM_write(74, ADC211);
            }

            if (Cell27<CellLV19)
            {
            EEPROM_write(75, ADC210);
            EEPROM_write(76, ADC211);
            }

//          CELL20
            Temp1 = ADC211;
            Temp2 = ADC212;
            Temp3 = (Temp1 & 0xF0);
            Cell28 = ( Temp2 << 8 ) + Temp3;
            Cell28 = (Cell28 >> 4);

//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell20;
            Temp2 = HighHCell20;
            Temp3 = (Temp1 & 0xF0);
            CellHV20 = ( Temp2 << 8 ) + Temp3;
            CellHV20 = (CellHV20 >> 4);
//          Low
            Temp1 = LowLCell20;
            Temp2 = LowHCell20;
            Temp3 = (Temp1 & 0xF0);
            CellLV20 = ( Temp2 << 8 ) + Temp3;
            CellLV20 = (CellLV20 >> 4);

            if (Cell28>CellHV20)
            {
            EEPROM_write(77, ADC211);
            EEPROM_write(78, ADC212);
            }

            if (Cell28<CellLV20)
            {
            EEPROM_write(79, ADC211);
            EEPROM_write(80, ADC212);
            }

//          CELL21
            Temp1 = ADC213;
            Temp2 = ADC214;
            Temp3 = (Temp2 & 0x0F);
            Cell29 = ( Temp3 << 8 ) + Temp1;
```

```
//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell21;
            Temp2 = HighHCell21;
            Temp3 = (Temp2 & 0x0F);
            CellHV21 = ( Temp3 << 8 ) + Temp1;
//          Low
            Temp1 = LowLCell21;
            Temp2 = LowHCell21;
            Temp3 = (Temp2 & 0x0F);
            CellLV21 = ( Temp3 << 8 ) + Temp1;

            if (Cell29>CellHV21)
            {
            EEPROM_write(81, ADC213);
            EEPROM_write(82, ADC214);
            }

            if (Cell29<CellLV21)
            {
            EEPROM_write(83, ADC213);
            EEPROM_write(84, ADC214);
            }

//          CELL22
            Temp1 = ADC214;
            Temp2 = ADC215;
            Temp3 = (Temp1 & 0xF0);
            Cell210 = ( Temp2 << 8 ) + Temp3;
            Cell210 = (Cell210 >> 4);

//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell22;
            Temp2 = HighHCell22;
            Temp3 = (Temp1 & 0xF0);
            CellHV22 = ( Temp2 << 8 ) + Temp3;
            CellHV22 = (CellHV22 >> 4);
//          Low
            Temp1 = LowLCell22;
            Temp2 = LowHCell22;
            Temp3 = (Temp1 & 0xF0);
            CellLV22 = ( Temp2 << 8 ) + Temp3;
            CellLV22 = (CellLV22 >> 4);

            if (Cell210>CellHV22)
            {
            EEPROM_write(85, ADC214);
            EEPROM_write(86, ADC215);
            }

            if (Cell210<CellLV22)
            {
            EEPROM_write(87, ADC214);
            EEPROM_write(88, ADC215);
            }

//          CELL23
            Temp1 = ADC216;
            Temp2 = ADC217;
            Temp3 = (Temp2 & 0x0F);
            Cell211 = ( Temp3 << 8 ) + Temp1;

//          Reconstruct High and Low values
//          High
            Temp1 = HighLCell23;
            Temp2 = HighHCell23;
            Temp3 = (Temp2 & 0x0F);
            CellHV23 = ( Temp3 << 8 ) + Temp1;
//          Low
            Temp1 = LowLCell23;
```

```
        Temp2 = LowHCell23;
        Temp3 = (Temp2 & 0x0F);
        CellLV23 = ( Temp3 << 8 ) + Temp1;

        if (Cell211>CellHV23)
        {
        EEPROM_write(89, ADC216);
        EEPROM_write(90, ADC217);
        }

        if (Cell211<CellLV23)
        {
        EEPROM_write(91, ADC216);
        EEPROM_write(92, ADC217);
        }

//      CELL24
        Temp1 = ADC217;
        Temp2 = ADC218;
        Temp3 = (Temp1 & 0xF0);
        Cell212 = ( Temp2 << 8 ) + Temp3;
        Cell212 = (Cell212 >> 4);

