

University of Manitoba
Department of Electrical & Computer Engineering

ECE 4600 Group Design Project

Lithium-Ion Battery Management System for FSAE Electric Vehicles

by
Group 05

Zaw Aungkyaw

Houman Lahiji

Matthew Riesmeyer

Final report submitted in partial satisfaction of the requirements for the degree of
Bachelor of Science in Electrical and Computer Engineering in the **Faculty of Engineering** at
the **University of Manitoba**

Academic Supervisor

Dr. Robert McLeod

Industry Supervisor

Tyler Listoe - Iders Incorporated

Date of Submission

March 2016

© Copyright by Zaw Aungkyaw, Houman Lahiji, Matthew Riesmeyer, 2016

Abstract

This report describes the design and evaluation of a Battery Management System (BMS) that is compliant with the standards set out for the Formula SAE Electric (FSAEE) competition organized by SAE International. The motivation for this project originates from the lack of commercially available battery management systems designed specifically for application in electric open-wheel racecars. Commercially available battery management systems are inadequate in that they are bulky, over-engineered and expensive, or non-customizable with the high-energy batteries used in FSAEE vehicles. To reduce cost and provide customizability, the BMS described in this project can be programmed for different battery parameters and has a scalable management capacity from 3 to 192 cells.

The performance of the BMS was evaluated by monitoring a stack of six lithium-ion cells and two power supplies. The power supplies imitated an additional six cells each, while allowing the simulation of undervoltage and overvoltage fault conditions. The BMS monitored voltages from 0.465 V to 6.014 V, with an accuracy and reading time of 1.03 mV and 31.81 μ s per cell, respectively. These results were verified by reproduction of test results, with 100% fault detection, using 25 different programmed variations of voltage and temperature limits. Finally, the BMS successfully measured temperatures ranging from 0 °C to +65 °C. These test results demonstrated that the BMS had met and exceeded all of the required specifications.

Contributions

The project workload was carried out by the group members as listed in the table below.

	Zaw A.	Houman L.	Matthew R.
Battery Monitoring Sub-system	○	○	○
Communication & Annunciator Sub-system	●		○
Cell Balancing Sub-system	●	○	
Printed Circuit Board Design		●	○
System Coding	●		○
Final Report	○	○	●

Legend: ● Lead ○ Contributor

Acknowledgements

The success of this project was only possible thanks to the support of the Faculty of Engineering, Iders Incorporated, and the University of Manitoba Chapter of SAE International. The group would like to thank all of the individuals who contributed to the project. Support was provided to the group in many forms, including but not limited to technical expertise, in-kind contributions, and test equipment.

First of all, we would like to thank all of the individuals from Iders Incorporated, without whom our project would not have been possible. We are grateful to Mr. Brad Brown, the president of Iders Inc., for providing us the opportunity to collaborate and benefit from industry expertise, state-of-the-art equipment, and in-kind contributions. In addition, we thank Mr. Tyler Listoe, our industry supervisor, for all of his time, especially his time reviewing our designs and providing us with impactful feedback. Furthermore, we are grateful to Mr. Brett Trombo for coordinating the expedited population of our PCBs.

We would also like to thank the members of the Electrical and Computer Engineering Department, without whom this project would not have been possible. Specifically, thank you to Dr. Robert Mcleod and Dr. Ahmad Byagowi, our academic supervisors, for their guidance and support. Their assistance and expertise was immeasurable. Mr. Daniel Card, resident advisor, also provided invaluable advice and suggestions for our designs on numerous occasions. Mr. Glen Kolansky, Mr. Zoran Trajkoski, and Mr. Sinisa Janjic from the University of Manitoba's Electrical & Computer Engineering Tech Shop were also instrumental to the success of this project and we are grateful for their help.

In addition, we would also like to thank Dr. Derek Oliver for coordinating the ECE 4600 course and Ms. Aiden Topping for her guidance and feedback with both the written and oral reports. Thank you to the University of Manitoba for the financial contributions for this project.

Special thanks is given to the University of Manitoba Chapter of SAE International for providing us with the motivation for the project. Additionally, thank you for lending us various materials, lithium-ion batteries, and test equipment, and for the use of tools and facilities.

Finally, we would like to thank our families and friends for their support throughout the completion of the project, our development as future engineers, as well as throughout our lives.

Table of Contents

Contents

Abstract.....	1
Contributions	2
Acknowledgements.....	3
Table of Contents	4
List of Figures	6
List of Tables	7
Nomenclature.....	8
1. Introduction.....	9
1.1. Motivation.....	9
1.2. Project Scope.....	9
1.3. System Breakdown.....	10
2. Battery Monitoring.....	12
2.1. Design Criteria	12
2.2. Sub-System Design	12
2.2.1. Protection Circuit	13
2.2.2. Voltage Monitoring	14
2.2.3. Temperature Monitoring	14
3. Communications and Annunciator	16
3.1. Design Criteria	16
3.2. Host-to-Device and Device-to-Device Interface.....	17
3.3. LCD Shield Interface	18
4. Cell Balancing.....	19
4.1. Background Theory.....	19

4.2.	Design Criteria	20
4.3.	Sub-System Design	20
5.	System Coding	22
5.1.	Initialization	22
5.2.	Communication Protocols	23
5.3.	Battery Monitoring / Fault Detection	26
5.4.	Cell Balancing.....	28
6.	Printed Circuit Board Module	30
6.1.	PCB Layout.....	31
6.2.	Cell Balancing Circuit Considerations	32
6.3.	Isolation Circuit and Power Regulator	32
6.4.	Vertical Bus Configuration	32
7.	System Integration and Evaluation.....	34
	PCB Fabrication and Population	34
7.1.	Voltage Regulator	34
7.2.	System Evaluation.....	35
7.2.1.	Reading Time, Accuracy, and Fault Detection Tests	36
7.2.2.	Measurement Range Tests.....	40
7.2.3.	Cell Balancing Test	42
7.3.	Integration Summary	43
8.	Conclusion	45
8.1.	Recommendations for System Improvement.....	45
	References.....	47
	Appendix A: FSAE BMS Requirements	48
	Appendix B: Budget Summary	49
	Appendix C: Printed Circuit Board Design	51
	Appendix D: Software Flowcharts	55
	Appendix E: Project Code.....	61

Appendix F: Test Results	82
Appendix G: Curriculum Vitae	89
Appendix References	90

List of Figures

Figure 1.1: Block diagram of the BMS.	11
Figure 2.1: Transient voltage and inrush current protection.	13
Figure 2.2: Voltage monitoring schematic.	14
Figure 2.3: The external temperature monitoring circuit.	15
Figure 3.1: Block diagram of the communication and annunciator subsystem.....	16
Figure 3.2: Device-to-device communication lines.	17
Figure 3.3: Device-to-host communication lines.....	18
Figure 4.1: Cell balancing circuit schematic.	21
Figure 5.1: Packet formatting. (a) Write packet detail. (b) Read packet detail.	25
Figure 5.2: Program digital analyzer output. (a) Sleep packet. (b) Duration of main program before sleep cycle. (c) Duration of sleep cycle.....	28
Figure 6.1: PCB layout and different sub-circuits.	30
Figure 6.2: High voltage and low voltage ground planes.	31
Figure 6.3: Example of the North Vertical Bus of a board.	33
Figure 7.1: System diagram of the BMS.	35
Figure 7.2: The test setup for full system evaluation.	36
Figure 7.3: Logic analyzer output displaying DRDY duration.	37
Figure 7.4: Test #10. (a) Prior to UV trip. (b) Successful UV detection.	38
Figure 7.5: Test #14, showing successful OV detection.....	39
Figure 7.6: Test #22. (a) Prior to OT trip. (b) Successful OT detection.	40
Figure 7.7: The minimum operating voltage of the BMS.....	41
Figure 7.8: The maximum tested operating voltage of the BMS.	41
Figure 7.9: The minimum tested temperature reading of the BMS.	42
Figure 7.10: Cell voltages as a function of time for cell balancing test.	43
Figure C.1: Schematic of the PCB.....	51
Figure C.2: Top Layer of the PCB.....	52
Figure C.3: Ground Layer of the PCB.....	52

Figure C.4: Ground Layer of the PCB.....	53
Figure C.5: Bottom Layer of the PCB.....	53
Figure C.6: Voltage regulator schematic.....	54
Figure C.7: Voltage Regulator PCB layout.....	54
Figure D.1: Main flowchart.....	55
Figure D.2: Flowchart for automatic addressing of bq76PL536 devices.....	56
Figure D.3: Flowchart for cell balancing.....	57
Figure D.4: Flowchart for interfacing with the pushbuttons and LCD.....	58
Figure D.5: Flowchart for interfacing with the LCD to display temperature and voltage values..	59
Figure D.6: Flowchart for interfacing with the LCD to display alerts and faults.....	60

List of Tables

Table 1.1: Proposed specification of the project BMS.	10
Table 2.1: Battery monitoring specifications summary.	12
Table 5.1: Communication connections.	24
Table 7.1: BMS proposed specifications compared to achieved performance.....	43
Table B.1: Project budget summary.....	49
Table F.1: Cell voltages in comparison to voltages displayed by the BMS LCD.....	82
Table F.2: Fault detection and DRDY high-low-high time for different system settings.....	85
Table F.3: Cell voltages as a function of time to test cell balancing.....	87

Nomenclature

List of abbreviations and used throughout the report.

Acronym/Truncation	Expanded Form
AC	Alternating Current
ADC	Analog-to-Digital Converter
BMS	Battery Management System
D2D	Device-to-Device
D2H	Device-to-Host
EPROM	Erasable Programmable Read-Only Memory
ESR	Equivalent Series Resistance
EV	Electric Vehicle
FE	Formula Electric
FSAE	Formula SAE
HV	High Voltage
IC	Integrated Circuit
I ² C	Inter-Integrated Circuit
LCD	Liquid Crystal Display
LV	Low Voltage
OT	Over Temperature
OV	Over Voltage
PCB	Printed Circuit Board
SPI	Serial Peripheral Interface
TI	Texas Instruments
UMSAE	University of Manitoba Chapter of SAE International
UV	UnderVoltage

Chapter 1

1. Introduction

This report describes the design and evaluation of a **battery management system (BMS)** that is compliant for the **Formula SAE Electric (FSAEE)** competition organized by SAE International.

1.1. Motivation

Growing concern over carbon emissions and the impending decline of fossil fuel availability have led to increased interest in electric drive vehicles. This interest has sparked SAE International to facilitate an annual student design competition aimed at developing efficient, high-performance **electric vehicle (EV)** technology called FSAEE. Locally, the **University of Manitoba Chapter of SAE International (UMSAE)** is an active participant in the Formula SAE Electric competition.

Many FSAEE competition teams rely on batteries composed of lithium-ion cells to power their vehicles. Lithium-ion cells provide a high energy density to weight ratio, but are dangerous if mistreated, so must be monitored using a BMS [1]. The primary role of a BMS is to protect both the cells and the system the cells are powering by preventing the cells from operating outside of safe voltage and temperature conditions [2]. Like other FSAEE teams, the UMSAE **Formula Electric (FE)** team presently uses a commercially available BMS for its open-wheeled all-electric race car. However there are two problems associated with commercially available BMS: high costs and/or non-customizability. These two problems can be alleviated by using an elemental, modular, and programmable BMS, to provide an affordable system with a scalable management capacity that allows for inexpensive component replacement.

1.2. Project Scope

The BMS developed in this project is a proof-of-concept system that is compliant with the FSAEE competition requirements and can be programmed to service a number of cell specifications. A list of the proposed features that were to be delivered by the project BMS, including the rule specifications required for competition, are summarized in Table 1.1. See Appendix A for the full list of applicable rules. The features that are not required by the FSAE competition were selected to provide customizability to the end user.

Table 1.1: Proposed specification of the project BMS.

Specification	Proposed	FSAEE Requirement
Voltage Measurement Range (per cell)	0 V to +6 V	
Voltage accuracy @ 3.6V, 25 °C	50 mV	
Voltage reading time / cell	100 μ s to 250 μ s	Yes
Voltage Sensors	1 per 1 cell	Yes
Temperature Measurement Range	-40 °C to +60 °C	Yes
Temperature Sensors	1 per 3 cells	Yes
Programmable Under-Voltage and Overvoltage	Yes	
Programmable Maximum Operating Temperature	Yes	
Fault Indication	Red LED	Yes
Critical Operating Voltage Level Fault Detection	Yes	Yes
Critical Operating Temperatures Fault Detection	Yes	Yes
Management Capacity	6 cells in series	

During the development of the BMS, the design group decided to expand beyond the proposed project scope to provide additional system functionality. The additional features are: (I) implementing scalable management capacity, (II) implementing a cell balancing function, (III) the addition of a **Liquid Crystal Display (LCD)** human-machine interface, and (IV) implementing basic system functionality error detection. The resulting BMS is fully programmable and can be scaled to manage a range from 3 to 192 cells. In addition, the system can engage in cell balancing to balance the voltage of each cell, which improves the life of the cells. Effective implementation of the BMS will not only reduce the UMSAE FE's costs of initial purchase and component replacement, but also provide them a means of customizing the number of cells in their vehicle without acquiring a new BMS.

1.3. System Breakdown

The BMS consists of four sub-systems, as shown in Figure 1.1: (I) the **Printed Circuit Board (PCB)** module, (II) the battery monitoring sub-system, (III) the communication and annunciator sub-system, and (IV) the cell balancing sub-system. The cell balancing sub-system is made up of hardware within the battery monitoring sub-system, and software within the communication sub-system.

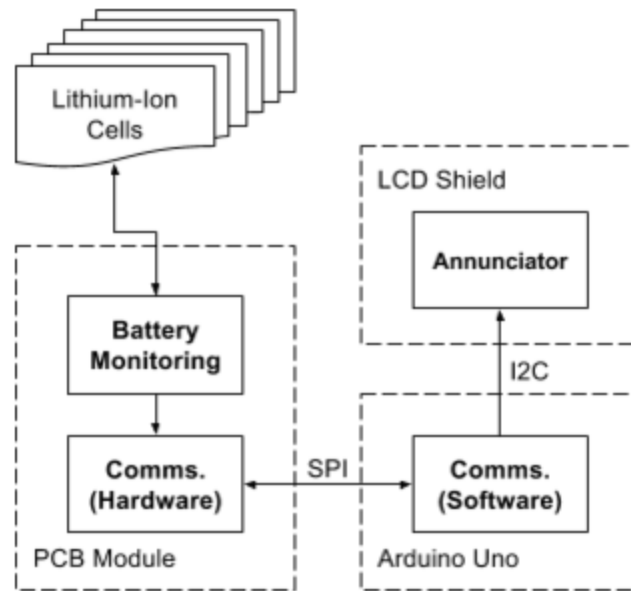


Figure 1.1: Block diagram of the BMS.

Sub-system I, the PCB module, contains the hardware for battery monitoring and communication. The software for the communication and annunciator sub-system is programmed using an Arduino Uno. Sub-systems II, III, and IV, were developed and tested independently of each other before being integrated together by means of the developed PCB modules. The design and implementation of each sub-system are discussed in the following chapters.

Chapter 2

2. Battery Monitoring

In EVs, the battery, or fuel cells, are one of the most expensive components of the vehicle. Due to their high cost, monitoring the cells becomes vital to make sure that they are operating in an efficient and non-destructive way. Lithium-ion cells are especially important to monitor as they may rapidly catch fire or explode if operating outside of safe conditions for a prolonged period of time. With safety as a large focus, the FSAEE competition requires that if a team is using lithium-ion cells, the voltage of every cell and the temperature of at least one per three cells is monitored.

2.1. Design Criteria

The objective of the battery monitoring sub-system is to meet the requirements of the FSAEE competition by monitoring cell voltage and temperature. As outlined in more detail in Appendix A, the voltage of the cells must be monitored accurately and continuously, which is open for interpretation by the group. To allow the BMS to be customized to a variety of lithium-ion cells, the BMS needs to monitor a voltage and temperature range from 0 V to 6 V. The specifications of the battery monitoring sub-system are summarized in Table 2.1.

Table 2.1: Battery monitoring specifications summary.

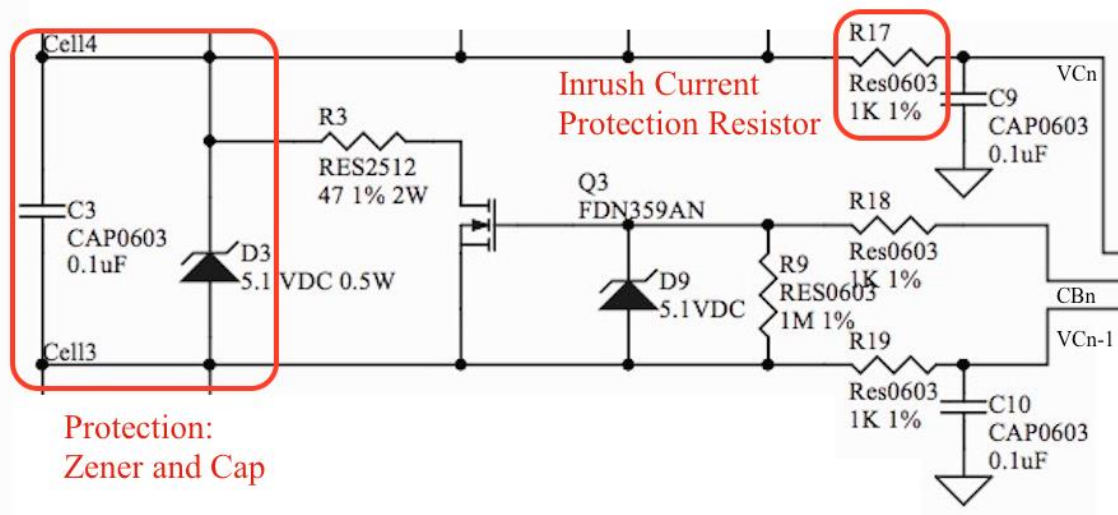
Specification	Target Level
Voltage Measurement Range (per cell)	0 V to +6 V
Voltage accuracy @ 3.6V, 25°C	50 mV
Voltage reading time / cell	100 μ s to 250 μ s
Voltage Sensors	1 per 1 cell
Temperature Measurement Range	-40 °C to +60 °C
Temperature Sensors	1 per 3 cells

2.2. Sub-System Design

The major component used in the BMS is the bq76PL536A-Q1, an **Integrated Circuit (IC)** developed by **Texas Instruments (TI)**. The selection of the bq76PL536A-Q1 was performed by comparing 38 different BMS ICs against the following criteria: (I) the IC must meet or exceed the FSAE requirements; (II) the IC must meet the battery specifications of the UMSAE FE vehicle; (III) the IC must use SPI communications; and (IV) the IC must have

reliable availability [3]. SPI communications was selected as a criterion because the design group has experience with SPI protocol. Each bq76PL536A-Q1 can monitor 3-6 cells with $\pm 5\text{mV}$ per cell voltage accuracy, measure temperature of 2 cells, and is capable of daisy chaining such that it can stack up to 32 ICs to monitor up to 192 cells with only one isolation circuit. It also provides overvoltage (OV) and undervoltage (UV) protection for the system and is capable of cell balancing. The bq76PL536A-Q1 met and exceeded all of the criteria set out by the group.

2.2.1. Protection Circuit



2.2.2. Voltage Monitoring

The bq76PL536A-Q1 features integrated ADCs, which use the bq76PL536A-Q1's internal reference voltage, pin Vref, to measure cell voltages. For a more accurate voltage reading, Nyquist filters with a cutoff frequency of $f_c = 1.6$ kHz are added to the ADC inputs. The circuit for voltage monitoring, without the aforementioned protection circuit, is shown in Figure 2.2.

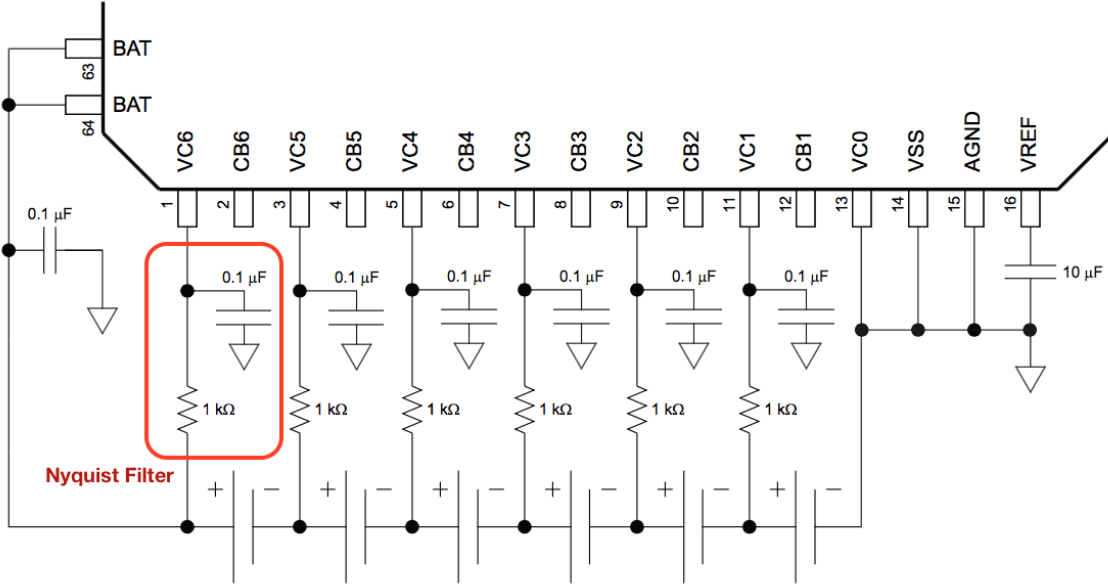


Figure 2.2: Voltage monitoring schematic.

The BMS designed in this report monitors the maximum capacity of six cells for each bq76PL536A-Q1, but if less than six cells must be monitored, the unconnected ADC pins need to be tied to the most positive cell of the stack. For instance, in case of a three cell monitoring configuration, the ADC pins VC4, VC5, and VC6 should be connected to VC3 using a 1 kΩ resistor.

2.2.3. Temperature Monitoring

Each bq76PL536A-Q1 is capable of managing two separate temperature sensors (thermistors) by measuring the voltage TS_n⁺ and TS_n[−] differential inputs using the ADC, where n represents temperature sensor 1 or 2. The TS_n inputs use external thermistor and resistor circuits driven by the output pin REG50. The generated results are ratiometric, which eliminates the need for REG50 voltage considerations. This configuration provides a 2.5 V full-scale input to the TS_n.

Figure 2.3 shows the external components used for the temperature monitoring. The thermistor used is rated at $10\text{ k}\Omega$ at $25\text{ }^{\circ}\text{C}$ with a negative temperature coefficient. The 47 nF bypass capacitor is used to reduce noise in the measurement system and resistors R_T and R_B are $1.47\text{ k}\Omega$ and $1.82\text{ k}\Omega$ respectively, as recommended by the datasheet [4].

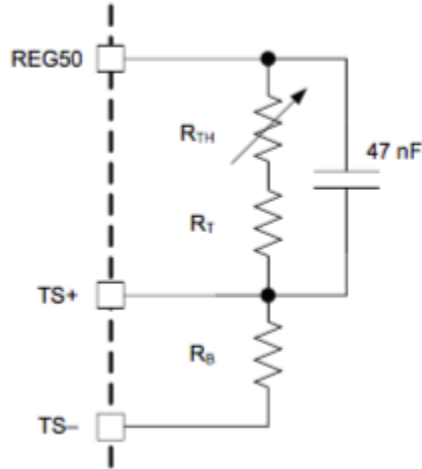


Figure 2.3: The external temperature monitoring circuit.