//      Reconstruct High and Low values
//      High
        Temp1 = HighLCell24;
        Temp2 = HighHCell24;
        Temp3 = (Temp1 & 0xF0);
        CellHV24 = ( Temp2 << 8 ) + Temp3;
        CellHV24 = (CellHV24 >> 4);
//      Low
        Temp1 = LowLCell24;
        Temp2 = LowHCell24;
        Temp3 = (Temp1 & 0xF0);
        CellLV24 = ( Temp2 << 8 ) + Temp3;
        CellLV24 = (CellLV24 >> 4);

        if (Cell212>CellHV24)
        {
        EEPROM_write(93, ADC217);
        EEPROM_write(94, ADC218);
        }

        if (Cell212<CellLV24)
        {
        EEPROM_write(95, ADC217);
        EEPROM_write(96, ADC218);
        }

// END Cell Voltage Calculations

//      DISCHARGE SWITCHES                                  // Variables for discharge switch state
        int     SW1, SW2, SW3, SW4, SW5, SW6, SW7;
        int SW8, SW9, SW10, SW11, SW12;
        SW1 = SW2 = SW3 = SW4 = SW5 =SW6 = SW7 = 0;
        SW8 = SW9 = SW10 = SW11 = SW12 = 0;
        int     SW13, SW14, SW15, SW16, SW17, SW18, SW19;
        int SW20, SW21, SW22, SW23, SW24;
        SW13 = SW14 = SW15 = SW16 = SW17 =SW18 = SW19 = 0;
        SW20 = SW21 = SW22 = SW23 = SW24 = 0;

//      BOTTOM LTC
        if(Cell1>Cell2)                                     // Compare against previous cell
        {
        CFG1R1 |= 1;                                        // Turn switch ON
        SW1 = 1;
        }
```

```
if(Cell2>Cell3)                                   // Compare against previous cell
{
        if((SW1==0))                              // If the previous switch is off
        {
        CFG1R1 |= 2;                              // Turn switch ON
        SW2 = 1;
        }
}

if(Cell3>Cell4)                                   // Same for all the cells
{
        if((SW2==0))
        {
        CFG1R1 |= 4;
        SW3 = 1;
        }
}

if(Cell4>Cell5)
{
        if((SW3==0))
        {
        CFG1R1 |= 8;
        SW4 = 1;
        }
}

if(Cell5>Cell6)
{
        if((SW4==0))
        {
        CFG1R1 |= 16;
        SW5 = 1;
        }
}

if(Cell6>Cell7)
{
        if((SW5==0))
        {
        CFG1R1 |= 32;
        SW6 = 1;
        }
}

if(Cell7>Cell8)l
{
        if((SW6==0))
        {
        CFG1R1 |= 64;
        SW7 = 1;
        }
}

if(Cell8>Cell9)
{
        if((SW7==0))
        {
        CFG1R1 |= 128;
        SW8 = 1;
        }
}

if(Cell9>Cell10)l
{
        if((SW8==0))
        {
        CFG1R2 |= 1;
        SW9 = 1;
        }
}
```

```
        if(Cell10>Cell11)
        {
                if((SW9==0))
                {
                CFG1R2 |= 2;
                SW10 = 1;
                }
        }

        if(Cell11>Cell12)
        {
                if((SW10==0))
                {
                CFG1R2 |= 4;
                SW11 = 1;
                }
        }

        if(Cell12>Cell1)
        {
                if((SW11==0))
                {
                CFG1R0 |= 8;
                SW12 = 1;
                }
        }

//      TOP LTC
        if(Cell21>Cell22)
        {
                if((SW12==0))
                {
                CFG2R1 |= 1;
                SW13 = 1;
                }
        }

        if(Cell22>Cell23)
        {
                if((SW13==0))
                {
                CFG2R1 |= 2;
                SW14 = 1;
                }
        }

        if(Cell23>Cell24)
        {
                if((SW14==0)) f
                {
                CFG2R1 |= 4;
                SW15 = 1;
                }
        }

        if(Cell24>Cell25)
        {
                if((SW15==0))
                {
                CFG2R1 |= 8;
                SW16 = 1;
                }
        }
```

```
if(Cell25>Cell26)l
{
        if((SW16==0)) f
        {
        CFG2R1 |= 16;
        SW17 = 1;
        }
}

if(Cell26>Cell27)
{
        if((SW17==0))
        {
        CFG2R1 |= 32;
        SW18 = 1;
        }
}

if(Cell27>Cell28)
{
        if((SW18==0))f
        {
        CFG2R1 |= 64;
        SW19 = 1;
        }
}

if(Cell28>Cell29)
{
        if((SW19==0))
        {
        CFG2R1 |= 128;
        SW20 = 1;
        }
}

if(Cell29>Cell210)
{
        if((SW20==0))
        {
        CFG2R2 |= 1;
        SW21 = 1;
        }
}

if(Cell210>Cell211)
{
        if((SW21==0))
        {
        CFG2R2 |= 2;
        SW22 = 1;
        }
}

if(Cell211>Cell212)
{
        if((SW22==0))
        {
        CFG2R2 |= 4
        SW23 = 1;
        }
}

if(Cell212>Cell21)
{
        if((SW23==0))
        {
        CFG2R2 |= 8;
        SW24 = 1;
        }
}
```

```
// START Send New Config                                    // Send discharge switches to LTC Chip

//        CHIP HIGH
          PORTB &= ~(1<<SS);                                // Pull slave select low

          SPI_MasterTransmit(WRCFG);                        // Send command address
                                                            // Send config registers, they are the
                                                            // different for all the LTC Chips
          SPI_MasterTransmit(CFG1R0);                       // Send config register 0
          SPI_MasterTransmit(CFG1R1);                       // Send config register 1
          SPI_MasterTransmit(CFG1R2);                       // Send config register 2
          SPI_MasterTransmit(CFG1R3);                       // Send config register 3
          SPI_MasterTransmit(CFG1R4);                       // Send config register 4
          SPI_MasterTransmit(CFG1R5);                       // Send config register 5

//        CHIP LOW
          SPI_MasterTransmit(CFG2R0);                       // Send config register 0
          SPI_MasterTransmit(CFG2R1);                       // Send config register 1
          SPI_MasterTransmit(CFG2R2);                       // Send config register 2
          SPI_MasterTransmit(CFG2R3);                       // Send config register 3
          SPI_MasterTransmit(CFG2R4);                       // Send config register 4
          SPI_MasterTransmit(CFG2R5);                       // Send config register 5

          PORTB |= (1<<SS);                                 // Pull slave select high

// END Send New Config


// START Temperature Calculations

//        CHIP LOW
//        External Temp 1
          Temp1 = TMPR0;                                    // Move 8 bits to variable
          Temp2 = TMPR1;                                    // Move 8 bits to variable
          Temp3 = (Temp2 & 0x0F);                           // Mask the 4 MSBs
          TEMPEX1 = ( Temp3 << 8 ) + Temp1;

//        External Temp 2
          Temp1 = TMPR1;
          Temp2 = TMPR2;
          Temp3 = (Temp1 & 0xF0);
          TEMPEX2 = ( Temp2 << 8 ) + Temp3;
          TEMPEX2 = (TEMPEX2 >> 4);

//        Internal Temp
          Temp1 = TMPR3;                                    // Move 8 bits to variable
          Temp2 = TMPR4;                                    // Move 8 bits to variable
          Temp3 = (Temp2 & 0x0F);                           // Mask the 4 MSBs
          TEMPINT = ( Temp3 << 8 ) + Temp1;

//        CHIP HIGH
//        External Temp 1
          Temp1 = TMPR20;                                   // Move 8 bits to variable
          Temp2 = TMPR21;                                   // Move 8 bits to variable
          Temp3 = (Temp2 & 0x0F);                           / Mask the 4 MSBs
          TEMPEX21 = ( Temp3 << 8 ) + Temp1;

//        External Temp 2
          Temp1 = TMPR21;                                   // Move 8 bits to variable
          Temp2 = TMPR22;                                   // Move 8 bits to variable
          Temp3 = (Temp1 & 0xF0);                           // Mask the 4 LSBs
          TEMPEX22 = ( Temp2 << 8 ) + Temp3;
          TEMPEX22 = (TEMPEX22 >> 4);