Chapter 3

3. Communications and Annunciator

The communication and annunciator sub-system is used to communicate between the bq76PL536A-Q1 (device) and the Arduino microcontroller (host), as well as between the Arduino and the LCD shield. **Device-to-host (D2H)** communications serves two purposes: (I) for the Arduino to send commands to the bq76PL536A-Q1, and (II) to transfer voltage and temperature information from the bq76PL536A-Q1 ICs to the Arduino. Likewise, communication between the Arduino and the LCD shield is bidirectional and serves two purposes: (I) for the Arduino to send the information to be displayed by the LCD, and (II) for the user to toggle the information being transferred by the Arduino and displayed on the LCD shield. Figure 3.1 shows a block diagram of the communication and annunciator sub-system. For the purposes of this report, this chapter focuses on hardware, while the affiliated code is discussed in Chapter 5.

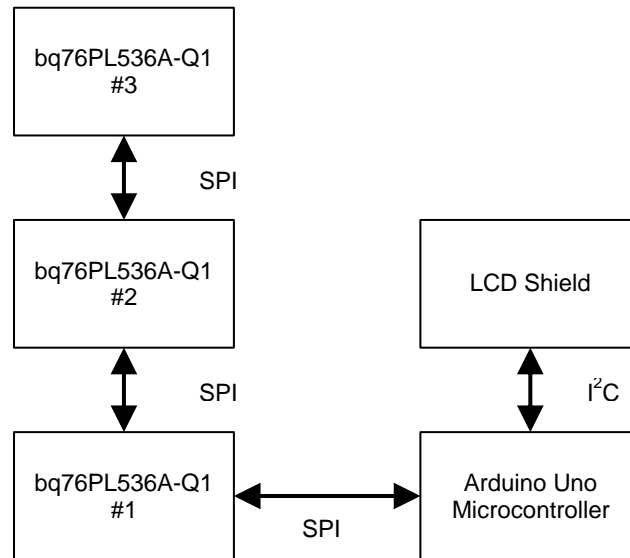


Figure 3.1: Block diagram of the communication and annunciator sub-system.

3.1. Design Criteria

The design of the communication and annunciator sub-system was largely open-ended, but did require that the H2D interface had two separate ground planes. This is because the ground plane on the device side is considered to have high potential, whereas the ground plane on the

host side is considered to have low potential. In addition, interfacing the Arduino with two separate peripheral devices requires the use of two separate communication protocols, **Serial Peripheral Interface (SPI)** and **Inter-Integrated Circuit (I²C)**, due to the limitations of the physical pins on the Arduino. These criteria were considered throughout the design of the communication and annunciator sub-system.

3.2. Host-to-Device and Device-to-Device Interface

All bq76PL536A-Q1 devices communicate to the Arduino microcontroller (device-to-host) and to one another (device-to-device) through the synchronous serial communication SPI protocol. As such, the D2H and **Device-to-Device (D2D)** communication protocol was dictated by the requirements of the bq76PL536A-Q1. This protocol is typically used for short-distance communication systems and is appropriate for the BMS application. The communication bus for D2H and D2D consists of eight lines. The SPI connections require four lines and the remaining four lines are used to directly interface with the bq76PL536A-Q1 devices for fault, alert, data ready, and conversion conditions. The fault and alert lines are asserted by the device when fault and alert conditions are detected. A feature of the bq76PL536A-Q1 is that isolation is not required on the D2D interface, allowing for a straight connection as shown in Figure 3.2. To isolate the device from the host, an isolation device, ISO7241, is used. Figure 3.3 shows the D2H communication interface.

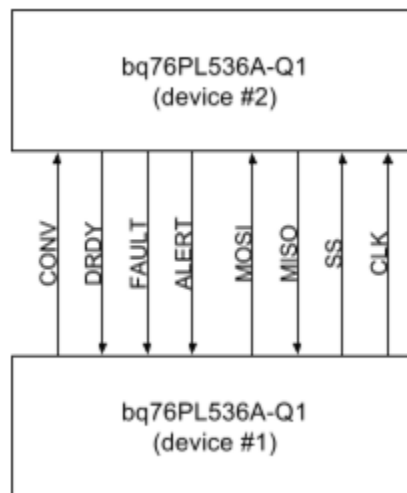


Figure 3.2: Device-to-device communication lines.

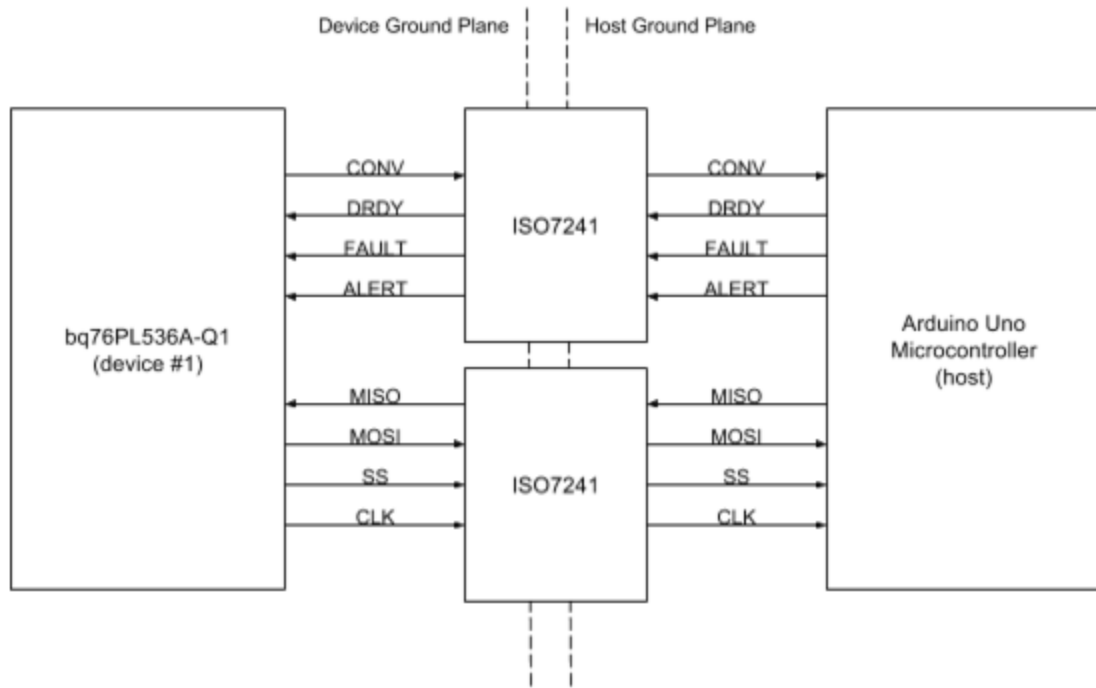


Figure 3.3: Device-to-host communication lines.

3.3. LCD Shield Interface

The Arduino microcontroller interfaces with the Adafruit LCD shield and accompanying five pushbuttons through I²C protocol. A “shield” is a developed module that can be plugged directly on top of an Arduino to extend its capabilities [5]. The Adafruit LCD shield was selected because its five pushbuttons and display allow for a dynamic range of user interfacing with ease of development and low costs. Although the LCH shield is plugged directly on top of the Arduino with every pin connected, only the power, ground, and I²C pins are used.

Chapter 4

4. Cell Balancing

A problem common to all battery cells when they are connected in series is that they may become unbalanced; i.e., the cell voltages are not always equal to the battery pack voltage divided by the number of cells [6]. It is important to maintain balanced cell voltages in any battery pack for reasons of safety, longevity, and efficiency. Lithium-ion cells may undergo thermal runaway if cell voltages exceed safe operating voltages by even a few hundred millivolts during a charge cycle of an unbalanced battery pack. This may lead to dangerous conditions that may result in destruction of the battery pack and the devices it is powering and could potentially cause harm to the user. If the charging voltage of an unbalanced cell is exceeded even a little, the degradation rate of the cell will be accelerated, leading to reduced longevity of the battery cell [6]. Furthermore, if the battery pack is unbalanced, cells that have a lower cell voltage may not be fully charged when the charge cycle is complete. The same cells with lower cell voltages will also reach undervoltage conditions earlier than the fully charged cells, resulting in a lower discharge capacity of the battery pack. A cell balancing feature was included in the BMS to improve battery cell efficiency and to promote healthy battery cells with increased longevity.

4.1. Background Theory

No two battery cells are manufactured identically [7]. Slight differences in cell characteristics can lead to divergence in the voltages over time. Battery cells may be unbalanced in multiple ways and can become unbalanced for multiple reasons. These causes include differences in the **state of charge (SOC)** of the cells, differences in the self-discharge rates, total capacity differences, and cell impedance differences. Unbalanced cells have a capacity limitation at one end when a cell is fully charged, and at the other end when a cell is fully discharged. Cells with lower capacity or higher internal impedance tend to have higher voltages when at full charge. This suggests that these cells will reach full SOC before the other cells in the battery pack, preventing the other cells from reaching full SOC due to the overvoltage protection circuitry. Conversely, these same cells tend to have lower voltages when discharged as a result of faster rates of discharge [7].

Cell balancing maximizes the charge that the battery pack can deliver by equalizing the voltage and SOC among the cells at some arbitrary SOC level. Since EVs are fully charged regularly, balancing is performed at the 100 % SOC level. This balancing reference point is

referred to as top balancing. Cell balancing may be achieved by a BMS by either removing extra charge from the most charged cells or by adding charge to the least charged cells through transference of energy from a higher SOC cell to a lower SOC cell. The former method is often referred to as passive balancing and the latter method is referred to as active balancing.

4.2. Design Criteria

The amount of time that is required to balance battery cells is dependent on two parameters: the amount of balancing required, and the balancing current. It is assumed that most FSAEE participants purchase battery packs that are balanced at the time of purchase. Therefore, the purpose of this design is to keep the battery cells balanced in regular maintenance intervals rather than provide a circuit that is expected to balance largely unbalanced battery packs in a reasonable time. The design specifically compensates for the self-discharge leakage variations in the cells. In a typical scenario, a self-discharge leakage rate of 2 % per month can be assumed for a lithium-ion battery pack stored at room temperature [8]. For the application of UMSAE's 20 Ah battery cells, a maximum variation of leakage rate between any two cells of 2 % would translate to 0.4 Ah discharged in a month. At roughly 720 hours in a month, the maximum leakage difference would be around 56 μ A between any two cells. If the recommended balancing current for the bq76PL536A-Q1 of around 80mA is implemented, it would take five hours of balancing per month to maintain battery pack balance. Maintenance intervals could be extrapolated from this rate for what would be convenient for the user of the BMS.

4.3. Sub-System Design

A passive cell balancing circuit is implemented in the BMS as it is more cost effective than an active circuit, does not require large capacitive or inductive charge storage devices, and is more than adequate for maintaining cell balancing against self-discharge leakages of the cells. The passive circuit discharges the higher SOC cells through a 47 Ω balancing resistor when cell balancing is active. Each bq76PL536A-Q1 IC is capable of providing a control signal [CB1 to CB6] to balance six battery cells. When active, CBn control outputs turn on a corresponding n-channel mosfet which in turn allows for a cell to be discharged through the balancing resistor. Cell balancing control logic and algorithms are programmed through the Arduino Uno microcontroller. Refer to Figure 4.1 for a schematic of the balancing circuit.

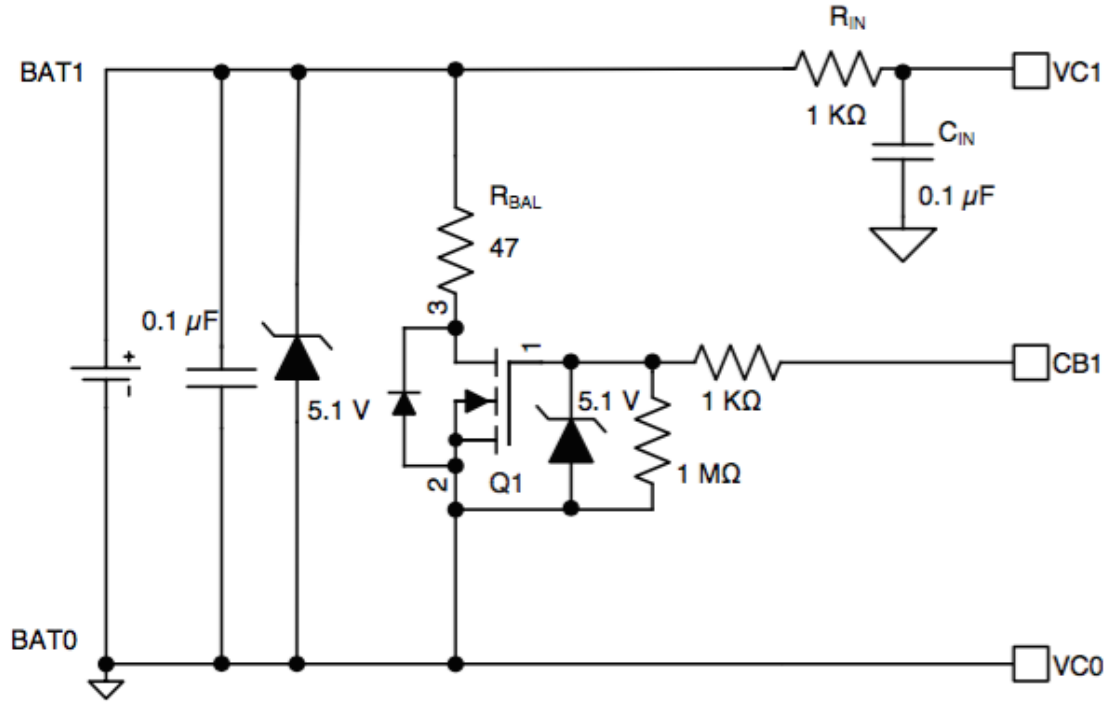


Figure 4.1: Cell balancing circuit schematic.

In addition to the main components used to perform cell balancing, secondary components are used in the circuit to provide circuit protection and noise rejection. Decoupling capacitors (0.1 μF) are placed strategically to reduce the amount of noise that could potentially cause faulty logic. A zener diode (5.1 V) is placed at the source of the mosfet for additional protection of the fet from high voltages. The 1 MΩ pull-down resistor ensures that the gate of the mosfet is turned off and does not accidentally float into an ON state. A 1 kΩ resistor is placed at the output of the CBn output to limit the current supplied to the gate of the mosfet during hot-plug or other high-transient events.

Chapter 5

5. System Coding

The system program is written using the Arduino programming language and is executed using an Arduino Uno microcontroller. The microcontroller is able to automatically address all bq76PL536A-Q1 battery monitoring devices to an upper limit of 32 devices. The microcontroller communicates with all bq76PL536A-Q1 devices using a **Serial Peripheral Interface (SPI)** bus and communicates with the LCD shield using **Inter-Integrated Circuit (I²C)** protocol. An 8-bit **Cyclic Redundancy Check (CRC)** error detection method is used to ensure that the correct data is transmitted. The user is able to define set points for battery protection parameters such as overvoltage, undervoltage, and over temperature through the program. The program is then able to detect fault conditions to protect the battery cells from reaching unsafe and failure conditions. Finally, the algorithms for cell balancing are also programmed and executed through the microcontroller. The latest version of the software for the BMS can be found in Appendix E.

5.1. Initialization

The program is capable of automatically addressing all bq76PL536A-Q1 devices used in the BMS by executing an automatic addressing sequence when the device is first powered on. The detailed flowchart of the automatic addressing sequence can be found in Appendix D, Figure D.2. All bq76PL536A-Q1 devices are addressed as 0x00 during manufacturing for device discovery during the automatic addressing routine. This routine searches for the first device in the vertical stack with the address 0x00 and incrementally addresses the devices starting with 0x01 for the base device up to 0x3E for the top device. Address 0x3F is reserved to broadcast messages to all devices at the same time in the stack. Once a device has been assigned an address, the program attempts to read the assigned address from the address register of the device to verify that it has been addressed correctly. If the addressing fails, an error message, “Addr Failed on Dn” is displayed on the LCD where, Dn signifies the device number (n) that the failure occurred on. The addressing routine is completed and returns to the main program once all devices have been addressed successfully.

The main program continues with its initialization sequence by defining all battery protection parameters as programmed by the user. The bq76PL536A-Q1 devices store all device configuration and functionality parameters in write-protected **Erasable Programmable Read-Only Memory (EPROM)** registers. The battery protection set points, such as overvoltage,

undervoltage, and overtemperature, are initialized from the EPROM. A bq76PL536PGM programming board is required to modify the EPROM but, due to budget limitations, the design group was not able to purchase the programming board. However, a workaround was implemented in the programming to temporarily modify and override the set points in these protected registers. A special sequential-write sequence is used to interact with the shadow control (SHDW_CTRL) register which allows temporary overwriting of the EPROM values until the next reset of the device. To initialize the battery protection set points which reflect the safe operating specifications of the battery cells used, the program first writes 0x35 to SHDW_CTRL register, followed immediately by the write command to program the set point to the desired register. The user may define the set points for overvoltage, undervoltage, and over temperature, which reflect their application in the defined variables COV, CUV, and OT, respectively.

5.2. Communication Protocols

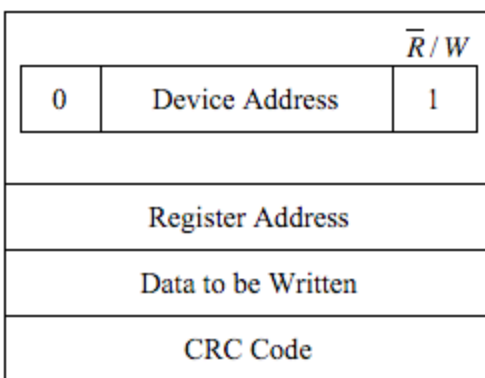
The Arduino Uno microcontroller interfaces with the LCD and accompanying pushbuttons through I²C protocol. The Arduino-based LCD device is accompanied by a library code supplied by the manufacturer. This library is referenced in the main program for ease of interfacing with the LCD and pushbuttons.

The SPI communication operates in voltage mode with standard TTL-compatible logic levels to allow for ease of connection with the bq76PL536A-Q1 devices [4]. The communication operates in SPI mode 1, where the SPI clock is normally low, the data changes on rising edges and is sampled on the falling edge, and the most significant bits are transferred first. The SPI communication clock speed is set at 3 MHz after practical testing showed that communication failures occur around 4 MHz for the BMS. The clock speed was scaled back to 3 MHz for communication reliability. The conversion line can be asserted by the microcontroller to initiate synchronous conversion, which communicates to the bq76PL536A-Q1 devices that a sampling of the voltages and temperatures is requested. The data-ready line is asserted by the device when conversion data is ready for reading. Table 5.1 shows the signal connections for D2D and D2H communications.

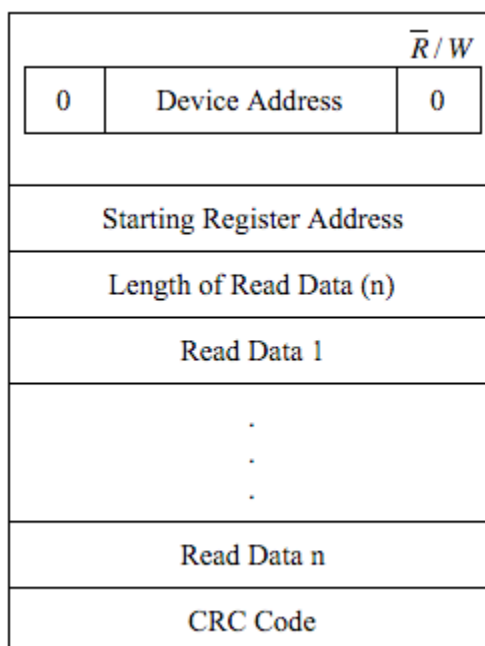
Table 5.1: Communication connections.

Connection Name	Microcontroller Pin Number: Pin Name	BMS PCB D2D Pin: D2H Pin
SPI: Slave Select	Pin 10: SS	P2/P3-8: P4-9
SPI: Master Output/Slave Input	Pin 11: MOSI	P2/P3-10: P4-7
SPI: Master Input/Slave Output	Pin 12: MISO	P2/P3-9: P4-8
SPI: Clock	Pin 13: SCK	P2/P3-7: P4-10
Fault	Pin 8: Fault_H	P2/P3-3: P4-2
Alert	Pin 7: Alert_H	P2/P3-1: P4-3
Data Ready (DRDY)	Pin 6: DRDY_H	P2/P3-2-: P4-4
Conversion (CONV)	Pin 5: CONV_H	P2/P3-4: P4-5

Reading from, and writing to, the bq76PL536A-Q1 devices is performed by formatting packets of information into a specific configuration. The data write packet is formed as shown in Figure 5.1(a). The first byte of information includes the device address as well as the last bit set to 1 to signify a write sequence. The second byte is the address of the register to be written to, followed by the third byte, which consists of the data to be written into the register. The last byte is the CRC code to ensure that the correct data is transmitted. The data write packet is always 4 bytes long, since only 1 byte of information can be written at a time. The data read packet is formed as shown in Figure 5.1(b). Similar to the write packet, the read packet consists of a first byte that includes the device address as well as a second byte with the register address. The third byte of the read packet signifies the length of the data to be read. The following bytes reflect the data that is read from the bq76PL536A-Q1 devices. The last byte of the read packet is the CRC code generated by the device.



(a)



(b)

Figure 5.1: Packet formatting. (a) Write packet detail. (b) Read packet detail.

An 8-bit CRC error-checking byte is calculated and appended to all SPI packets to ensure that the correct information was transmitted. The CRC algorithm is calculated to verify that the polynomial of the form $C(x) = x^8 + x^2 + x^1 + 1$ is validated for the summation of all bytes included in the packet that is transmitted. When a packet is written to the bq76PL536A-Q1 device from the microcontroller, the program within the microcontroller is capable of calculating the CRC code. When a packet is read from the device, the CRC code is automatically produced by the device. If the CRC verification fails, the transmitted message is discarded and the fault status line is asserted signaling that the CRC failure flag has been set.

5.3. Battery Monitoring / Fault Detection

The primary purpose of the main program, as seen in Figure D.1 of Appendix D, is to interface all devices of the BMS through the microcontroller and articulate all pertinent information to the user through the LCD. The user is able to interact with the device through the pushbuttons and LCD to view individual cell voltages across all devices, temperature readings, as well as all alert and fault conditions that may be active. The flowcharts that demonstrate the interfacing of the pushbuttons and LCD to display information can be found in Figures D.4 to D.6 of Appendix D.

The process in which cell voltages and temperature readings are sampled by the bq76PL536A-Q1 devices is called a conversion cycle. Each conversion cycle is asserted by the microcontroller through the firmware by setting the CONV bit in the CONVERT_CTRL register. The broadcast address is used to assert the start conversion process to all devices concurrently once every second. A sampling frequency of once every second is adequate for the purposes of the BMS. Once the conversion cycle is complete, the sampled data is stored in the read-only memory of the bq76PL536A-Q1 devices. Each cell voltage reading is stored in a 14-bit big-endian format that spans two adjacent 8-bit registers [4]. The **most significant bit (MSB)** is stored in the lower-address register of the pair. Cell voltage measurements are obtained by first reading the MSB and **least significant bit (LSB)** data. The millivolt value of the cell voltage is then obtained from the MSB and LSB values by solving the following equation in the program:

$$mV = (REG_{MSB} * 256 + REG_{LSB}) * 6250/16383 \quad (5.1)$$

Each temperature sensor reading is stored in two adjacent 8-bit registers. The MSB and LSB data are read then used to solve the following equations to obtain a temperature reading:

$$R_{TS1} = ((Temp_{MSB} * 256 + Temp_{LSB}) + 2)/33046 \quad (5.2)$$

$$R_{TS2} = ((Temp_{MSB} * 256 + Temp_{LSB}) + 9)/33068 \quad (5.3)$$

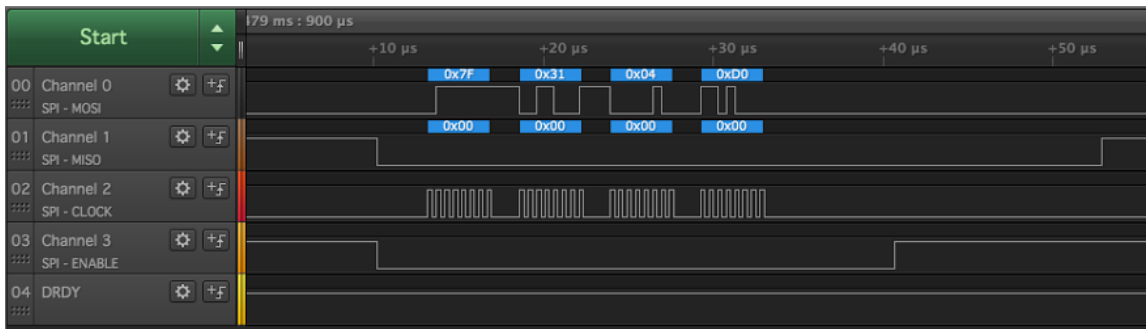
$$\frac{1}{T} = A_1 + B_1 \ln\left(\frac{R_{TS}}{R_{25}}\right) + C_1 \ln^2\left(\frac{R_{TS}}{R_{25}}\right) + D_1 \ln^3\left(\frac{R_{TS}}{R_{25}}\right) \quad (5.4)$$

where $A_1 = 0.003354016$, $B_1 = 0.000300131$, $C_1 = 5.08516E-06$, and $D_1 = 2.18765E-07$.