//        Internal Temp
          Temp1 = TMPR23;                                   // Move 8 bits to variable
          Temp2 = TMPR24;                                   // Move 8 bits to variable
          Temp3 = (Temp2 & 0x0F);                           / Mask the 4 MSBs
          TEMPINT2 = ( Temp3 << 8 ) + Temp1;
```

```
// END Calculations

// START USART Communications

        int num=0;                                          // Variable for binary 2 ascii converter

        USART_Int();                                        // Initialise the USART for Asynchronous
                                                            // operation
        USART_Enable();                                     // Enable transmit and receive

//      CHIP LOW
//      START CONDITION
        USART_TransmitString("B");

//      CELL VOLTAGE START
        num = Cell1;                                        // Give num the value of Cell1
        if (num<0x3E8)                                      // if num < 1000
        {
        USART_TransmitString("0");                          // Add a 0 and transmit string
        USART_TransmitString(itoa(num, buf, 10));
        }
         else
        {
        USART_TransmitString(itoa(num, buf, 10));           // Otherwise just transmit string
        }

        num = Cell2;                                        // Same for all 24 cells
        if (num<0x3E8)
        {
        USART_TransmitString("0");
        USART_TransmitString(itoa(num, buf, 10));
        }
         else
        {
        USART_TransmitString(itoa(num, buf, 10));
        }

        num = Cell3;
        if (num<0x3E8)
        {
        USART_TransmitString("0");
        USART_TransmitString(itoa(num, buf, 10));
        }
         else
        {
        USART_TransmitString(itoa(num, buf, 10));
        }

        num = Cell4;
        if (num<0x3E8)
        {
        USART_TransmitString("0");
        USART_TransmitString(itoa(num, buf, 10));
        }
         else
        {
        USART_TransmitString(itoa(num, buf, 10));
        }

        num = Cell5;
        if (num<0x3E8)
        {
        USART_TransmitString("0");
        USART_TransmitString(itoa(num, buf, 10));
        }
         else
        {
        USART_TransmitString(itoa(num, buf, 10));
        }
```

```
num = Cell6;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell7;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell8;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell9;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell10;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell11;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell12;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
```

```
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell21;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell22;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell23;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell24;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell25;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell26;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}
```

```
num = Cell27;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell28;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell29;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell210;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell211;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}

num = Cell212;
if (num<0x3E8)
{
USART_TransmitString("0");
USART_TransmitString(itoa(num, buf, 10));
}
 else
{
USART_TransmitString(itoa(num, buf, 10));
}
//          CELL VOLTAGE END
```

```
//          INTERNAL TEMPERATURE
            num = TEMPINT;
            USART_TransmitString(itoa(num, buf, 10));

            num = TEMPINT2;
            USART_TransmitString(itoa(num, buf, 10));


//          OVER AND UNDERVOLTAGE

            num = CFG1R4;
            USART_TransmitString(itoa(num, buf, 10));

            num = CFG1R5;
            USART_TransmitString(itoa(num, buf, 10));

//          END CONDITION
            USART_TransmitString("E");
            USART_TransmitString("*");
            USART_TransmitString("*");

            USART_Disable();

// END USART Communications

// START Sleep and Low Power Mode

            PORTB |= (1<<MOSI)|(1<<SCK)|(1<<SS);                    // All pins high

            STATUS = (1<<STOP);                                     // Enable red LED to show end condition

            set_sleep_mode(SLEEP_MODE_PWR_DOWN);                    // Select power down mode
            sleep_mode();                                           // Activate sleep mode and wait for
                                                                    // reset from watch dog timer

}

//=============== MAIN PROGRAM END ==================
```

12.4.2 Visual Basic Code

```
Dim a As Integer                                                    'Create integer A
Dim Data As String                                                  'Create string DATA
-------------------------------------------------------------------------------------------------------------------------------------------
Private Sub MSComm1_OnComm()                                        'START OF PROGRAM
If Worksheets("Display").MSComm1.CommEvent = comEvReceive Then      'If data received then
    Worksheets("Display").MSComm1.InputLen = 0                      'Data of unknown length
    Data = Worksheets("Display").MSComm1.Input                      'Assign the input to DATA

  If Mid$(Data, 1, 1) = "B" Then            'check Header           'Check start condition

    If Mid$(Data, 112, 1) = "E" Then        'check Tailer           'Check end condition

        TextBoxCell1 = Round((Mid$(Data, 2, 4) * 0.0015), 3)        'Give TextBoxCell1 the value of Cell 1
        TextBoxCell2 = Round((Mid$(Data, 6, 4) * 0.0015), 3)        'Goes on till all 24 Cells are displayed
        TextBoxCell3 = Round((Mid$(Data, 10, 4) * 0.0015), 3)
        TextBoxCell4 = Round((Mid$(Data, 14, 4) * 0.0015), 3)
        TextBoxCell5 = Round((Mid$(Data, 18, 4) * 0.0015), 3)
        TextBoxCell6 = Round((Mid$(Data, 22, 4) * 0.0015), 3)
        TextBoxCell7 = Round((Mid$(Data, 26, 4) * 0.0015), 3)
        TextBoxCell8 = Round((Mid$(Data, 30, 4) * 0.0015), 3)
        TextBoxCell9 = Round((Mid$(Data, 34, 4) * 0.0015), 3)
        TextBoxCell10 = Round((Mid$(Data, 38, 4) * 0.0015), 3)
        TextBoxCell11 = Round((Mid$(Data, 42, 4) * 0.0015), 3)
        TextBoxCell12 = Round((Mid$(Data, 46, 4) * 0.0015), 3)
        TextBoxCell13 = Round((Mid$(Data, 50, 4) * 0.0015), 3)
        TextBoxCell14 = Round((Mid$(Data, 54, 4) * 0.0015), 3)
        TextBoxCell15 = Round((Mid$(Data, 58, 4) * 0.0015), 3)
        TextBoxCell16 = Round((Mid$(Data, 62, 4) * 0.0015), 3)
        TextBoxCell17 = Round((Mid$(Data, 66, 4) * 0.0015), 3)
        TextBoxCell18 = Round((Mid$(Data, 70, 4) * 0.0015), 3)
        TextBoxCell19 = Round((Mid$(Data, 74, 4) * 0.0015), 3)
        TextBoxCell20 = Round((Mid$(Data, 78, 4) * 0.0015), 3)
        TextBoxCell21 = Round((Mid$(Data, 82, 4) * 0.0015), 3)
```

```
        TextBoxCell22 = Round((Mid$(Data, 86, 4) * 0.0015), 3)

        TextBoxCell23 = Round((Mid$(Data, 90, 4) * 0.0015), 3)

        TextBoxCell24 = Round((Mid$(Data, 94, 4) * 0.0015), 3)


        TextBoxTemp = ((Mid$(Data, 98, 4) * 1.5 / 8) - 273.15)                  'Give TextBoxTemp the value Temp1

        TextBoxTemp2 = ((Mid$(Data, 102, 4) * 1.5 / 8) - 273.15)                'Give TextBoxTemp the value Temp2


        TextBoxUVL = ((Mid$(Data, 106, 3) * 0.0015 * 16))                       'Give TextBoxUVL the UV level

        TextBoxOVL = ((Mid$(Data, 109, 3) * 0.0015 * 16))                       'Give TextBoxOVL the OV level


        TextBoxStack = ((Mid$(Data, 2, 4) * 0.0015) + (Mid$(Data, 6, 4) * 0.0015) + (Mid$(Data, 10, 4) * 0.0015) + (Mid$(Data, 14, 4) *
0.0015) + (Mid$(Data, 18, 4) * 0.0015) + (Mid$(Data, 22, 4) * 0.0015) + (Mid$(Data, 26, 4) * 0.0015) + (Mid$(Data, 30, 4) * 0.0015) +
(Mid$(Data, 34, 4) * 0.0015) + (Mid$(Data, 38, 4) * 0.0015) + (Mid$(Data, 42, 4) * 0.0015) + (Mid$(Data, 46, 4) * 0.0015))

                                                                                'Adds the first 12 Cells values

        TextBoxStack = TextBoxStack + ((Mid$(Data, 50, 4) * 0.0015) + (Mid$(Data, 54, 4) * 0.0015) + (Mid$(Data, 58, 4) * 0.0015) +
(Mid$(Data, 62, 4) * 0.0015) + (Mid$(Data, 66, 4) * 0.0015) + (Mid$(Data, 70, 4) * 0.0015) + (Mid$(Data, 74, 4) * 0.0015) + (Mid$(Data, 78,
4) * 0.0015) + (Mid$(Data, 82, 4) * 0.0015) + (Mid$(Data, 86, 4) * 0.0015) + (Mid$(Data, 90, 4) * 0.0015) + (Mid$(Data, 94, 4) * 0.0015))