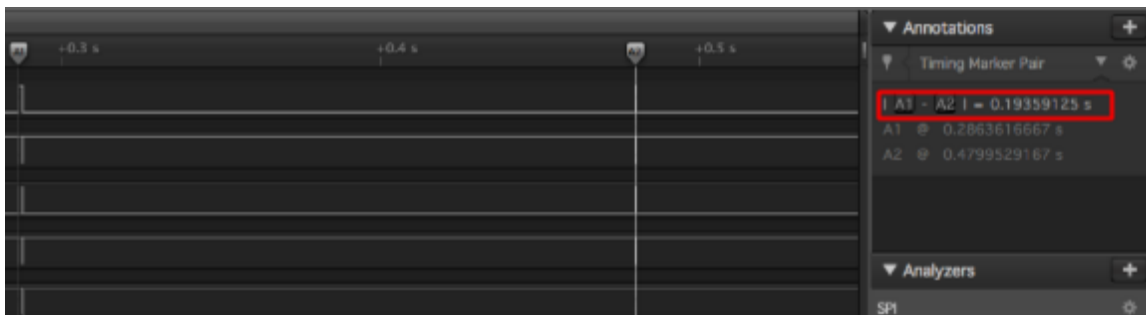
Each bq76PL536A-Q1 device detects if any of the battery protection set points have been exceeded. If either an overvoltage or undervoltage condition is detected on any of the cells, the device asserts the fault signal and indicates the source of the fault into the FAULT_STATUS

register. If an over temperature condition is detected on any of the temperature sensors, the bq76PL536A-Q1 device asserts the alert signal and indicates the source of the alert into the ALERT_STATUS register. The main program, which polls the status of both the fault signal and alert signal will illuminate the red LED, which indicates if either signal is asserted. The appropriate FAULT_STATUS or ALERT_STATUS registers are read by the program and then displayed on the LCD to indicate the source of the fault or alert to the user. The user may refer to the bq76PL536A-Q1 datasheet to interpret the binary fault or alert code displayed on the LCD. The program then attempts to clear the fault or alert status if the physical condition which asserted the fault or alert is no longer active.

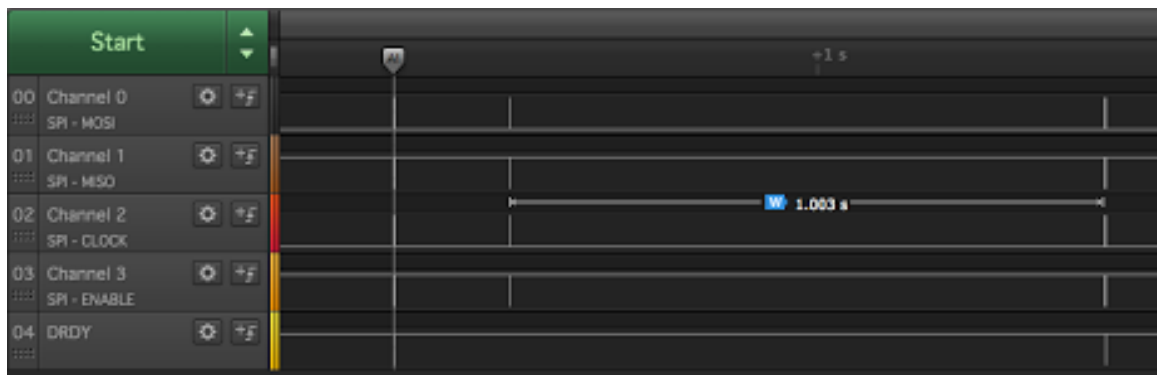
The BMS program puts all bq76PL536A-Q1 devices into a low power sleep mode between requests for conversion cycles. Sleep mode is asserted by asserting the SLEEP bit in the IO_CONTROL register as shown in Figure 5.2(a). As can be seen from Figure 5.2(b), a single cycle of the main program requires the bq76PL536A-Q1 to be active for 0.19 seconds and then is put in sleep mode until the next request for conversion 1.003 seconds later (as seen in Figure 5.2(c)). In the worst-case scenario, the device draws a maximum of 10mA while active, and only draws 12uA while in sleep mode [4]. This translates to a power efficiency improvement of roughly 84%.



(a)



(b)



(c)

Figure 5.2: Program digital analyzer output. (a) Sleep packet. (b) Duration of main program before sleep cycle. (c) Duration of sleep cycle.

5.4. Cell Balancing

The flowchart for the cell balancing program can be found in Figure D.3 of Appendix D. The status of a pushbutton interfaced with the Arduino Uno microcontroller is polled at the start of the main program to determine if the BMS is in charging mode. The user closes the switch if they wish to put the BMS in charging mode. Unbalanced lithium-ion batteries are charged and balanced in three stages: (I) constant current, (II) balancing, and (III) constant voltage. Any two-stage constant-current and constant-voltage charger can be used with a relay or contactor controlled by the microcontroller at the **Alternating Current (AC)** input of the charger. The cell balancing program provides a control signal for the charger to turn off charging during the balancing stage.

The program continually polls the voltages of the cells to determine if any of the cells has reached the full charge voltage during stage (I). When a cell has reached full charge voltage, the program then determines if any other cells are out of tolerance (greater than 0.001 V) from the fully charged cell to perform top balancing. If any cells are out of tolerance, the program performs stage (II) by turning off the charger, then setting the corresponding CBAL bit in the CB_CTRL register to discharge the full cell. The program continues to monitor the cell voltages and stops cell balancing once the discharged cell is within balancing tolerance of the lower voltage cells. The charger is able to perform stage (III) of the charging cycle once all cells are balanced.

A safety timeout timer is used in conjunction with the cell balancing control outputs to ensure that a cell is not accidentally discharged longer than it should be. The internal safety timer programmed into the CB_TIMER register clears the CB_CTRL outputs when expired. In order to prevent discharging beyond the balancing tolerance per program cycle, a CB_TIMER value of 30 seconds was chosen. At full charge, a cell has a maximum discharge rate of 88 mA through the

47 Ω discharge resistor. To discharge the allowable balancing tolerance of 0.001 V, it would take a lot longer than 30 seconds at 88 mA. All battery protection features of the BMS program remain active within charging mode. The program displays cell voltage and temperature readings on the LCD during stage (I) and (III) of the charging cycle, then notifies the user that cell balancing is active during stage (II).

Chapter 6

6. Printed Circuit Board Module

The PCB design was done with the use of Altium Designer 16, provided by UMSAE. It consists of circuitry for (I) battery monitoring and cell balancing; (II) temperature monitoring; (III) the vertical bus; and (IV) the isolation circuit for communications with the Arduino Uno. Each of these sub-circuits was tested individually on the PCB before integrating the entire system. The step-by-step testing and integration of the sub-systems allowed the design team to narrow down and ultimately eliminate the challenges for the final integration. Figure 6.1 shows the layout and different sub-circuits of the PCB.

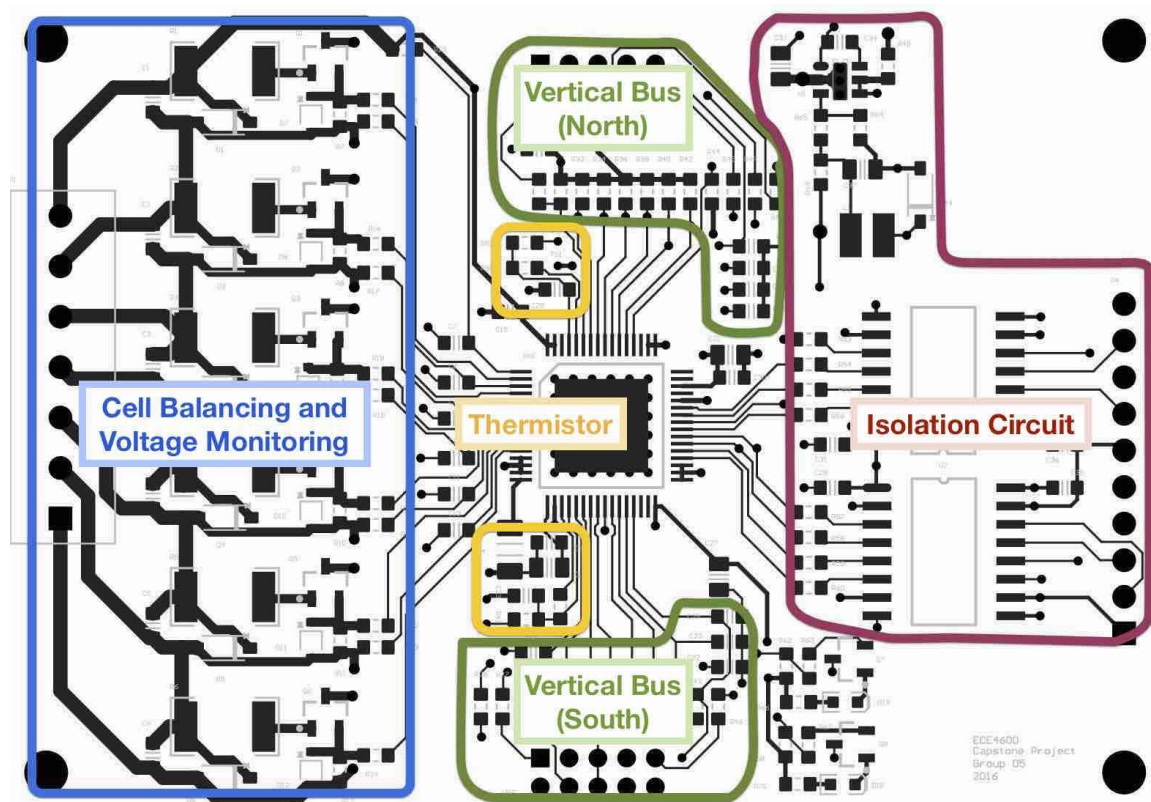


Figure 6.1: PCB layout and different sub-circuits.

6.1. PCB Layout

Component placement was completed with the board size optimization in mind. One of the motivations for this project was to reduce the size and cost of the BMS; thus, making the PCB smaller served both goals. For a large system, multiple cascaded PCBs are necessary and even a miniscule optimization will have a considerable impact on the overall cost of the system. To reduce noise in the system and to enhance the stability of the bq76PL536A-Q1, all of the decoupling capacitors were placed in close proximity to the bq76PL536A-Q1. In addition, the internal 5 V analog and digital supplies were connected to low **Equivalent Series Resistance (ESR)** capacitors. In order to obtain the desired ESR, a 0.1 μF and a 2.2 μF were used in parallel.

Another important consideration in the PCB design was the thermal management of the bq76PL536A-Q1. The bq76PL536A-Q1 features a PowerPAD underneath it for more efficient heat dissipation; nonetheless, proper heat removal from the chip is only possible with the appropriate PCB design and assembly. To fulfil this requirement, an 8x8 mm 1 oz copper pad with 36 vias was used as a thermal land to transfer heat from the chip to the ground plane.

The PCB consists of four layers: the top signal, the ground layer, the power layer, and the bottom layer. The four layers were necessary to simplify the layout of the complex connections in comparison to a two-layer PCB. The ground and power layers are each divided into two parts: **High Voltage (HV)** and **Low Voltage (LV)** sides, with galvanic isolation between them as shown in Figure 6.2. Both the HV ground and power planes act as a thermal heatsink for the board. Additionally, the ground plane was necessary to minimize unwanted noise in the system.

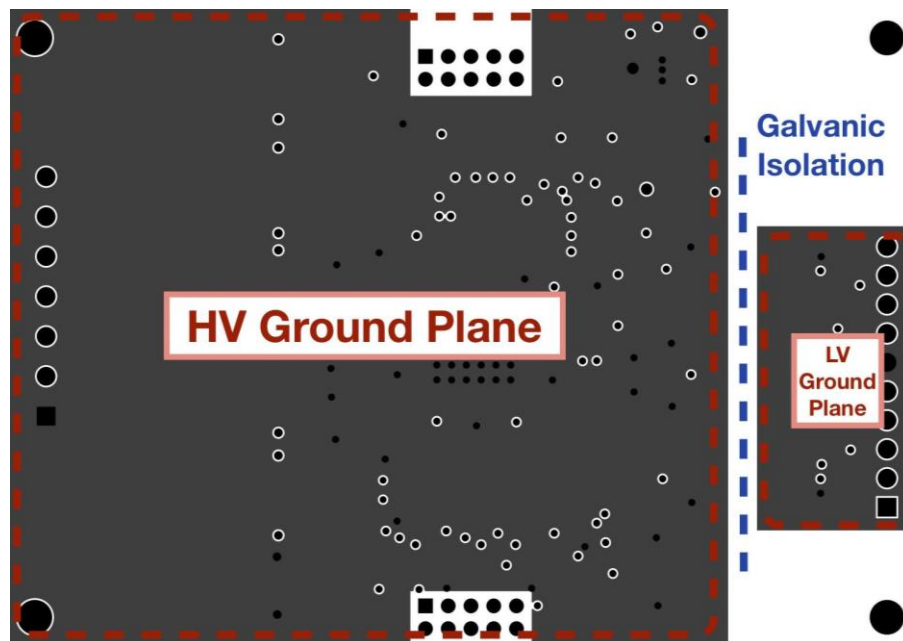


Figure 6.2: High voltage and low voltage ground planes.

6.2. Cell Balancing Circuit Considerations

To dissipate the power from an unbalanced cell, each cell can be discharged through a $47\ \Omega$ power resistor by turning on a MOSFET through the bq76PL536A-Q1 cell balancing pins. As cell balancing will discharge a cell at 88 mA, the traces are at least 25 mil, which are rated for 1.7 A of current with a $10\ ^\circ\text{C}$ rise [9]. These traces also act as small heatsinks and they assist the power resistors in dissipating heat.

6.3. Isolation Circuit and Power Regulator

One of the FSAEE requirements is to have isolation between HV and LV of the system. The PCBs HV and LV ground and power planes have a galvanic isolation as shown in Figure 6.2. The isolation devices (iso7241) used in this design provide an isolated communication path between the Arduino and the first bq76PL536A-Q1.

The iso7241 must be powered by two separate 5 V sources with $0.1\ \mu\text{F}$ bypass capacitor in close approximation to the VCC pins. In addition, a switching regulator with 70 % efficiency was used in the design to step down the VBat to 5 V on the HV. On the LV side, the 5 V is taken from the Arduino. However, during the subsystem testing, it was determined that the regulator was did not supply the expected voltage even though it was operating under its electrical ratings.

6.4. Vertical Bus Configuration

The vertical bus can be enabled/disabled through hardware with $0\ \Omega$ resistors that act as jumpers that allow a board to be either connected to another board or to operate alone. For a single board operation, R29 - R44 should be populated using the $0\ \Omega$ resistors, which connect the north and south vertical communication of the bq76PL536A-Q1 to the VBAT pin and ground. Alternatively, if the boards are to be daisy-chained, each board should have a vertical bus set up according to the position of the board in the stack:

1. The first board: $0\ \Omega$ resistors should be used to populate the odd numbered resistors from R29 to R44 in order to disable the south communication bus, while the rest of the south communications components should not be populated. Additionally, $1\ \text{k}\Omega$ resistors should be used to populate R25, R45, R47, R48 and $33\ \text{pF}$ ceramic capacitors used to populate C13, C18, C19, C24 to prepare the north communication connection.
2. Intermediate boards: $1\ \text{k}\Omega$ resistors should be used to populate R25-R28, R45-R48 and $33\ \text{pF}$ ceramic capacitors used to populate C13, C17-C19, C22-C25 to enable both the south and north communication bus, and R29 -R44 should not be populated.

- The top board: 1 k Ω resistors should be used to populate R26-R28, R46 and 33 pF ceramic capacitors to populate C17, C22, C23, C25 enable the south communication bus. In addition, 0 Ω resistors should be used to populate the even-numbered resistors from R29 to R44 in order to disable the north communication bus.

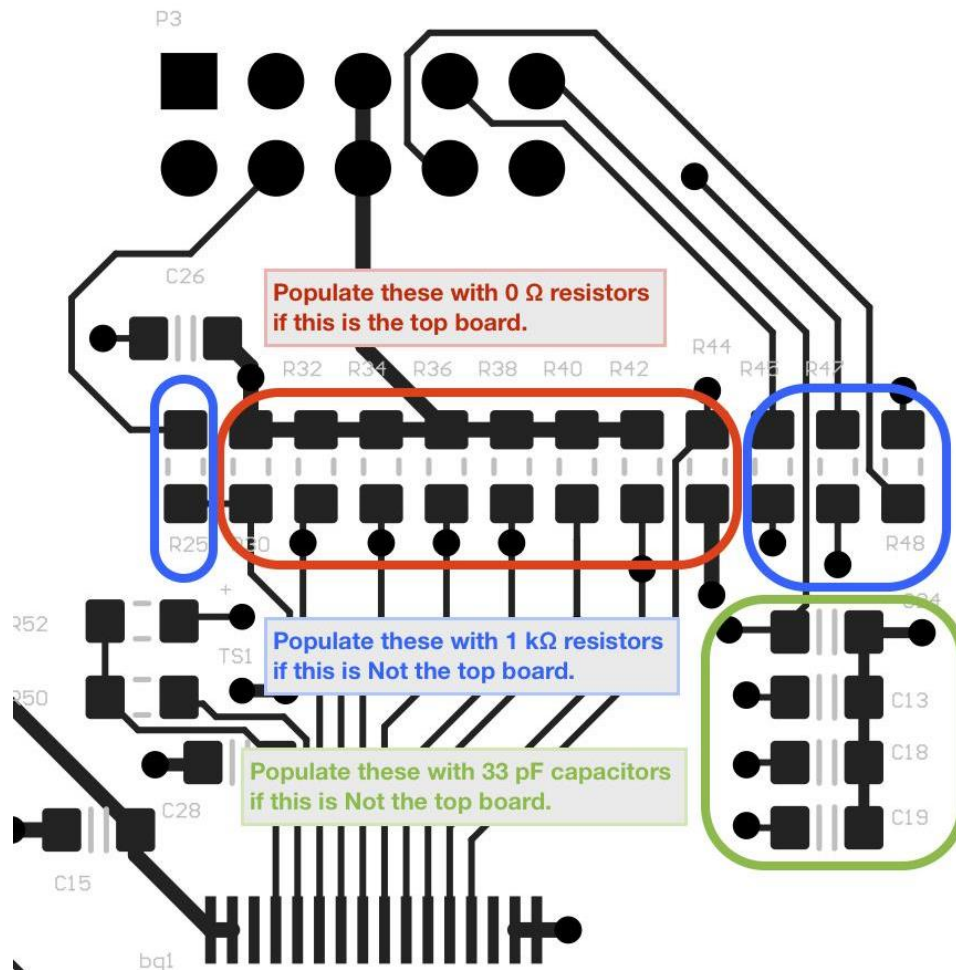


Figure 6.3: Example of the North Vertical Bus of a board.

Chapter 7

7. System Integration and Evaluation

Following the development and testing of the individual sub-systems, the designed PCB was fabricated in order to integrate and test the system. The first stage of system testing was to populate the PCB and to ensure that it was performing as expected. The following sections detail the integration and testing of the sub-system on the PCB platform, as well as the final system evaluation.

PCB Fabrication and Population

Because the team's design consisted of a four-layer board that could not be fabricated by the University of Manitoba's Electrical and Computer Engineering Tech Shop, the PCB fabrication was outsourced to a PCB manufacturing company. For proper heat transfer from the chip to the ground plane, the PowerPAD beneath the bq76PL536A-Q1 must be soldered to its designated thermal land. The team members' inexperience with soldering of these types of packages prompted the team to seek Zoran Trajkoski's assistance with the mounting of the bq76PL536A-Q1. Unfortunately, the University of Manitoba did not possess the necessary equipment for a reliable population of ICs with PowerPAD. Mr. Trajkoski improvised by soldering the chip using an infrared oven, but this method only resulted in one functional mounted chip and damaged the other two, as the excessive heat that was applied to the top of the chip likely damaged its internal components.

Consequently, the team sought out other facilities that were capable of mounting the IC. IDERS Inc. was kind enough to mount the bq76PL536A-Q1 ICs on very short notice at their state-of-the-art facility.

7.1. Voltage Regulator

The voltage regulator that was used in the initial design did not deliver the required voltage with a 200 mA load current, despite the fact that it was rated to supply up to 300 mA. It was determined that the reason for the voltage drop at a higher-load current was the IC's lack of heat dissipation capabilities. The LMR14203 has a small SOT-23-6 Package (2.97 x 1.65 x 1mm). To supply 200 mA at 5 V, the output power is 1 W. Given the 80 % efficiency of the device, there is 250 mW of power to be dissipated from the package [10].

As an alternative solution, a linear 3-terminal voltage regulator LM7805 was selected to supply the isolated 5 V needed for the isolation devices. This regulator steps down the external 9 V battery to 5 V. A single-layer PCB was designed and manufactured at the University of Manitoba to accommodate the new regulator where its negative terminal is tied down to the base board's ground plane [11].

7.2. System Evaluation

Three PCB modules, the Arduino Uno, and the LCD were connected together to create the prototype BMS. The Arduino Uno, acting as the host, was connected to the first PCB module. The second and third PCB modules were then connected through each PCB modules' vertical bus. The final BMS system diagram is shown in Figure 7.1. To evaluate the performance of the system, two types of tests were configured:

1. To evaluate reading time, accuracy, and fault detection.
2. To evaluate the system's measurement ranges for voltage and temperature.
3. To demonstrate that the BMS is able to perform cell balancing.

The BMS was tested using two power supplies to simulate cells due to the limited supply of only six cells. When connecting a power supply to the BMS, the voltage is roughly divided into six values, each one representing a cell. Each of the three types of tests was performed using the set-up shown in Figure 7.2.

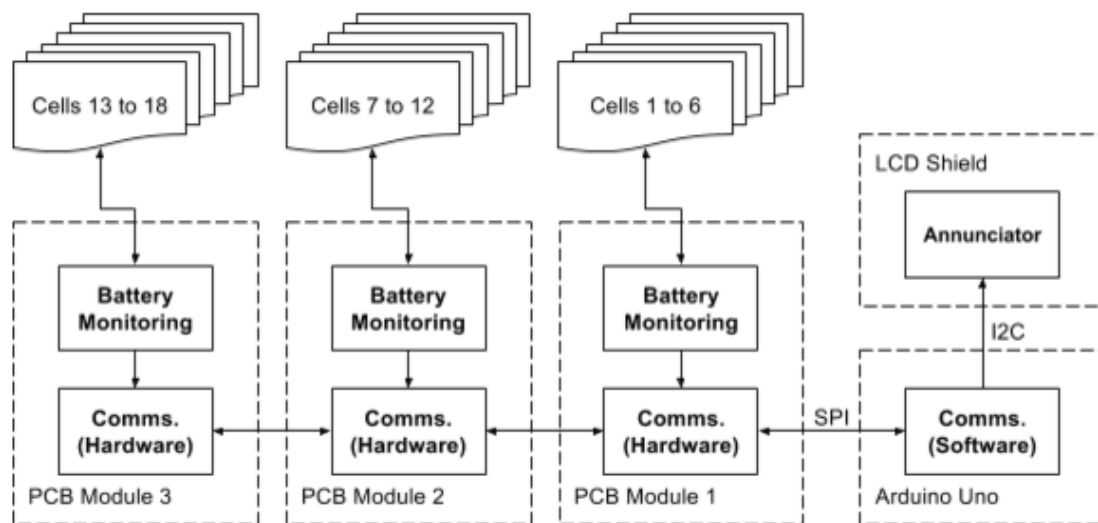


Figure 7.1: System diagram of the BMS.