                                                                                'Adds the second 12 Cells values tot
                                                                                the 'first and displays it in
                                                                                TextBoxStack


    End If


End If


If ToggleButton1 = True Then                                                    'If LOG DATA button is pushed

                                                                                'then do the following

    a = a + 1                                                                   'Increment A by one every time

    Sheet2.Cells(a, 1) = TextBoxCell1.Value                                     'Display Cell 1's value in column 1

    Sheet2.Cells(a, 2) = TextBoxCell2.Value                                     'Do for all 24 Cells

    Sheet2.Cells(a, 3) = TextBoxCell3.Value

    Sheet2.Cells(a, 4) = TextBoxCell4.Value

    Sheet2.Cells(a, 5) = TextBoxCell5.Value

    Sheet2.Cells(a, 6) = TextBoxCell6.Value

    Sheet2.Cells(a, 7) = TextBoxCell7.Value

    Sheet2.Cells(a, 8) = TextBoxCell8.Value

    Sheet2.Cells(a, 9) = TextBoxCell9.Value

    Sheet2.Cells(a, 10) = TextBoxCell10.Value
```

```
      Sheet2.Cells(a, 11) = TextBoxCell11.Value
      Sheet2.Cells(a, 12) = TextBoxCell12.Value
      Sheet2.Cells(a, 13) = TextBoxCell13.Value
      Sheet2.Cells(a, 14) = TextBoxCell14.Value
      Sheet2.Cells(a, 15) = TextBoxCell14.Value
      Sheet2.Cells(a, 16) = TextBoxCell16.Value
      Sheet2.Cells(a, 17) = TextBoxCell17.Value
      Sheet2.Cells(a, 18) = TextBoxCell18.Value
      Sheet2.Cells(a, 19) = TextBoxCell19.Value
      Sheet2.Cells(a, 20) = TextBoxCell20.Value
      Sheet2.Cells(a, 21) = TextBoxCell21.Value
      Sheet2.Cells(a, 22) = TextBoxCell22.Value
      Sheet2.Cells(a, 23) = TextBoxCell23.Value
      Sheet2.Cells(a, 24) = TextBoxCell24.Value

   End If

End If

End Sub
```

```
Private Sub CommandButtonOpenPort_Click()                    'If START MEASUREMENT button is
                                                             'pushed
a = 1                                                        'Equate A to 1
If Worksheets("Display").MSComm1.PortOpen = True Then        'Check Comm Port status
   TextBoxError = "Com1 Already Opened"                      'If open, then display
Else
TextBoxError = "Com1 Open"                                   'Else, display

 If Worksheets("Display").MSComm1.PortOpen = False Then      'If not open then do following

   Worksheets("Display").MSComm1.Settings = "38400,N,8,1"    'Set Com. Port settings
   Worksheets("Display").MSComm1.RThreshold = 112            'Set threshold
   Worksheets("Display").MSComm1.InBufferSize = 4096         'Set max buffer size
   Worksheets("Display").MSComm1.PortOpen = True             'Open Com Port
```

```
End If
End If
Application.EnableEvents = True                    'Enable events

End Sub
Private Sub CommandButtonClosePort_Click()         'If STOP MEASUREMENT button is
                                                   'pushed
  a = 1                                            'Equate A to 1
  If Worksheets("Display").MSComm1.PortOpen = False Then   'Check Comm Port status
    TextBoxError = "Com1 Already Closed"           'If closed, then display
  Else
    TextBoxError = "Com1 Close"                    'Else display
    Worksheets("Display").MSComm1.PortOpen = False 'Close Comm Port
  End If
End Sub
Private Sub CommandButtonCLRDATA_Click()           'If CLEAR button is pushed

  MSComm1.PortOpen = False                         'Close Comm Port
  Sheet2.Cells = Clear                             'Clear all cells on Sheet2
  a = 1                                            'Equate A to 1
  MSComm1.PortOpen = True                          'Open Comm Port

End Sub                                            'END OF PROGRAM
```