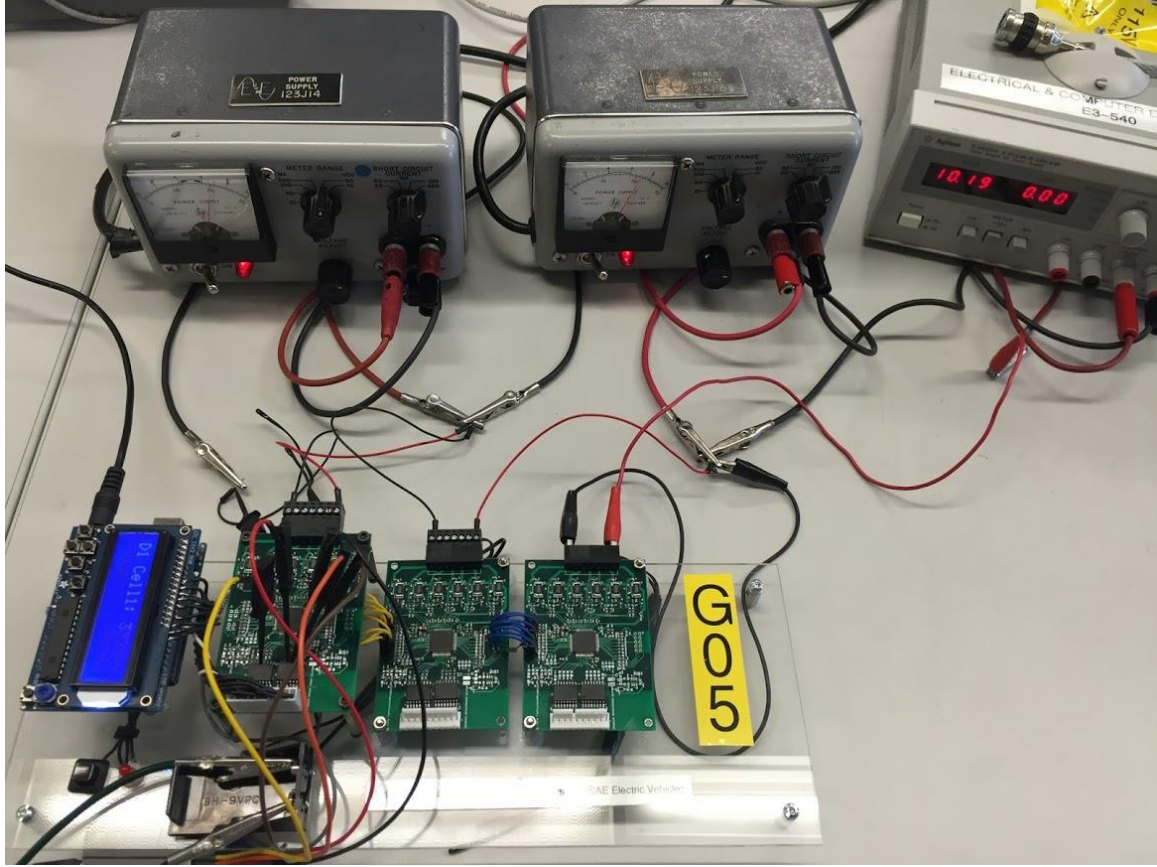


Figure 7.2: The test setup for full system evaluation.

7.2.1. Reading Time, Accuracy, and Fault Detection Tests

Using the BMS setup shown in Figure 7.2, 24 tests were performed to provide validation. Each of the 24 tests used different variations of programmed **undervoltage (UV)**, **overvoltage (OV)**, and **over temperature (OT)** limits to validate that the system was programmable as required. The steps taken for each test were as follows:

1. Program system with undervoltage, overvoltage, and over temperature limits.
2. Using a multimeter, measure and record the voltages of every cell connected to the BMS, while at the same time recording the respective cell voltages displayed by the BMS LCD.
3. Using a Logic analyzer, sample the host DRDY pin and record the time for DRDY to switch from low to high to low, as shown in Figure 7.3.
4. To simulate an undervoltage condition, carefully lower the power supply voltage to the undervoltage limit and record if a fault is detected.
5. To simulate an overvoltage condition, carefully raise the power supply voltage to the overvoltage limit and record if a fault is detected.

6. To simulate an over temperature condition, use a cup of hot water to warm the thermistors to the over temperature limit and record if a fault is detected.

The results of all 24 tests validate that the BMS performs as required; see Appendix F for full system test results. Tests #1 through #12 focus on UV detection; tests #13 through #18 focus on OV detection, and finally, tests #19 through #24 focus on OT detection. Tests #10, #14, and #24 are programmed for the fault conditions needed (UV = 2.5 V, OV = 4.15 V, OT = 65 °C) for the six EIG C020 lithium-ion polymer cells, provided by UMSAE [12].

As can be seen in Figure 7.4, during test #10, UV fault detection was demonstrated. The system is operating when reading a voltage level $>UV = 2.5$ V, as seen in Figure 7.4(a). The system sends a shutdown signal, indicated by a red LED, when reading a voltage level $<UV = 2.5$ V, as seen in Figure 7.4(b). Likewise, in test #14, OV fault detection was demonstrated, as shown in Figure 7.5. The system is operating when reading a voltage level $<OV = 4.15$ V, but then sends a shutdown signal when reading a voltage level $>OV = 4.15$ V. Finally, test #22, OT fault detection, is demonstrated in Figure 7.6. The system is operating when reading a temperature level $<OT = 65$ °C, as shown in Figure 7.6(a), but then sends a shutdown signal when reading a voltage level $>OT = 65$ °C, as shown in Figure 7.6(b).

The end results of all of the tests showed an average accuracy of voltage readings at 1.03 mV, with a worst-case accuracy of 40 mV. The average reading time per cell was 31.81 μ s, with a worst-case reading time of 31.82 μ s. It is important to note that each test successfully detected the fault condition, providing a 100 % detection rate. These results demonstrate that the BMS meets the requirements listed in Table 1.1, as it is programmable, accurate, detects faults, and exceeds the required reading time.

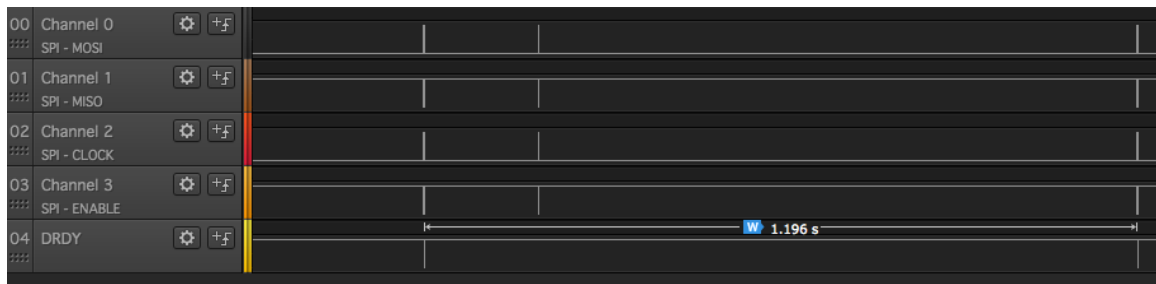
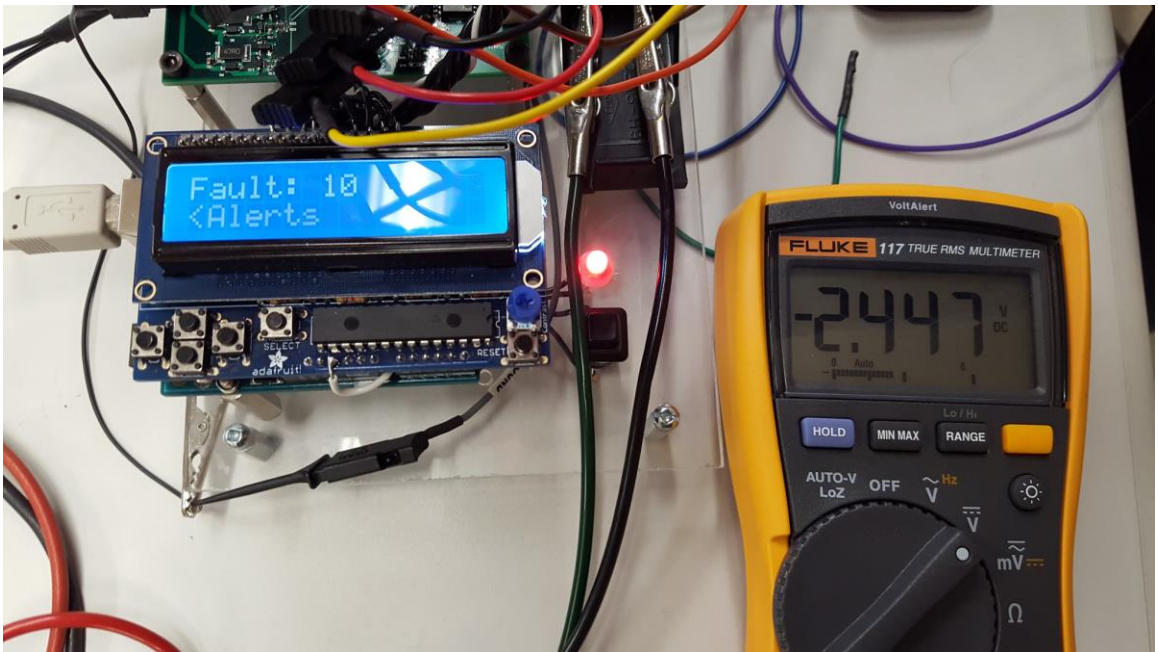


Figure 7.3: Logic analyzer output displaying DRDY duration.



(a)



(b)

Figure 7.4: Test #10. (a) Prior to UV trip. (b) Successful UV detection.

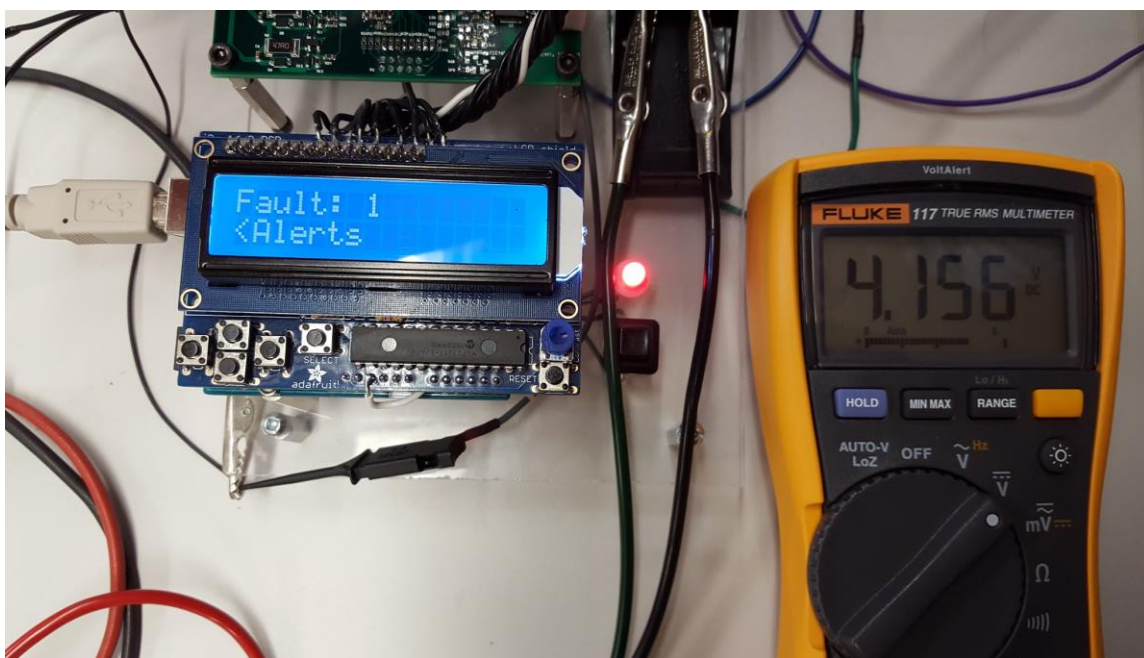
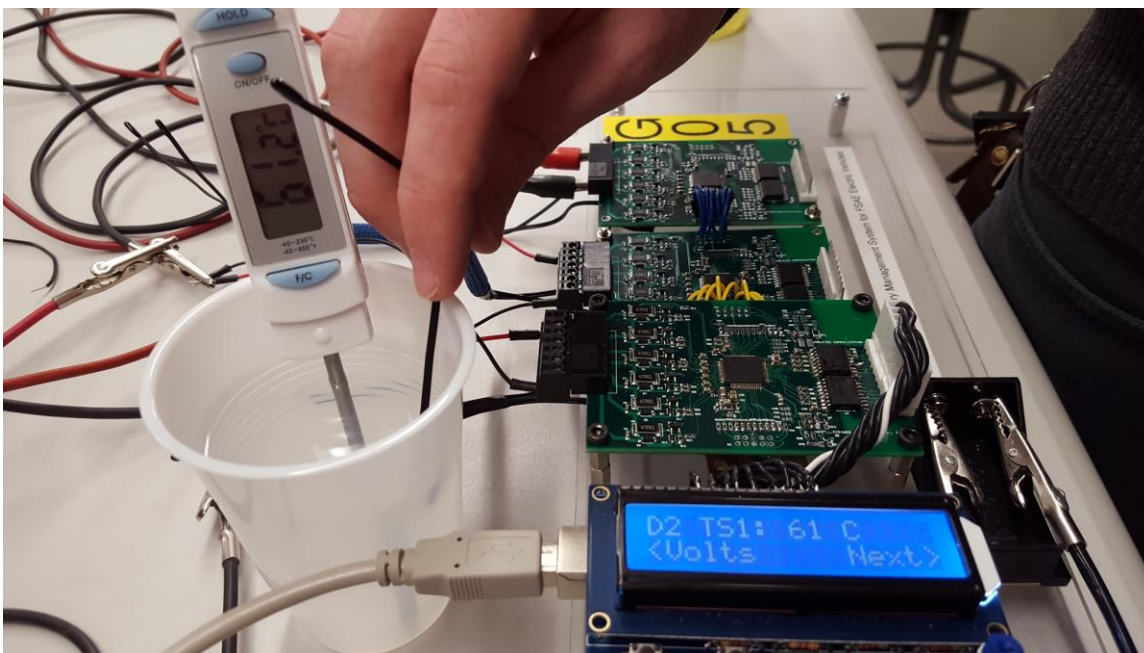
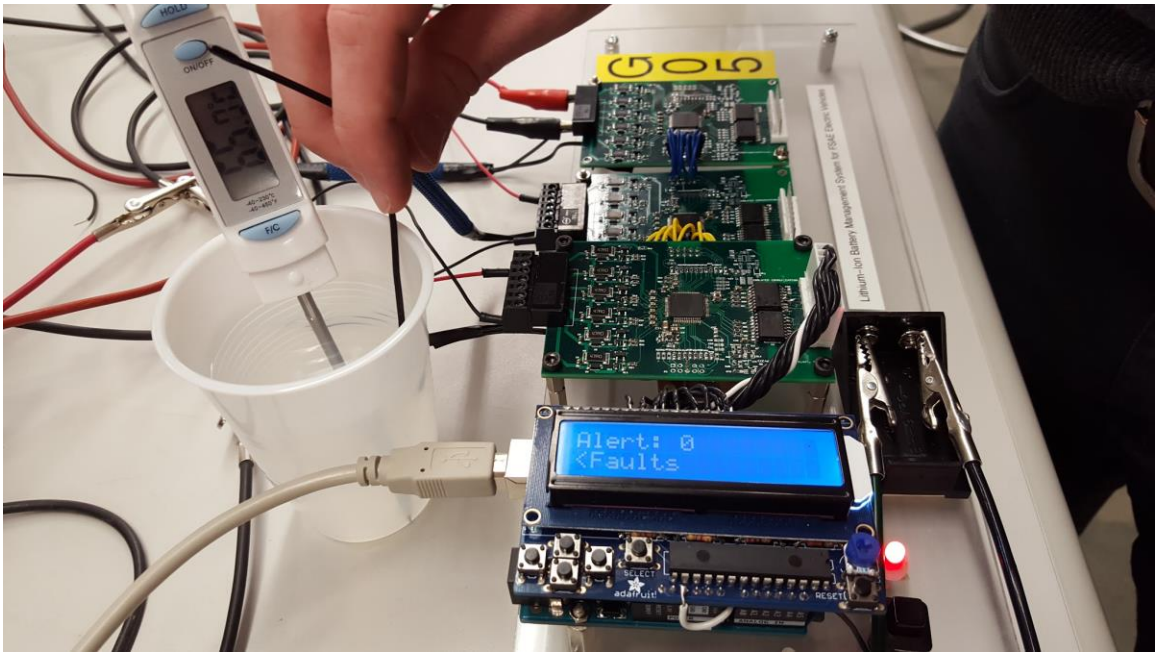


Figure 7.5: Test #14, showing successful OV detection.



(a)



(b)

Figure 7.6: Test #22. (a) Prior to OT trip. (b) Successful OT detection.

7.2.2. Measurement Range Tests

The test setup shown in Figure 7.2 was used to measure the ranges of operating voltages and temperature measurement. The ranges of operating voltages were evaluated by disabling the systems undervoltage and overvoltage fault detection, then adjusting one of the power supplies voltage levels to minimum and maximum values. The BMS operated with voltages as low as 465 mV per cell as seen in Figure 7.7. The BMS was unable to operate at voltages lower than 465 mV per cell, because the PCB module is powered by the cells. The BMS operated with voltages as high as 6.014 V per cell, as shown in Figure 7.8.

As previously demonstrated, test #24 showed that the BMS measured temperatures of above 60 °C. To show that the BMS can measure colder temperatures, the temperature of a cup containing ice water was measured at 0 °C, as shown in Figure 7.9. A test to validate that the BMS was able to measure temperatures as low as -40 °C was not performed. These results demonstrate that the operating voltage and temperature range requirement as listed in Table 1.1 were nearly met.

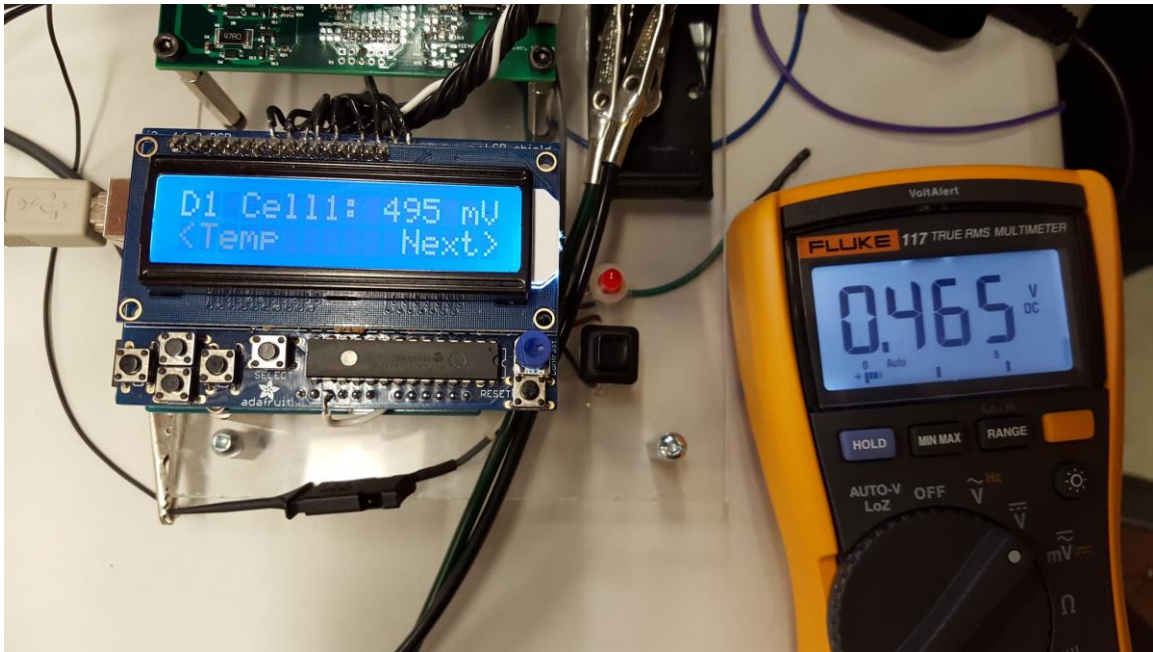


Figure 7.7: The minimum operating voltage of the BMS.

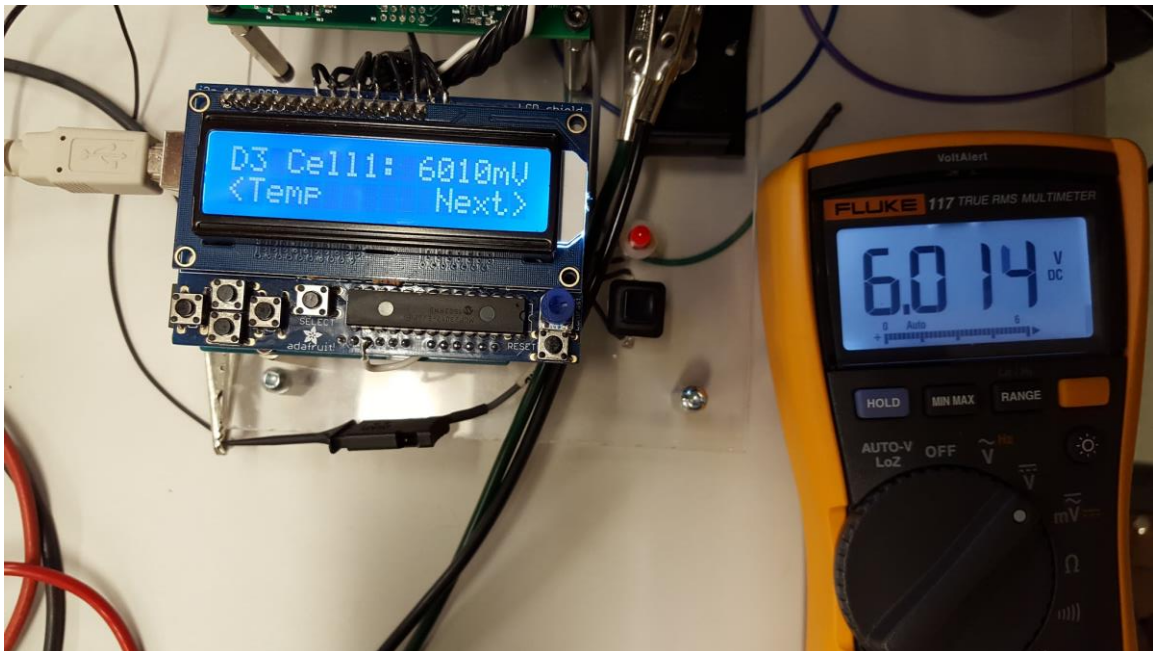


Figure 7.8: The maximum tested operating voltage of the BMS.

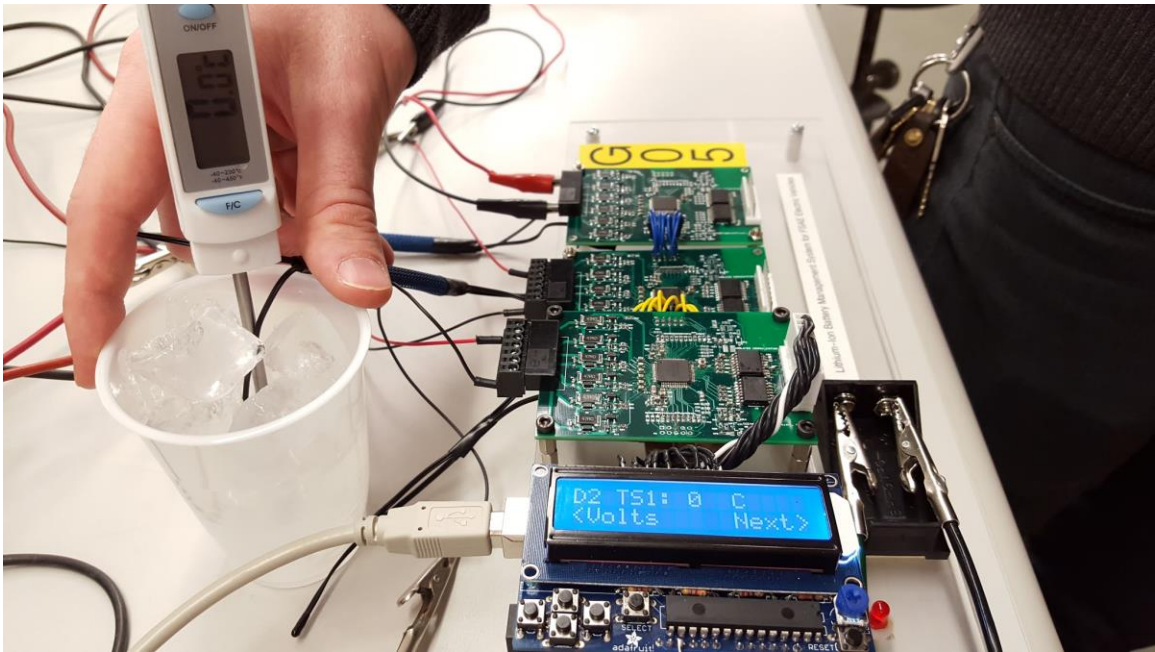


Figure 7.9: The minimum tested temperature reading of the BMS.

7.2.3. Cell Balancing Test

For the purposes of demonstrating that both the software and hardware for cell balancing are functional, a few test conditions had to be modified. Unfortunately, a lithium-ion battery charger was not readily available and could not be purchased within the team's budget. Thus, testing was not completed with integration of a charging circuit. Therefore, the following test results demonstrate that the BMS is capable of discharging battery cells to balance the cells. The BMS program was modified to balance all cells that are at or above 3.727 V. This is different from the normal set point of 4.14 V for top balancing below the OV threshold (4.15 V) due to the lack of a charger to charge the cell voltage to full SOC.

As can be observed from the test results in Table F.3 of Appendix F and Figure 7.10, the software in conjunction with the cell balancing circuit is capable of balancing all cells at or above 3.727 V to the lowest cell voltage (3.726 V). Roughly 3.5 hours were required to balance six cells that were between 1-3 mV out of balance.

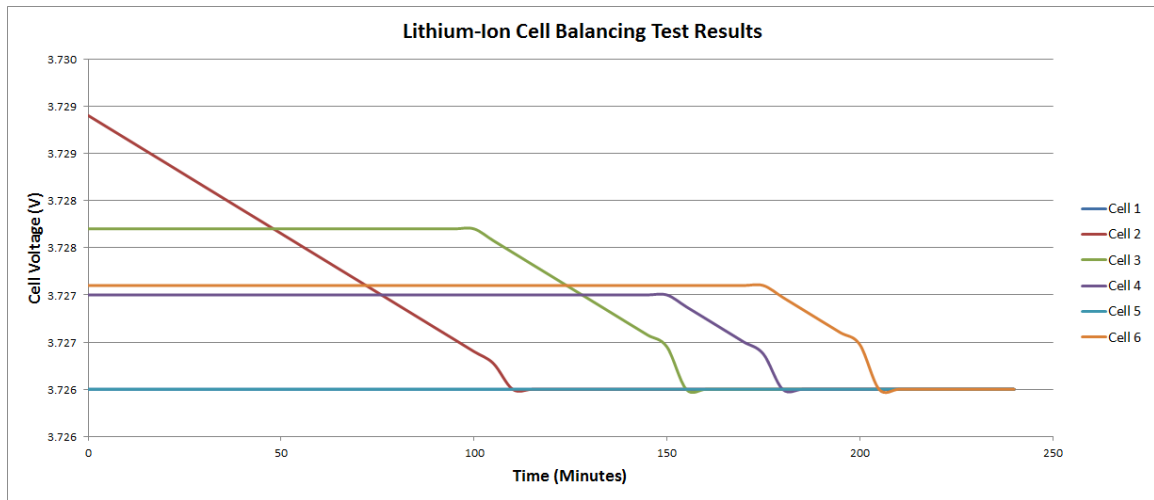


Figure 7.10: Cell voltages as a function of time for cell balancing test.

7.3. Integration Summary

The integration and evaluation of all four sub-systems was successfully completed. Additionally, a vertical bus interface between multiple PCB modules was implemented. After performing the prototype testing, the results demonstrated that nearly all project specifications had been achieved or exceeded. The target specifications for the project, as well as the achieved levels, are summarized in Table 7.1.

Table 7.1: BMS proposed specifications compared to achieved performance.

Specification	Target	Achieve
Voltage Measurement Range (per cell)	0V to +6 V	0.5 V to +6.0 V
Voltage accuracy @ 3.6V, 25°C	50 mV	1.03 mV
Voltage reading time / cell	100 μ s to 250 μ s	32 μ s
Temperature Measurement Range	-40 °C to +60 °C	0 °C to 65 °C
Temperature Sensors	1 per 3 cells	1 per 3 cells
Programmable Under-Voltage and Overvoltage	Yes	Yes
Programmable Operating Temperature	Yes	Yes
Critical Operating Voltage Level Fault Detection	Yes	Yes
Critical Operating Temperatures Fault Detection	Yes	Yes
Fault Indication	Red LED	Red LED

Management Capacity	6 cells in series	18 cells in series
Scalable Management Capacity	N/A	Yes
Cell Balancing	Yes	Yes
LCD Human Machine Interface	Yes	Yes
System Function Error Detection	Yes	Yes

Chapter 8

8. Conclusion

The purpose of this project was to design and evaluate a proof-of-concept lithium-ion BMS for FSAEE vehicles. The project was successfully completed as the voltage of six cells in series as well as the temperature of two cells were monitored for safe operating conditions, with 100 % fault detection. The undervoltage, overvoltage and over temperature conditions were fully programmable and a shutdown fault signal was transmitted and emitted a red LED when reaching critical conditions. Voltages from 0.465 V to 6.014 V were monitored, with an accuracy and reading time of 1.03 mV and 31.81 μ s respectively, exceeding target specifications. Testing also verified that the BMS could successfully monitor a temperature range from 0 °C to 65 °C; however, cold weather testing of -40 °C was not completed.

Four additional features were developed for the BMS: (I) implementation of a vertical bus allowing for extended total management capacity (up to 192 cells); (II) functional cell balancing to prolong cell life span; (III) the addition of an LCD human-machine interface to improve the end-user's interaction with the BMS; and (IV) implementing basic system functionality error detection to allow the user to distinguish between cell faults and system faults. The project, including the additional features, was completed on time and on budget, requiring \$469.83 in total expenditures.

The BMS was divided into four sub-systems: (I) the PCB module; (II) the battery monitoring sub-system; (III) the communication and annunciator sub-system; and (IV) the cell balancing sub-system. Each PCB module contains the battery monitoring sub-system, which can monitor the voltages of six cells, with one temperature sensor per three cells. In order to increase BMS management capacity, the PCB modules can also be stacked in daisy chain fashion. The entire system is controlled by an Arduino Uno, which features an Adafruit LCD Shield. With the successful completion of the project, the BMS offers a possible alternative to the BMS currently being used by the UMSAE Formula Electric team.

8.1. Recommendations for System Improvement

Although the proof-of-concept BMS has been successfully completed, there are still several possibilities for system improvements. Some recommendations for future iterations of this project include the following:

- Reduce the production costs of the PCB module that does not use the device-to-host interface.
- Develop a graphical user interface to allow the user to more easily program the fault conditions.
- Remove the Arduino Uno as the host device and interface the BMS with the UMSAE FE main controller.
- Added cell parameter monitoring such as continuous discharge current, maximum discharge current, and internal cell resistance.
- Introduce memory to the system for data acquisition.

References

- [1] L. Lua et al., “A review on the key issues for lithium-ion battery management in electric vehicles,” J. Power Sources, vol. 226, pp.272-288, Mar. 2013.
- [2] D. Andrea. (2011, Oct. 25). Battery Management Systems [Online]. Available: http://liionbms.com/php/about_bms.php
- [3] Li-Ion BMS. *BMS chips options*. [Online]. Available: http://liionbms.com/php/bms_chips_options.php [Accessed: Mar 01, 2016]
- [4] Texas Instruments. (2011, May). “*bq76PL536A-Q1*”. [Online]. Available: <http://www.ti.com/lit/ds/symlink/bq76pl536a-q1.pdf> [Accessed: Nov. 01, 2015]
- [5] Arduino.cc. (2016). *Arduino-ArduinoShields*. [Online]. Available: <https://www.arduino.cc/en/Main/ArduinoShields> [Accessed: Mar 01, 2016].
- [6] Y. Barsukov, Battery Power Management for Portable Devices, Norwood: Artech House, 2013.
- [7] S. Wen. (2009).“Cell balancing buys extra run time and battery life,” *Analog Applications Journal [Online]*,pp. 14-18. Available: <http://www.ti.com/lit/an/slyt322/slyt322.pdf> [Accessed: Mar 01, 2016].
- [8] A.H. Zimmerman, *Life Test and Performance Comparisons of Lithium-ion with Traditional Satellite Battery Types*, 2002, The Aerospace Corporation
- [9] University of Michigan, Department of Electrical and Computer Engineering. *ECE Technical PCB Services*. [Online]. Available: <https://www.egr.msu.edu/eceshop/pcb/DRC.php> [Accessed: Mar 01, 2016].
- [10] Texas Instruments. (2013, Apr.). “*LMR14203 SIMPLE SWITCHER*”. [Online]. Available: <http://www.ti.com/lit/ds/symlink/lmr14203.pdf> [Accessed: Jan. 14, 2016].
- [11] Fairchild. (2014, Sep.). “*LM7805*” [Online]. Available: <https://www.fairchildsemi.com/datasheets/LM/LM7805.pdf> [Accessed: Feb. 20, 2016].
- [12] Energy Innovation Group Ltd. (EIG). “*C020 Lithium Ion Pouch Cell*” [Online]. Available: <http://www.ebaracus.com/sites/default/files/2012/12/EIG-ePLB-C020-Datasheet.pdf> [Accessed: Mar 01, 2016].

Appendix A: FSAE BMS Requirements

The articles listed below are the rules applicable to the requirements of the design of a BMS for a FSAEE vehicle. The “tractive system” mentioned in the rules refers to the high voltage battery system of a FSAEE vehicle. Ground Low Voltage or “GLV” mentioned in the rules refers to the low voltage system of a FSAEE vehicle.

EV3.6 Accumulator Management System (AMS) [A.1]

EV3.6.1 Each accumulator must be monitored by an accumulator management system whenever the tractive system is active or the accumulator is connected to a charger. For battery systems this is generally referred to as a battery management system (BMS) however alternative electrical energy storage systems are allowed and therefore AMS will be the terminology used in this document.

EV3.6.2 The AMS must continuously measure the cell voltage of every cell, in order to keep the cells inside the allowed minimum and maximum cell voltage levels stated in the cell data sheet. If single cells are directly connected in parallel, only one voltage measurement is needed.

EV3.6.3 The AMS must continuously measure the temperatures of critical points of the accumulator to keep the cells below the allowed maximum cell temperature limit stated in the cell data sheet or below 60°C, whichever is lower.

Cell temperature must be measured at the negative terminal of the respective cell and the sensor used must be in direct contact with either the negative terminal or less than 10mm away from the terminal on the respective busbar.

EV3.6.5 Any GLV connection to the AMS must be galvanically isolated from the tractive system.

EV3.6.6 For lithium based cells the temperature of at least 30% of the cells must be monitored by the AMS. The monitored cells have to be equally distributed within the accumulator container(s).

EV3.6.7 The AMS must shutdown the tractive system by opening the AIRs, if critical voltage or temperature values according to the cell manufacturer’s datasheet and taking into account the accuracy of the measurement system are detected. If the AMS does perform a shutdown then a red LED marked AMS must light up in the cockpit to confirm this.

Appendix B: Budget Summary

The allotted budget supplied by the department of Electrical and Computer Engineering was \$400.00. The development costs for this project includes the prices of items received in kind such as the evaluation module and the many components that needed repurchasing due to component failures incurred during the populating of the PCBs . The total amount invested for the development of the BMS by all parties involved is \$1910.13. The actual cost supplied by the department of Electrical and Computer Engineering totals \$469.83. The department agreed to fund the additional \$69.83 beyond the allotted budget to compensate for the weak Canadian dollar at the time of purchase. The per unit production cost of the BMS is \$185.61 for the monitoring and protection of the first six cells. The BMS can then be expanded to monitor up to a total of 192 cells at a cost of \$111.29 per six cells.

Table B.1: Project budget summary.

Item	Qty	Unit Cost	Proposed Cost	Development Cost	Actual Cost	Production Cost
Microcontroller		Subtotal	\$57.12	\$62.18	\$62.18	\$29.99
Arduino Uno Rev. 3	1	\$26.99	\$26.99	\$29.99	\$29.99	\$29.99
CAN BUS Shield	1	\$30.13	\$30.13	\$32.19	\$32.19	N/A
Monitoring Circuit		Subtotal	\$132.93	\$894.56	\$217.19	\$78.30
Battery Management IC	3	\$21.27	\$63.81	\$106.40	\$63.84	\$21.28
Temperature Sensor	6	\$6.58	\$39.48	\$11.64	\$11.64	\$3.88
Digital Isolators	2	\$6.52	\$13.04	\$74.30	\$44.58	\$14.86
Fuses	10	\$0.56	\$0	N/A	N/A	N/A
PCB Fuse Holders	10	\$0.66	\$6.60	N/A	N/A	N/A
PCB Components	1	\$10.00	\$10.00	\$128.45	\$97.13	\$38.28
IC Evaluation Module	1	\$406.98	In Kind	\$587.37	N/A	N/A
Annunciator		Subtotal	\$26.18	\$27.63	\$27.33	\$27.63
Adafruit LCD Shield	1	\$25.58	\$25.58	\$27.33	\$27.33	\$27.33

Red LED	2	\$0.30	\$0.60	\$0.30	N/A	\$0.30
Power Supplies		Subtotal	\$7.63	\$591.65	\$8.15	\$8.15
9V Wall Adapter 650mA	1	\$7.63	\$7.63	\$8.15	\$8.15	\$8.15
Lithium-Ion Battery Cell	6	\$77.25	In Kind	\$583.50	N/A	N/A
Testing/Prototyping		Subtotal	\$13.34	\$13.34	N/A	N/A
Breadboard	2	\$6.67	\$13.34	\$13.34	N/A	N/A
Code Composer Studio	1	\$1054.00	\$0	N/A	N/A	N/A
Required Resources		Subtotal	\$55.00	\$100.93	\$100.93	\$20.19
Evaluation PCB printing	1	\$12.00	\$12.00	N/A	N/A	N/A
Project PCB printing	1	\$43.00	\$43.00	\$100.93	\$100.93	\$20.19
Subtotal			\$292.20	\$1690.29	\$415.78	\$164.26
Shipping			\$42.00	N/A	N/A	N/A
Taxes			\$43.45	\$219.73	\$54.05	\$21.35
Total			\$377.65	\$1910.03	\$469.83	\$185.61

Appendix C: Printed Circuit Board Design

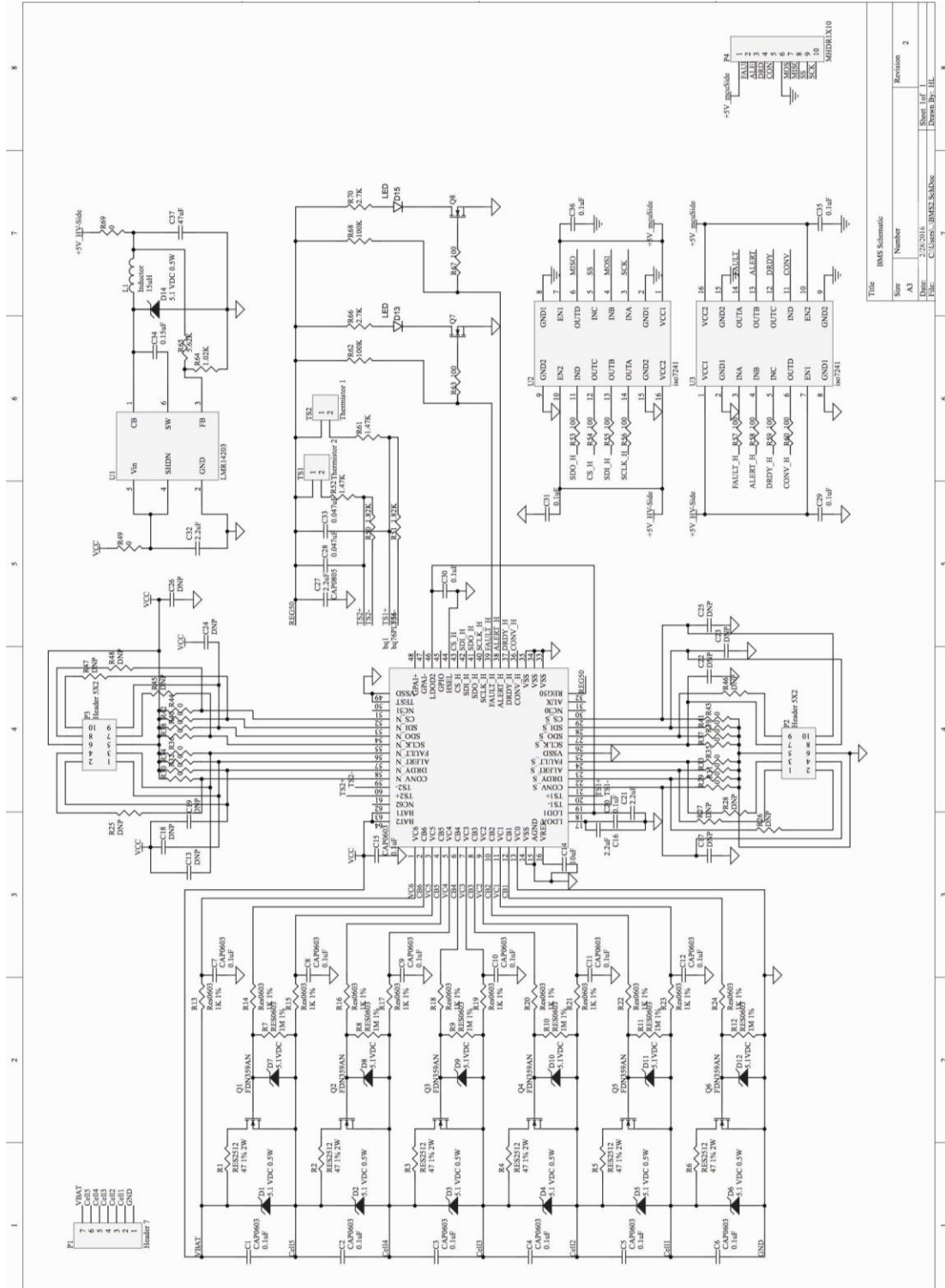


Figure C.1: Schematic of the PCB.

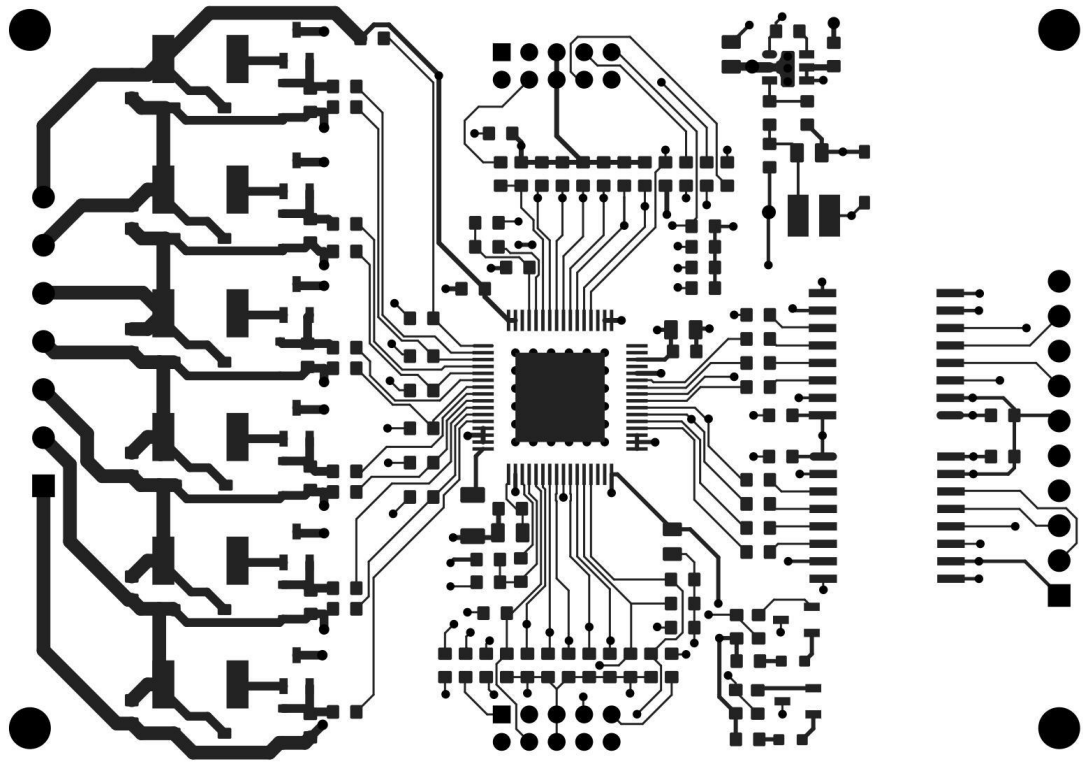


Figure C.2: Top Layer of the PCB.

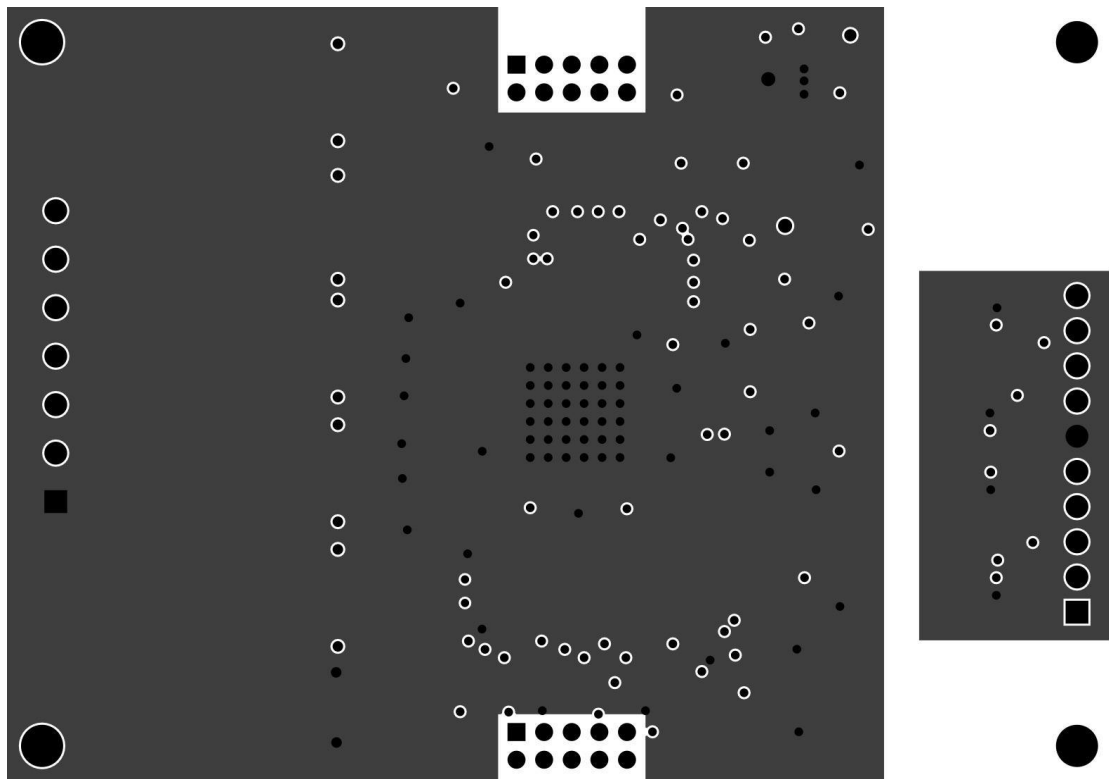


Figure C.3: Ground Layer of the PCB.

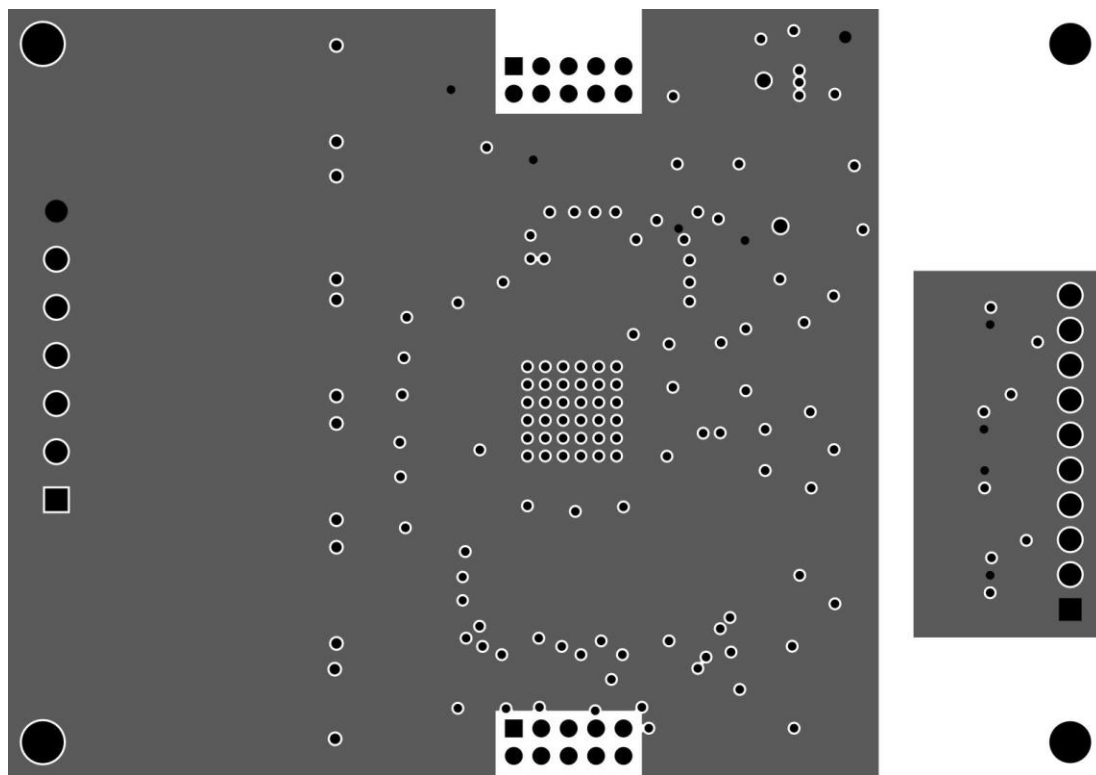


Figure C.4: Ground Layer of the PCB.

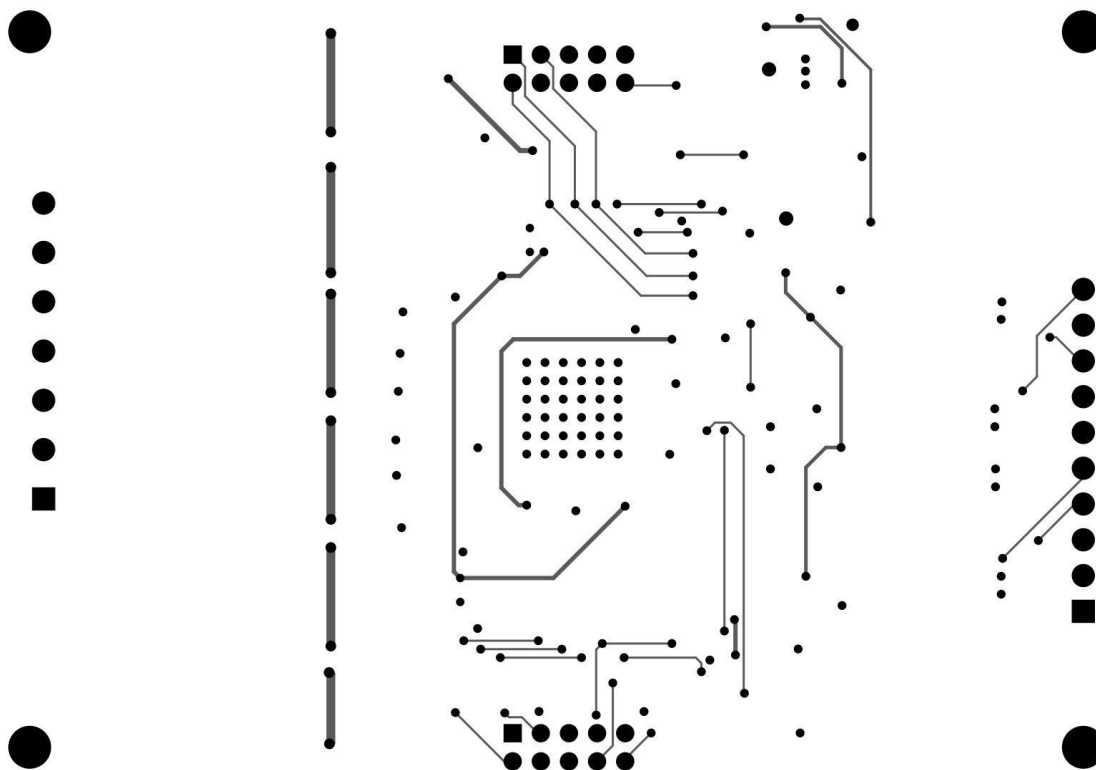


Figure C.5: Bottom Layer of the PCB.

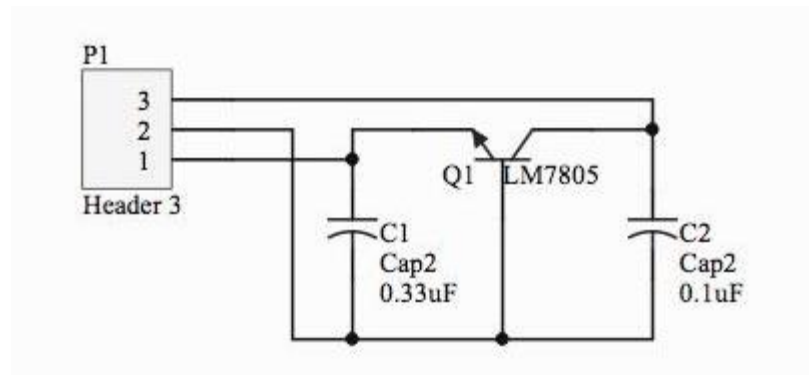


Figure C.6: Voltage regulator schematic.

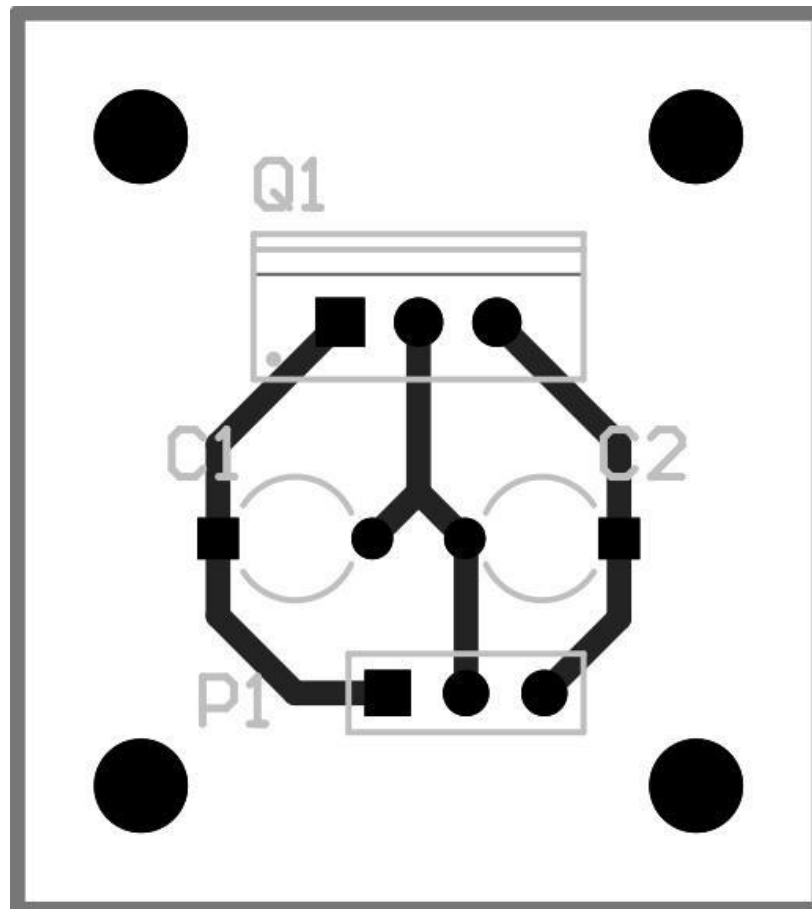


Figure C.7: Voltage Regulator PCB layout.

Appendix D: Software Flowcharts

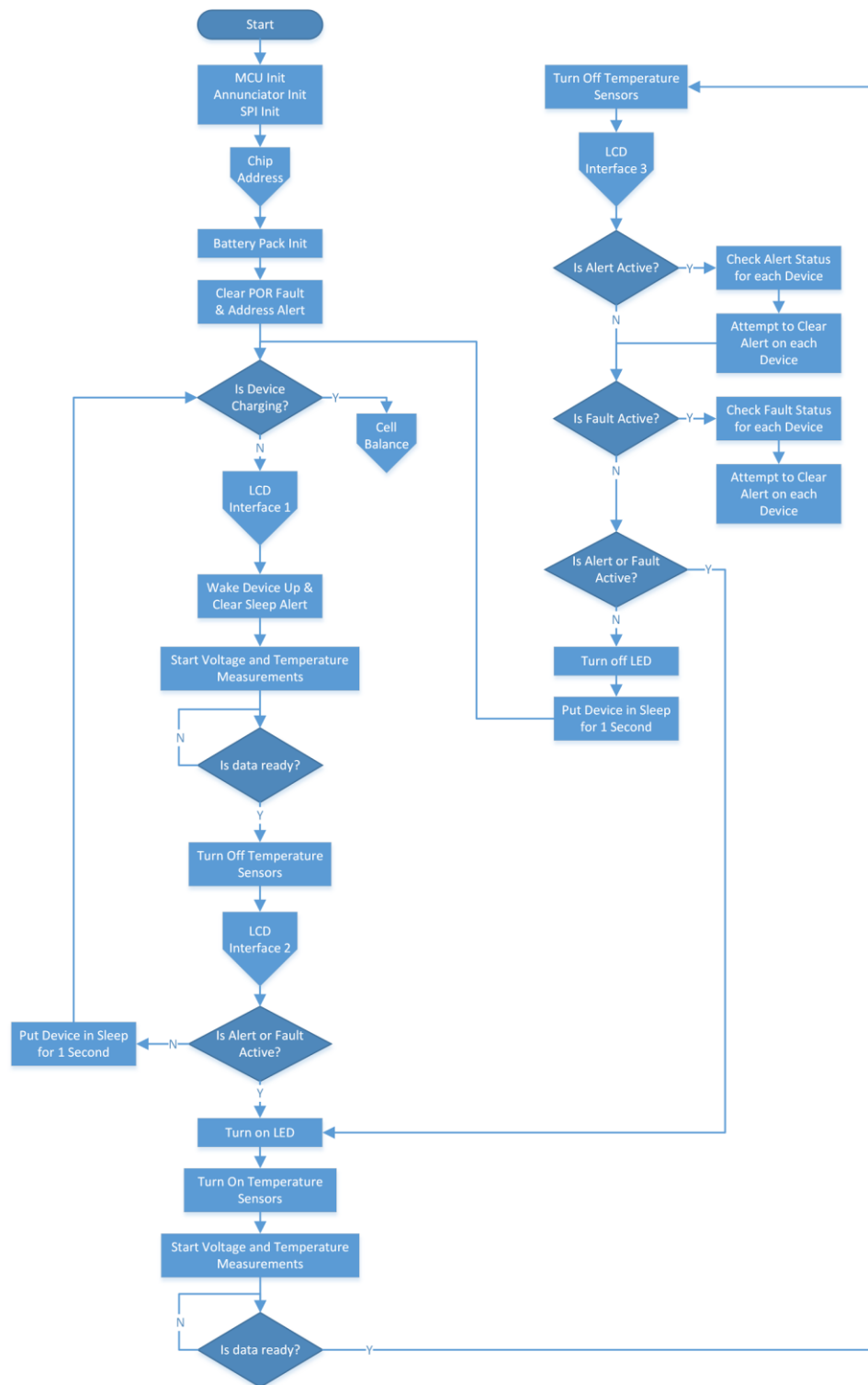


Figure D.1: Main flowchart.

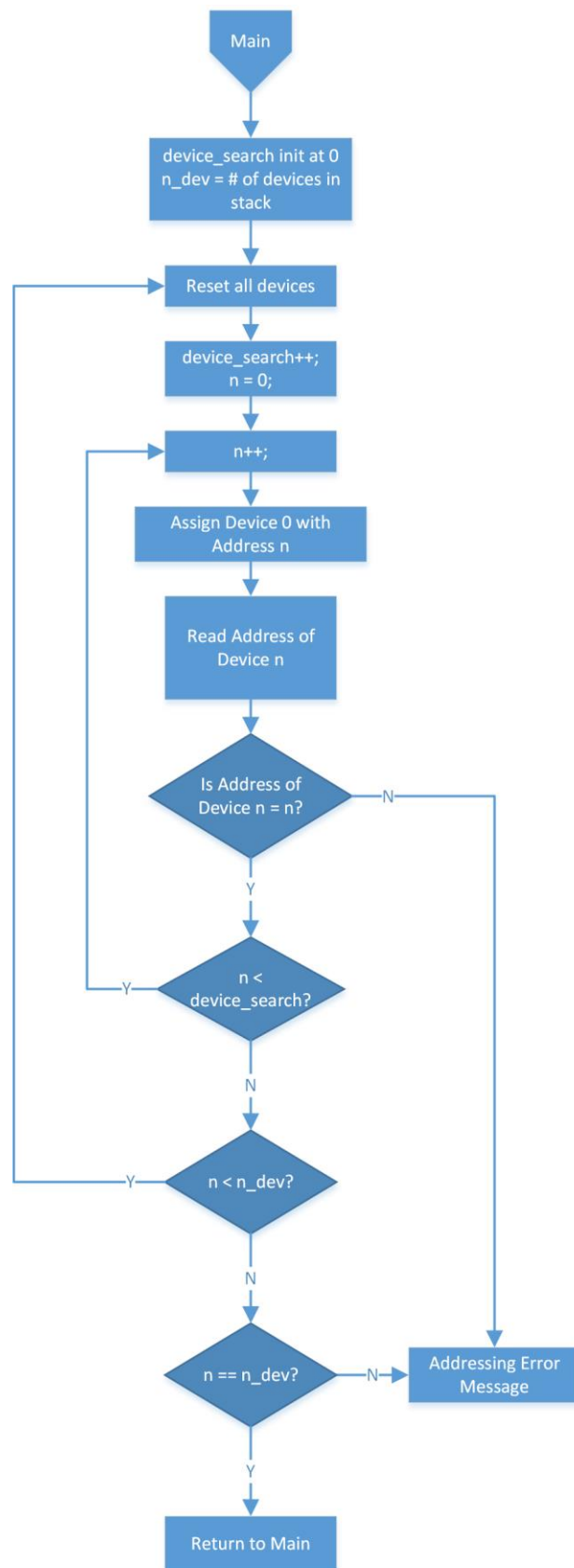


Figure D.2: Flowchart for automatic addressing of bq76PL536 devices.

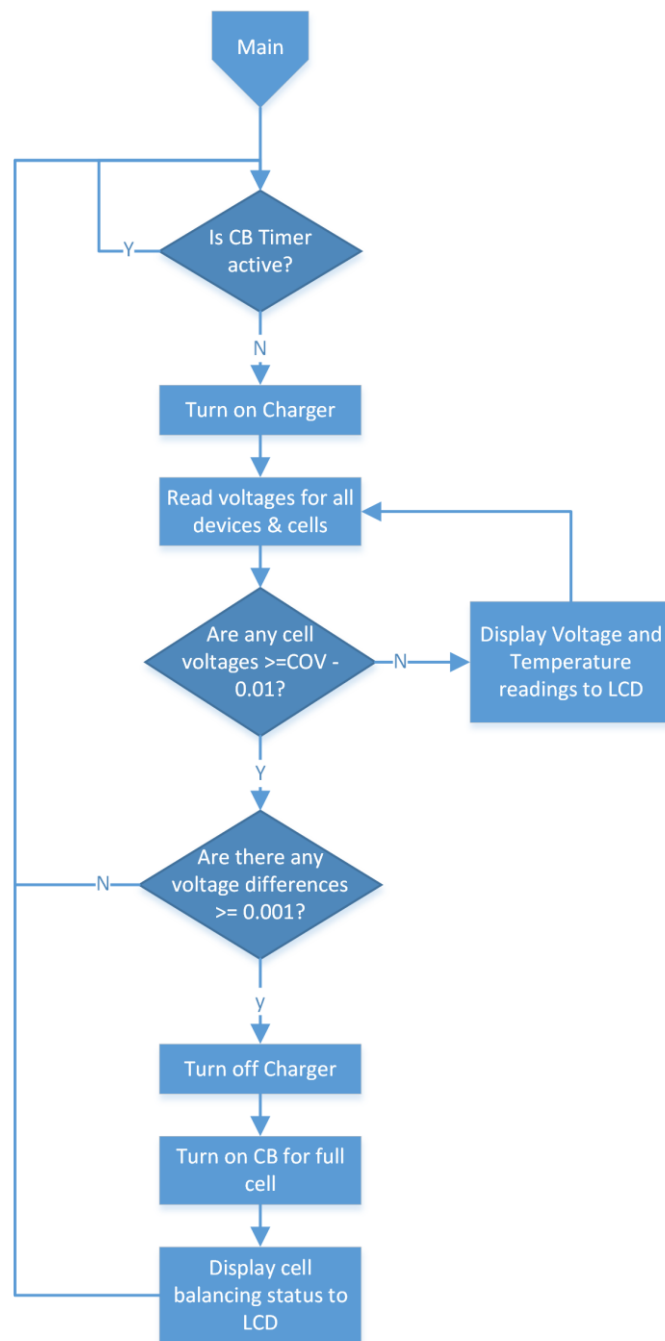


Figure D.3: Flowchart for cell balancing.

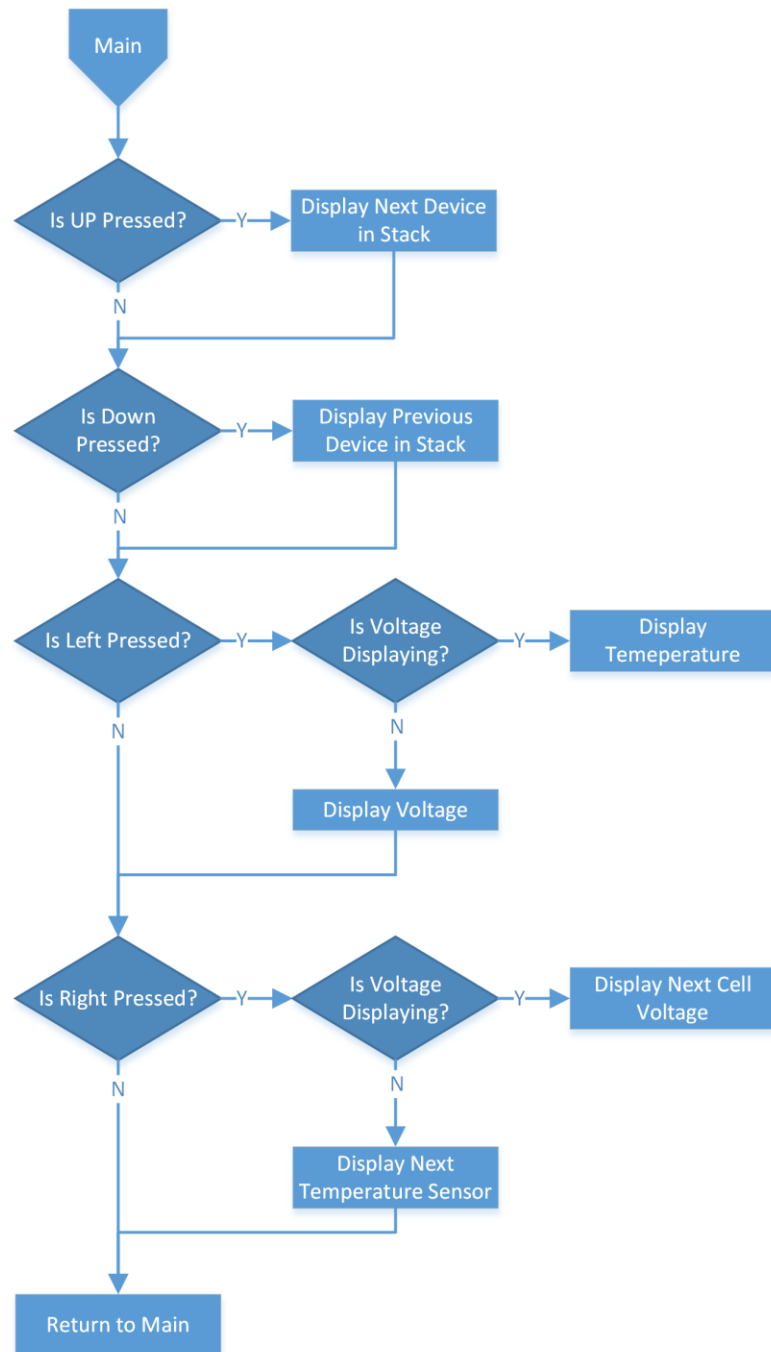


Figure D.4: Flowchart for interfacing with the pushbuttons and LCD.

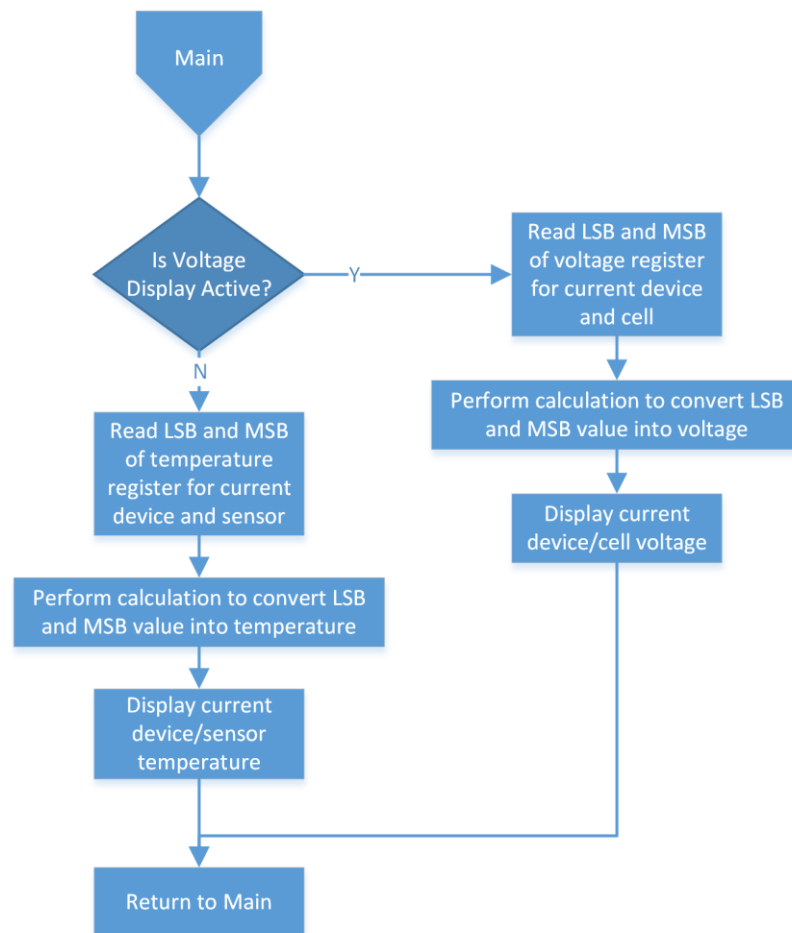


Figure D.5: Flowchart for interfacing with the LCD to display temperature and voltage values.

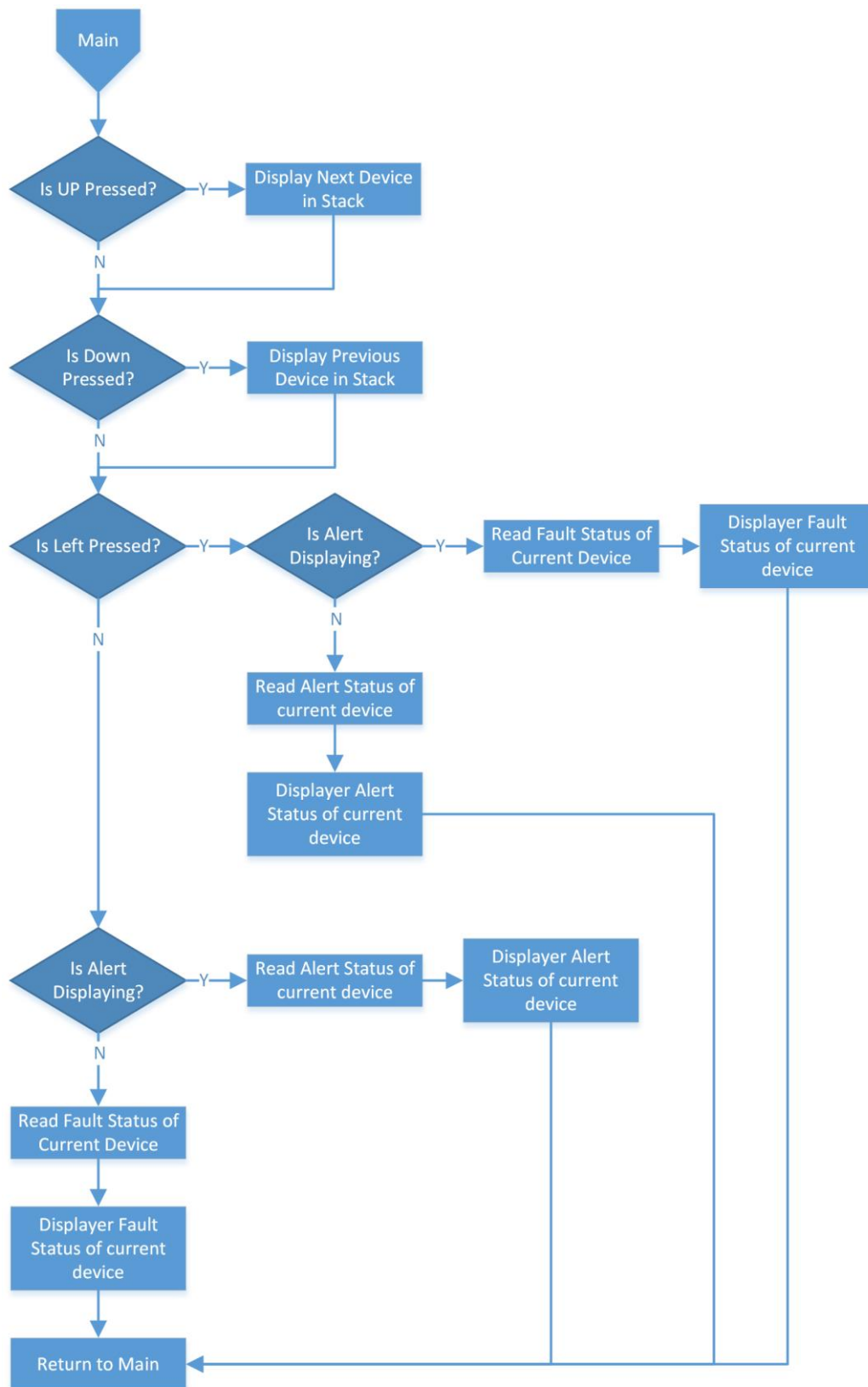


Figure D.6: Flowchart for interfacing with the LCD to display alerts and faults.

Appendix E: Project Code

```
/******Code for Battery Management System based on
bq76PL536A-Q1 platform.Communication between host and chip is done via
SPI.This code also displays messages through the LCD shields and
interactsthrough the buttons on the keypad.This version is basic
functional BMS with complete HMI but without Cell
Balancing.******/

// include the library code:
#include <SPI.h>
#include <Wire.h>
#include <math.h>
#include <utility/Adafruit_MCP23017.h>
#include <Adafruit_RGBLCDShield.h>

Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();

//Declared Variables
SPISettings settingsA(3000000, MSBFIRST, SPI_MODE1);
const byte Fault_H = 8;           //Fault indication pin from bq76
const byte Alert_H = 7;           //Alert indication pin from bq76
const byte DRDY_H = 6;            //Data ready indication from bq76
const byte CONV_H = 5;            //Conversion start pin for bq76
const byte Charge_Mode = 4;       //Charger status to indicate if
charging mode is active
const byte Charge_Relay = 3;      //Output to toggle charger on or off
const byte slaveSelectPin = 10;   //SPI slave select
const byte LED = 9;               //LED indicator
const byte n_dev = 3;             //Expected number of devices in stack
const byte COV = 0x2B;            //Cell overvoltage (4.15V). 0x00 = 2V,
bit increments increase set point by 50mV.
const byte CUV = 0x12;            //Cell undervoltage (2.5V). 0x00
=0.7V, bit increments increase set point by 100mV.
const byte OT = 0x44;             //Over Temp set for 55C for both OT1
and OT2.
const byte FC = 0x20;             //Set ratiometric mode for GPAI_REF
const byte OV = 4150;             //Set point for Cell balancing

byte w_data[3];                   //Write packet
byte r_header[3];                 //Read packet header
byte r_header2[3];                //Alternate read packet header
byte drdy;                        //Data ready
byte CRC;                          //CRC
byte read_byte;                   //Shifted byte to read
byte write_byte;                  //Shifted byte to write
byte RegValue;                   //Value read from register
byte dev_fault;                  //Fault Code of current device
byte dev_alert;                  //Alert Code of current device
uint8_t buttons;                 //Read out from LCD pushbuttons
byte vt_state = 0;                //State to display either temperature
or voltage on LCD. 0 = Volts, 1 = Temp.
byte af_state = 0;                //State to display either alert or
fault on LCD. 0 = Alert, 1 = Fault.
byte dev_now = 1;                 //Current selected device for display.
1 to n_dev
```

```

byte cell_now = 0x03;           //Current selected cell reg address,
must increment by 2. 0x03 to 0x0E
byte cell_num = 1;             //Current selected cell number
byte ts_now = 0x0f;           //Current selected temperature sensor,
must increment by 2. 0x0F to 0x1F
byte ts_num = 1;               //Current selected temperature sensor
byte alert_now = 7;            //Alert bit status to be displayed
byte fault_now = 7;            //Fault bit status to be displayed
byte v_msb;                    //MSB of volt measurement
byte v_lsb;                    //LSB of volt measurement
int volts;                     //Volts calculated
byte ts_msb;                   //MSB of temperature measurement
byte ts_lsb;                   //LSB of temperature measurement
int temp;                      //Temperature calculated
int CB_Timer;                  //Count for number of CB Timers active
byte volts_array[6 * n_dev];   //Array to store all voltages

//Functions:
void read_header(byte r_header[]);
byte read_data();
byte read_CRC();
void write_reg(byte w_data[]);
byte function_CRC(byte data[]);
void reset_all();
void auto_addr(int n_dev);
byte read_shift(byte d_addr);
byte write_shift(byte d_addr);
void batt_init(byte COV, byte CUV, byte OT, byte FC);
void conv(byte adc_ctrl[]);
int volt_calc(byte v_msb, byte v_lsb);
int temp_calc(byte ts_msb, byte ts_lsb, byte ts_num);

//CRC LUT:
byte CrcTable[] = {
    0x00, 0x07, 0x0E, 0x09, 0x1C, 0x1B, 0x12, 0x15,
    0x38, 0x3F, 0x36, 0x31, 0x24, 0x23, 0x2A, 0x2D,
    0x70, 0x77, 0x7E, 0x79, 0x6C, 0x6B, 0x62, 0x65,
    0x48, 0x4F, 0x46, 0x41, 0x54, 0x53, 0x5A, 0x5D,
    0xE0, 0xE7, 0xEE, 0xE9, 0xFC, 0xFB, 0xF2, 0xF5,
    0xD8, 0xDF, 0xD6, 0xD1, 0xC4, 0xC3, 0xCA, 0xCD,
    0x90, 0x97, 0x9E, 0x99, 0x8C, 0x8B, 0x82, 0x85,
    0xA8, 0xAF, 0xA6, 0xA1, 0xB4, 0xB3, 0xBA, 0xBD,
    0xC7, 0xC0, 0xC9, 0xCE, 0xDB, 0xDC, 0xD5, 0xD2,
    0xFF, 0xF8, 0xF1, 0xF6, 0xE3, 0xE4, 0xED, 0xEA,
    0xB7, 0xB0, 0xB9, 0xBE, 0xAB, 0xAC, 0xA5, 0xA2,
    0x8F, 0x88, 0x81, 0x86, 0x93, 0x94, 0x9D, 0x9A,
    0x27, 0x20, 0x29, 0x2E, 0x3B, 0x3C, 0x35, 0x32,
    0x1F, 0x18, 0x11, 0x16, 0x03, 0x04, 0x0D, 0x0A,
    0x57, 0x50, 0x59, 0x5E, 0x4B, 0x4C, 0x45, 0x42,
    0x6F, 0x68, 0x61, 0x66, 0x73, 0x74, 0x7D, 0x7A,
    0x89, 0x8E, 0x87, 0x80, 0x95, 0x92, 0x9B, 0x9C,
    0xB1, 0xB6, 0xBF, 0xB8, 0xAD, 0xAA, 0xA3, 0xA4,
    0xF9, 0xFE, 0xF7, 0xF0, 0xE5, 0xE2, 0xEB, 0xEC,
    0xC1, 0xC6, 0xCF, 0xC8, 0xDD, 0xDA, 0xD3, 0xD4,
    0x69, 0x6E, 0x67, 0x60, 0x75, 0x72, 0x7B, 0x7C,
    0x51, 0x56, 0x5F, 0x58, 0x4D, 0x4A, 0x43, 0x44,
    0x19, 0x1E, 0x17, 0x10, 0x05, 0x02, 0x0B, 0x0C,
    0x21, 0x26, 0x2F, 0x28, 0x3D, 0x3A, 0x33, 0x34,

```

```

0x4E, 0x49, 0x40, 0x47, 0x52, 0x55, 0x5C, 0x5B,
0x76, 0x71, 0x78, 0x7F, 0x6A, 0x6D, 0x64, 0x63,
0x3E, 0x39, 0x30, 0x37, 0x22, 0x25, 0x2C, 0x2B,
0x06, 0x01, 0x08, 0x0F, 0x1A, 0x1D, 0x14, 0x13,
0xAE, 0xA9, 0xA0, 0xA7, 0xB2, 0xB5, 0xBC, 0xBB,
0x96, 0x91, 0x98, 0x9F, 0x8A, 0x8D, 0x84, 0x83,
0xDE, 0xD9, 0xD0, 0xD7, 0xC2, 0xC5, 0xCC, 0xCB,
0xE6, 0xE1, 0xE8, 0xEF, 0xFA, 0xFD, 0xF4, 0xF3
};

void setup() {

    //MCU Initialization START
    pinMode(Fault_H, INPUT);           // Fault_H
    pinMode(Alert_H, INPUT);           // Alert_H
    pinMode(DRDY_H, INPUT);             // DRDY_H (Data Ready)
    pinMode(CONV_H, OUTPUT);            // CONV_H (Conversion)
    pinMode(Charge_Mode, INPUT);        // Input to determine if in
charging mode
    pinMode(Charge_Relay, OUTPUT);      // Output to turn charger on or off
    //MCU Initialization END

    //Annunciator Initialization START
    pinMode(LED, OUTPUT);               // LED output
    Serial.begin(9600);                 // Debugging output
    lcd.begin(16, 2);                   // set up the LCD's number of columns and
rows
    lcd.setBacklight(0x7);              // Set LCD backlight colour to white
    //Annunciator Initialization END

    //SPI Initialization START
    pinMode(slaveSelectPin, OUTPUT);    //set the slaveSelectPin as an
output
    digitalWrite(slaveSelectPin, HIGH); //Set CS_H HIGH
    SPI.begin();                        //Initializes SPI bus
    //SPI Initialization END

    //bq76 Automatic Device Addressing START
    auto_addr(n_dev);
    //bq76 Automatic Device Addressing END

    //Battery Initialization START
    batt_init(COV, CUV, OT, FC);
    //Battery Initialization END

    //Clear POR Fault START
    byte por_clear[] = {0x7F, 0x21, 0x08};
    write_reg(por_clear);
    por_clear[2] = 0x00;
    write_reg(por_clear);
    //Clear POR Fault END

    //Clear Address Alert START
    byte ar_clear[] = {0x7F, 0x20, 0x80};
    write_reg(ar_clear);
    ar_clear[2] = 0x00;
    write_reg(ar_clear);
    //Clear Address Alert END

```



```

}

//////////////////////////////////////////////////
//Main Loop
////////////////////////////////////////////////

void loop() {

    byte charging = digitalRead(Charge_Mode);
    while (charging) {
marker:   while (CB_Timer > 0) {           //While there is
at least one CB-Timer active
        CB_Timer = 0;
        for (int i = 1; i < n_dev + 1; i++) {    //cycle through each
device
            r_header[0] = read_shift(i);          //Device address
            r_header[1] = 0x00;                    //Device status
register
            r_header[2] = 0x01;                    //One byte length
            read_header(r_header);                 //Read device status
from current device
            RegValue = read_data();
            CRC = read_CRC();
            if ((RegValue & 0x02) == 0x02) {        //If CB-Timer is active
                CB_Timer++;                        //Increment count for
number of CB-Timers active
            }
        }
    }
    digitalWrite(Charge_Relay, HIGH);             //Turn on charger

    byte w_wake[] = {0x7F, 0x31, 0x03}; //Wake up from sleep, turn on
temp sensors
    write_reg(w_wake);
    delay (1);                                     //Wait to stabilize the
reference voltage
    w_wake[1] = 0x20;
    w_wake[2] = 0x04;
    write_reg(w_wake);                             //Clear Sleep alert
    w_wake[2] = 0x00;
    write_reg(w_wake);
    byte adc_ctrl[] = {0x7F, 0x30, 0x35}; //Set ADC_Ctrl to read all
voltages and temps
    conv(adc_ctrl);                                //Start conversion
    drdy = digitalRead(DRDY_H);
    while (drdy == 0) {                            //Wait for conversion process
to finish.
        drdy = digitalRead(DRDY_H);
    }
    for (int i = 1; i < n_dev + 1; i++) {          //cycle through each
device
        for (int j = 0x03; j <= 0x0E; j + 0x02) { //cycle through each
cell
            read_byte = read_shift(i);
            byte r_header[] = {read_byte, j, 0x01}; //read from msb of
volts for current cell of current device

```



```

        dev_now = 1;          //roll over current device being monitored
at top
    }
    else {
        dev_now++;           //increase the current device being
monitored
    }
}
if (buttons & BUTTON_DOWN) {
    if (dev_now == 1) {
        dev_now = n_dev;    //roll over current device being monitored
at bottom
    }
    else {
        dev_now--;          //decrease the current device being monitored
    }
}
if (buttons & BUTTON_LEFT) {
    if (vt_state == 0) {
        vt_state = 1;        //change monitoring from voltage to temp
    }
    else {
        vt_state = 0;        //change monitoring from temp to voltage
    }
}
if (buttons & BUTTON_RIGHT) {
    if (vt_state == 0) {      //if currently monitoring volts
        if (cell_num == 6) {
            cell_num = 1;     //roll over current cell
monitored at top
            cell_now = 0x03;   //roll over cell address
        }
        else {
            cell_num++;        //next cell monitored
            cell_now = cell_now + 2; //next cell monitored address
        }
    }
    else {                    //if currently monitoring temp
        if (ts_num == 1) {
            ts_num = 2;        //next temp sensor
            ts_now = 0x11;     //next temp sensor address
        }
        else {
            ts_num = 1;        //next temp sensor
            ts_now = 0x0F;     //next temp sensor address
        }
    }
}
}
w_wake[0] = 0x7F;
w_wake[1] = 0x31;
w_wake[2] = 0x03;           //Wake up from sleep, turn on
temp sensors
write_reg(w_wake);
delay(1);                   //Wait to stabilize the
reference voltage
w_wake[1] = 0x20;
w_wake[2] = 0x04;

```

```

write_reg(w_wake); //Clear Sleep alert
w_wake[2] = 0x00;
write_reg(w_wake);
adc_ctrl[0] = 0x7F;
adc_ctrl[1] = 0x30;
adc_ctrl[3] = 0x35; //Set ADC_Ctrl to read all
voltages and temps
conv(adc_ctrl); //Start conversion
drdy = digitalRead(DRDY_H);
while (drdy == 0) { //Wait for conversion process
to finish.
    drdy = digitalRead(DRDY_H);
}

if (vt_state == 0) { //Display voltage
    read_byte = read_shift(dev_now);
    byte r_header[] = {read_byte, cell_now, 0x01}; //read from msb of
volts for current cell of current device
    read_header(r_header);
    v_msb = read_data();
    CRC = read_CRC();

    byte r_header2[] = {read_byte, cell_now + 1, 0x01}; //read from
lsb of volts for current cell of current device
    read_header(r_header2);
    v_lsb = read_data();
    CRC = read_CRC();

    volts = volt_calc(v_msb, v_lsb); //Calculate voltage
in mV

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("D");
    lcd.setCursor(1, 0);
    lcd.print(dev_now);
    lcd.setCursor(3, 0);
    lcd.print("Cell");
    lcd.setCursor(7, 0);
    lcd.print(cell_num);
    lcd.setCursor(8, 0);
    lcd.print(":");
    lcd.setCursor(10, 0);
    lcd.print(volts);
    lcd.setCursor(14, 0);
    lcd.print("mV");
    lcd.setCursor(0, 1);
    lcd.print("<Temp");
    lcd.setCursor(11, 1);
    lcd.print("Next>");
}
else { //Display Temp
    read_byte = read_shift(dev_now);
    byte r_header[] = {read_byte, ts_now, 0x01}; //read from msb of
temperature sensor of current device
    read_header(r_header);
    ts_msb = read_data();
    CRC = read_CRC();

```

```

    byte r_header2[] = {read_byte, ts_now + 1, 0x01}; //read from
lsb of temperature sensor of current device
    read_header(r_header2);
    ts_lsb = read_data();
    CRC = read_CRC();

    temp = temp_calc(ts_msb, ts_lsb, ts_num); //Calculate temperature

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("D");
    lcd.setCursor(1, 0);
    lcd.print(dev_now);
    lcd.setCursor(3, 0);
    lcd.print("TS");
    lcd.setCursor(5, 0);
    lcd.print(ts_num);
    lcd.setCursor(6, 0);
    lcd.print(":");
    lcd.setCursor(8, 0);
    lcd.print(temp);
    lcd.setCursor(11, 0);
    lcd.print("C");
    lcd.setCursor(0, 1);
    lcd.print("<Volts");
    lcd.setCursor(11, 1);
    lcd.print("Next>");
}

byte alert = digitalRead(Alert_H);
byte fault = digitalRead(Fault_H);

if (alert || fault) {
    digitalWrite(LED, HIGH); //Turn on LED
    while (alert || fault) { //Wait for alerts or faults to
clear
        byte w_wake[] = {0x7F, 0x31, 0x03}; //Turn on temp sensors
        write_reg(w_wake);
        delay(1); //Wait to stabilize the reference voltage
        byte adc_ctrl[] = {0x7F, 0x30, 0x35}; //Set ADC_Ctrl to read
all voltages and temps
        conv(adc_ctrl); //Start conversion
        drdy = digitalRead(DRDY_H);
        while (drdy == 0) //Wait for conversion process to finish.
        {
            drdy = digitalRead(DRDY_H);
        }
        w_wake[2] = 0x00; //turn off temp sensors
        write_reg(w_wake);

        buttons = lcd.readButtons();
        if (buttons) {
            if (buttons & BUTTON_UP) {
                if (dev_now == n_dev) {
                    dev_now = 1; //roll over current device being
monitored at top
                }
            }
        }
    }
}

```

```

        else {
            dev_now++;          //increase the current device being
monitored
        }
    }
    if (buttons & BUTTON_DOWN) {
        if (dev_now == 1) {
            dev_now = n_dev;    //roll over current device being
monitored at bottom
        }
        else {
            dev_now--;          //decrease the current device being
monitored
        }
    }
    if (buttons & BUTTON_LEFT) {
        if (af_state == 0) {
            af_state = 1;        //change display from alert to
fault status
        }
        else {
            af_state = 0;        //change display from fault to
alert status
        }
    }
}

//Display Alert/Fault Status:
if (af_state == 0) {           //Display Alert Status
    if (alert) {               //If the device has an
alert, read which alert
        r_header[1] = 0x20;    //Alert status register
        read_header(r_header); //Read device status from
current device
        dev_alert = read_data(); //Store alert code in
dev_alert
        CRC = read_CRC();
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Alert:");
        lcd.setCursor(7, 0);
        lcd.print(dev_alert, BIN);
        lcd.setCursor(0, 1);
        lcd.print("<Faults");
    }
    else {                     //If the device does not
have an alert
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("No Alerts");
        lcd.setCursor(0, 1);
        lcd.print("<Faults");
    }
}
else {                         //Display Fault Status
    if (fault) {               //If the device has a
fault, read which fault
        r_header[1] = 0x21;    //Fault status register

```

```

        read_header(r_header);           //Read device status from
current device
        dev_fault = read_data();         //Store fault code in
dev_fault
        CRC = read_CRC();
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Fault:");
        lcd.setCursor(7, 0);
        lcd.print(dev_fault, BIN);
        lcd.setCursor(0, 1);
        lcd.print("<Alerts");
    }
    else {                               //If the device does not
have a fault
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("No Faults");
        lcd.setCursor(0, 1);
        lcd.print("<Alerts");
    }
}

//Attempt to clear alerts
if (alert) {                             //if an alert is
active on one of the devices
    for (int i = 1; i < n_dev + 1; i++) { //cycle through
each device
        r_header[0] = read_shift(i);     //Device address
        r_header[1] = 0x00;              //Device status
register
        r_header[2] = 0x01;              //One byte length
        read_header(r_header);           //Read device
status from current device
        RegValue = read_data();
        CRC = read_CRC();
        if ((RegValue & 0x20) == 0x20) { //If the device has
an alert, read which alert
            r_header[1] = 0x20;          //Alert status
register
            read_header(r_header);       //Read alert status
from current device
            dev_alert= read_data();       //Store alert code
in dev_alert
            CRC = read_CRC();
            w_data[0] = write_shift(i);  //Device address
            w_data[1] = 0x20;            //Alert status
register
            w_data[2] = dev_alert;        //Write 1 to alert
bits
            write_reg(w_data);
            w_data[2] = 0x00;            //Attempt to clear
alerts
            write_reg(w_data);
        }
    }
}
//Attempt to clear faults

```

```

        if (fault) { //if a fault is
active on one of the devices
        for (int i = 1; i < n_dev + 1; i++) { //cycle through
each device
            r_header[0] = read_shift(i); //Device address
            r_header[1] = 0x00; //Device status
register
            r_header[2] = 0x01; //One byte length
            read_header(r_header); //Read device
status from current device
            RegValue = read_data();
            CRC = read_CRC();
            if ((RegValue & 0x40) == 0x40) { //If the device has
a fault, read which fault
                r_header[1] = 0x21; //Fault status
register
                read_header(r_header); //Read fault status
from current device
                dev_fault = read_data(); //Store fault code
in dev_fault
                CRC = read_CRC();
                w_data[0] = write_shift(i); //Device address
                w_data[1] = 0x21; //Fault status
register
                w_data[2] = dev_fault; //Write 1 to fault
bits
                write_reg(w_data);
                w_data[2] = 0x00; //Attempt to clear
faults
                write_reg(w_data);
            }
        }
    }

    byte alert = digitalRead(Alert_H);
    byte fault = digitalRead(Fault_H);
    delay(1000);
}
digitalWrite(LED, LOW);
}

//Put Device in Sleep Mode
byte w_sleep[] = {0x7F, 0x31, 0x04};
write_reg(w_sleep);
delay(1000); //Put the devices to sleep for 1s
byte charging = digitalRead(Charge_Mode);
}

buttons = lcd.readButtons();
if (buttons) {
    if (buttons & BUTTON_UP) {
        if (dev_now == n_dev) {
            dev_now = 1; //roll over current device being monitored at
top
        }
        else {
            dev_now++; //increase the current device being monitored

```



```

    }
}
if (buttons & BUTTON_DOWN) {
    if (dev_now == 1) {
        dev_now = n_dev; //roll over current device being monitored at
bottom
    }
    else {
        dev_now--; //decrease the current device being monitored
    }
}
if (buttons & BUTTON_LEFT) {
    if (vt_state == 0) {
        vt_state = 1; //change monitoring from voltage to temp
    }
    else {
        vt_state = 0; //change monitoring from temp to voltage
    }
}
if (buttons & BUTTON_RIGHT) {
    if (vt_state == 0) { //if currently monitoring volts
        if (cell_num == 6) {
            cell_num = 1; //roll over current cell
monitored at top
            cell_now = 0x03; //roll over cell address
        }
        else {
            cell_num++; //next cell monitored
            cell_now = cell_now + 2; //next cell monitored address
        }
    }
    else { //if currently monitoring temp
        if (ts_num == 1) {
            ts_num = 2; //next temp sensor
            ts_now = 0x11; //next temp sensor address
        }
        else {
            ts_num = 1; //next temp sensor
            ts_now = 0x0F; //next temp sensor address
        }
    }
}
}
byte w_wake[] = {0x7F, 0x31, 0x03}; //Wake up from sleep, turn on
temp sensors
write_reg(w_wake);
delay(1); //Wait to stabilize the reference
voltage
w_wake[1] = 0x20;
w_wake[2] = 0x04;
write_reg(w_wake); //Clear Sleep alert
w_wake[2] = 0x00;
write_reg(w_wake);
byte adc_ctrl[] = {0x7F, 0x30, 0x35}; //Set ADC_Ctrl to read all
voltages and temps
conv(adc_ctrl); //Start conversion
drdy = digitalRead(DRDY_H);

```

```

while (drdy == 0) {                                     //Wait for conversion process
to finish.
    drdy = digitalRead(DRDY_H);
}

if (vt_state == 0) {                                     //Display voltage
    read_byte = read_shift(dev_now);
    byte r_header[] = {read_byte, cell_now, 0x01}; //read from msb of
volts for current cell of current device
    read_header(r_header);
    v_msb = read_data();
    CRC = read_CRC();

    byte r_header2[] = {read_byte, cell_now + 1, 0x01}; //read from
lsb of volts for current cell of current device
    read_header(r_header2);
    v_lsb = read_data();
    CRC = read_CRC();

    volts = volt_calc(v_msb, v_lsb);                     //Calculate voltage
in mV

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("D");
    lcd.setCursor(1, 0);
    lcd.print(dev_now);
    lcd.setCursor(3, 0);
    lcd.print("Cell");
    lcd.setCursor(7, 0);
    lcd.print(cell_num);
    lcd.setCursor(8, 0);
    lcd.print(":");
    lcd.setCursor(10, 0);
    lcd.print(volts);
    lcd.setCursor(14, 0);
    lcd.print("mV");
    lcd.setCursor(0, 1);
    lcd.print("<Temp");
    lcd.setCursor(11, 1);
    lcd.print("Next>");
}
else {           //Display Temp
    read_byte = read_shift(dev_now);
    byte r_header[] = {read_byte, ts_now, 0x01}; //read from msb of
temperature sensor of current device
    read_header(r_header);
    ts_msb = read_data();
    CRC = read_CRC();
    byte r_header2[] = {read_byte, ts_now + 1, 0x01}; //read from lsb
of temperature sensor of current device
    read_header(r_header2);
    ts_lsb = read_data();
    CRC = read_CRC();

    temp = temp_calc(ts_msb, ts_lsb, ts_num); //Calculate temperature

    lcd.clear();

```

```

    lcd.setCursor(0, 0);
    lcd.print("D");
    lcd.setCursor(1, 0);
    lcd.print(dev_now);
    lcd.setCursor(3, 0);
    lcd.print("TS");
    lcd.setCursor(5, 0);
    lcd.print(ts_num);
    lcd.setCursor(6, 0);
    lcd.print(":");
    lcd.setCursor(8, 0);
    lcd.print(temp);
    lcd.setCursor(11, 0);
    lcd.print("C");
    lcd.setCursor(0, 1);
    lcd.print("<Volts");
    lcd.setCursor(11, 1);
    lcd.print("Next>");
}

byte alert = digitalRead(Alert_H);
byte fault = digitalRead(Fault_H);
if (alert || fault) {
    digitalWrite(LED, HIGH);          //Turn on LED
    while (alert || fault) {          //Wait for alerts or faults to
clear
        byte w_wake[] = {0x7F, 0x31, 0x03}; //Turn on temp sensors
        write_reg(w_wake);
        delay(1); //Wait to stabilize the reference voltage
        byte adc_ctrl[] = {0x7F, 0x30, 0x35}; //Set ADC_Ctrl to read all
voltages and temps
        conv(adc_ctrl); //Start conversion
        drdy = digitalRead(DRDY_H);
        while (drdy == 0) //Wait for conversion process to finish.
        {
            drdy = digitalRead(DRDY_H);
        }

        buttons = lcd.readButtons();
        if (buttons) {
            if (buttons & BUTTON_UP) {
                if (dev_now == n_dev) {
                    dev_now = 1; //roll over current device being
monitored at top
                }
            }
            else {
                dev_now++; //increase the current device being
monitored
            }
        }
        if (buttons & BUTTON_DOWN) {
            if (dev_now == 1) {
                dev_now = n_dev; //roll over current device being
monitored at bottom
            }
            else {
                dev_now--; //decrease the current device being
monitored
            }
        }
    }
}

```

```

    }
}
if (buttons & BUTTON_LEFT) {
    if (af_state == 0) {
        af_state = 1;                //change display from alert to
fault status
    }
    else {
        af_state = 0;                //change display from fault to
alert status
    }
}
}
//Display Alert/Fault Status:
if (af_state == 0) {                //Display Alert Status
    if (alert) {                    //If the device has an
alert, read which alert
        r_header[1] = 0x20;        //Alert status register
        read_header(r_header);    //Read device status from
current device
        dev_alert = read_data();  //Store alert code in
dev_alert
        CRC = read_CRC();
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Alert:");
        lcd.setCursor(7, 0);
        lcd.print(dev_alert, BIN);
        lcd.setCursor(0, 1);
        lcd.print("<Faults");
    }
    else {                          //If the device does not
have an alert
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("No Alerts");
        lcd.setCursor(0, 1);
        lcd.print("<Faults");
    }
}
else {                              //Display Fault Status
    if (fault) {                    //If the device has a
fault, read which fault
        r_header[1] = 0x21;        //Fault status register
        read_header(r_header);    //Read device status from
current device
        dev_fault = read_data();  //Store fault code in
dev_fault
        CRC = read_CRC();
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Fault:");
        lcd.setCursor(7, 0);
        lcd.print(dev_fault, BIN);
        lcd.setCursor(0, 1);
        lcd.print("<Alerts");
    }
}

```

```

        else {
            //If the device does not
            have a fault
                lcd.clear();
                lcd.setCursor(0, 0);
                lcd.print("No Faults");
                lcd.setCursor(0, 1);
                lcd.print("<Alerts");
        }
    }

    //Attempt to clear alerts
    if (alert) {
        //if an alert is
        active on one of the devices
        for (int i = 1; i < n_dev + 1; i++) {
            //cycle through each
            device
                r_header[0] = read_shift(i);
                r_header[1] = 0x00;
            //Device address
            //Device status
            register
                r_header[2] = 0x01;
                read_header(r_header);
            //One byte length
            //Read device status
            from current device
                RegValue = read_data();
                CRC = read_CRC();
            //If the device has
            an alert, read which alert
                r_header[1] = 0x20;
            //Alert status
            register
                read_header(r_header);
            //Read alert status
            from current device
                dev_alert = read_data();
            //Store alert code in
            dev_alert
                CRC = read_CRC();
                w_data[0] = write_shift(i);
                w_data[1] = 0x20;
            //Device address
            //Alert status
            register
                w_data[2] = dev_alert;
            //Write 1 to alert
            bits
                write_reg(w_data);
                w_data[2] = 0x00;
            //Attempt to clear
            alerts
                write_reg(w_data);
        }
    }

    //Attempt to clear faults
    if (fault) {
        //if a fault is
        active on one of the devices
        for (int i = 1; i < n_dev + 1; i++) {
            //cycle through each
            device
                r_header[0] = read_shift(i);
                r_header[1] = 0x00;
            //Device address
            //Device status
            register
                r_header[2] = 0x01;
                read_header(r_header);
            //One byte length
            //Read device status
            from current device
                RegValue = read_data();
                CRC = read_CRC();
        }
    }

```

```

        if ((RegValue & 0x40) == 0x40) { //If the device has a
fault, read which fault
        r_header[1] = 0x21; //Fault status
register
        read_header(r_header); //Read fault status
from current device
        dev_fault = read_data(); //Store fault code in
dev_fault
        CRC = read_CRC();
        w_data[0] = write_shift(i); //Device address
        w_data[1] = 0x21; //Fault status
register
        w_data[2] = dev_fault; //Write 1 to fault
bits
        write_reg(w_data);
        w_data[2] = 0x00; //Attempt to clear
faults
        write_reg(w_data);
    }
}

    byte alert = digitalRead(Alert_H);
    byte fault = digitalRead(Fault_H);
    delay(1000);
}
digitalWrite(LED, LOW);
}

//Put Device in Sleep Mode
byte w_sleep[] = {0x7F, 0x31, 0x04};
write_reg(w_sleep);
delay(1000); //Put the devices to sleep for 1s
}

//Function to establish device address, register address, and length
for reading
void read_header(byte r_header[]) {
    digitalWrite(slaveSelectPin, LOW); //Set CS_H LOW to start packet
transfer
    SPI.beginTransaction(settingsA);
    for (int i = 0; i < 3; i++)
    {
        SPI.transfer(r_header[i]); //Set device address, read, register
address, & length
    }
}

//Function to clock in a read cycle and return the read data
byte read_data() {
    byte RegValue;
    RegValue = SPI.transfer(0x00);
    return RegValue;
}

//Function to clock in a CRC cycle and return CRC value
byte read_CRC() {
    byte CRC;

```

```

    CRC = SPI.transfer(0x00);
    SPI.endTransaction();
    digitalWrite(slaveSelectPin, HIGH); //Set CS_H HIGH to end packet
transfer
    return CRC;
}

//Function to write to bq71
void write_reg(byte w_data[]) {
    CRC = function_CRC(w_data);
    digitalWrite(slaveSelectPin, LOW); //Set CS_H LOW to start packet
transfer
    SPI.beginTransaction(settingsA);
    for (int i = 0; i < 3; i++)
    {
        SPI.transfer(w_data[i]); //Set device address, write, register
address, & write data
    }
    SPI.transfer(CRC); // CRC Code
    SPI.endTransaction();
    digitalWrite(slaveSelectPin, HIGH); //Set CS_H HIGH to end packet
transfer
}

//Function to generate a CRC
byte function_CRC(byte data[])
{
    byte crc = 0;
    int temp = 0;
    for (int i = 0; i < sizeof(data) + 1; i++)
    {
        temp = crc ^ data[i];
        crc = CrcTable[temp];
    }
    return crc;
}

//Broadcast RESET to all devices
void reset_all()
{
    byte w_data[3] = {0x7F, 0x3C, 0xA5};
    write_reg(w_data);
}

//Automatically address all devices in stack
void auto_addr(byte n_dev)
{
    byte device_search = 0;
point1: reset_all();
    device_search++;
    byte n = 0;
point2: n++;
    byte w_data[3] = {0x01, 0x3B, n}; //header for write address n to
dev[0]
    write_reg(w_data);
    read_byte = read_shift(n);
    byte r_header[3] = {read_byte, 0x3B, 0x01}; //header to read address
from address_ctrl register

```

```

    read_header(r_header);
    RegValue = read_data();
    CRC = read_CRC();
    RegValue = RegValue % 128;
    while (RegValue != n) { //Device will be stuck in this loop if
addressing fails. Requires POR to try again. Device addr critical.
        lcd.setCursor(0, 0);
        lcd.print("Addr Failed on D");
        lcd.setCursor(0, 1);
        lcd.print(n);
    }
    if (n < device_search) {
        goto point2;
    }
    if (n < n_dev) {
        goto point1;
    }
    while (n != n_dev) {
        lcd.setCursor(0, 0);
        lcd.print("Automatic Device");
        lcd.setCursor(0, 1);
        lcd.print("Address Failed 2");
    }
}

// Shift device address left 1 bit
byte read_shift(byte d_addr)
{
    byte read_byte = (d_addr << 1);
    return read_byte;
}

// Shift device address left 1 bit then add 1 to indicate write
byte write_shift(byte d_addr)
{
    byte write_byte = (d_addr << 1) + 1;
    return write_byte;
}

//Overwrite EPROM values using shadow control special sequence. Write
0x35 to shdw_ctrl reg followed by set point.
void batt_init(byte COV, byte CUV, byte OT, byte FC)
{
    byte shdw_ctrl[3] = {0x7F, 0x3A, 0x35};
    //Set Special feature config:
    write_reg(shdw_ctrl);
    byte w_func_con[] = {0x7F, 0x40, FC};
    write_reg(w_func_con);
    //Set Overvoltage:
    write_reg(shdw_ctrl);
    byte w_COV[3] = {0x7F, 0x42, COV};
    write_reg(w_COV);
    //Set Undervoltage:
    write_reg(shdw_ctrl);
    byte w_CUV[3] = {0x7F, 0x44, CUV};
    write_reg(w_CUV);
    //Set Over Temperature:
    write_reg(shdw_ctrl);

```



```

    byte w_OT[3] = {0x7F, 0x46, OT};
    write_reg(w_OT);
}

//Function to assert a conversion process
void conv(byte adc_ctrl[])
{
    byte conv_start[] = {0x7F, 0x34, 0x01};
    write_reg(adc_ctrl); //Sets ADC control register with desired
parameters
    write_reg(conv_start); //Start conversion
}

//Function to calculate voltage using register values
int volt_calc(byte v_msb, byte v_lsb)
{ \
    float ratio = 0.381493;
    int mV = ((v_msb * 0x100) + v_lsb) * ratio;
    return mV;
}

//Function to calculate temperature using register values
int temp_calc(byte ts_msb, byte ts_lsb, byte ts_num)
{
    float divisor1 = 0.030261;
    float divisor2 = 0.030241;
    float A1 = 33.540164;
    float A2 = 3.001308;
    float A3 = 0.050852;
    float A4 = 0.002188;
    float p1;
    float p2;
    float p3;
    int R25 = 10000;
    int RTS;
    int Rth;
    float nlog;
    float temp;
    if (ts_num == 1) {
        RTS = ((ts_msb * 256) + ts_lsb + 2) * divisor1;
        Rth = (1820000 / RTS) - 1470 - 1820;
        nlog = log((float)Rth / (float)R25) * 10000;
        p1 = A2 * (nlog / 10000);
        p2 = A3 * pow(nlog / 10000, 2);
        p3 = A4 * pow(nlog / 10000, 3);
        temp = 10000 / (A1 + p1 + p2 + p3) - 273.15;
    }
    else {
        RTS = ((ts_msb * 256) + ts_lsb + 9) * divisor2;
        Rth = (1820000 / RTS) - 1470 - 1820;
        nlog = log((float)Rth / (float)R25) * 10000;
        p1 = A2 * (nlog / 10000);
        p2 = A3 * pow(nlog / 10000, 2);
        p3 = A4 * pow(nlog / 10000, 3);
        temp = 10000 / (A1 + p1 + p2 + p3) - 273.15;
    }
    return temp;
}

```


Appendix F: Test Results

Table F.1: Cell voltages in comparison to voltages displayed by the BMS LCD.

Test #	C1 [mV]	BMS1 [mV]	C2 [mV]	BMS2 [mV]	C3 [mV]	BMS3 [mV]	C4 [mV]	BMS4 [mV]	C5 [mV]	BMS5 [mV]	C6 [mV]	BMS6 [mV]	Accuracy [mV]
1	3139	3142	3055	3058	3099	3101	3257	3259	3346	3345	3465	3463	1.17
	3119	3119	3022	3024	3110	3111	3221	3220	3271	3270	3460	3457	0.33
	2974	2975	2954	2956	3011	3014	3121	3122	3194	3195	3372	3372	1.33
2	2999	3000	2915	2915	2963	2964	3131	3133	3235	3236	3385	3384	0.67
	3080	3080	2984	2985	3070	3071	3181	3182	3242	3241	3437	3434	0.17
	2970	2972	2953	2955	3011	3013	3119	3120	3191	3193	3367	3368	1.67
3	2928	2930	2851	2853	2905	2907	3073	3074	3186	3187	3352	3351	1.17
	3089	3091	2994	2995	3080	3080	3189	3190	3249	3248	3442	3441	0.33
	2971	2973	2954	2956	3012	3013	3119	3120	3191	3192	3367	3368	1.33
4	2876	2878	2798	2800	2854	2855	3025	3027	3146	3146	3326	3324	0.83
	3094	3095	2998	3000	3084	3085	3193	3194	3252	3252	3445	3443	0.50
	2972	2974	2954	2956	3012	3014	3120	3122	3192	3194	3368	3369	1.83
5	3025	3028	2943	2945	2992	2993	3162	3161	3262	3264	3402	3402	1.17
	3098	3098	3002	3003	3087	3088	3196	3197	3256	3254	3447	3445	0.17
	2972	2976	2954	2956	3011	3013	3119	3120	3192	3192	3368	3369	1.67
6	2925	2927	2785	2789	2841	2843	3014	3016	3137	3137	3320	3318	1.33
	3104	3105	3007	3008	3093	3097	3261	3260	3452	3450	2974	2972	0.17
	2972	2974	2955	2959	3011	3013	3119	3122	3193	3194	3371	3370	1.83
7	2719	2720	2648	2650	2713	2716	2892	2895	3032	3031	3253	3252	1.17
	3108	3110	3012	3014	3098	3099	3206	3206	3263	3263	3454	3452	0.50
	2971	2974	2953	2956	3010	3012	3119	3120	3191	3192	3369	3370	1.83
8	3065	3065	2982	2984	3029	3031	3195	3196	3290	3291	3423	3422	0.83

	3131	3131	3035	3037	3122	3123	3234	3237	3285	3285	3460	3457	0.50
	2971	2973	2953	2954	3010	3012	3119	3120	3191	3192	3369	3368	1.00
9	3014	3015	2928	2931	2979	2981	3146	3148	3250	3251	3396	3396	1.50
	3134	3135	3039	3040	3124	3124	3228	3228	3284	3283	3469	3467	0.17
	2971	2974	2952	2955	3010	3015	3119	3120	3192	3193	3370	3371	2.33
10	3121	3123	3037	3039	3078	3079	3242	3239	3328	3330	3448	3447	0.50
	3148	3148	3049	3050	3133	3134	3237	3237	3290	3289	3473	3471	0.17
	2972	2975	2952	2955	3010	3012	3120	3122	3192	3194	3370	3370	2.00
11	2771	2773	2697	2699	2759	2761	2937	2937	3072	3072	3281	3280	0.83
	3129	3131	3031	3033	3116	3118	3224	3224	3279	3278	3466	3463	0.33
	2974	2974	2953	2955	3010	3012	3119	3122	3193	3196	3371	3372	1.83
12	3193	3196	3113	3116	3152	3154	3309	3311	3390	3391	3492	3492	1.83
	3136	3137	3039	3041	3124	3125	3242	3244	3287	3286	3472	3470	0.50
	2974	2977	2953	2955	3012	3013	3119	3120	3193	3195	3370	3371	1.67
13	2878	2877	2797	2800	2874	2875	3027	3029	3150	3151	3328	3327	0.83
	3153	3155	3055	3057	3139	3140	3243	3244	3300	3301	3481	3478	0.67
	2972	2975	2953	2956	3010	3012	3119	3120	3193	3193	3369	3370	1.67
14	3583	3586	3504	3507	3523	3525	3658	3660	3696	3698	3725	3724	1.83
	3144	3145	3046	3046	3130	3132	3235	3235	3288	3287	3472	3469	0.17
	2972	2974	2962	2955	3010	3012	3119	3122	3193	3195	3370	3370	0.33
15	3300	3303	3217	3219	3249	3252	3401	3404	3468	3468	3449	3447	1.50
	3291	3287	3217	3216	3297	3297	3372	3372	3412	3412	3566	3565	1.00
	2971	2974	2962	2956	3009	3011	3119	3120	3192	3193	3370	3372	0.50
16	2925	2927	2785	2789	2841	2843	3014	3016	3137	3137	3320	3318	1.33
	3104	3105	3007	3008	3093	3097	3261	3260	3452	3450	2974	2972	0.17
	2972	2974	2955	2959	3011	3013	3119	3122	3193	3194	3371	3370	1.83
17	3139	3142	3055	3058	3099	3101	3257	3259	3346	3345	3465	3463	1.17

	3119	3119	3022	3024	3110	3111	3221	3220	3271	3270	3460	3457	0.33
	2974	2975	2954	2956	3011	3014	3121	3122	3194	3195	3372	3372	1.33
18	3065	3065	2982	2984	3029	3031	3195	3196	3290	3291	3423	3422	0.83
	3131	3131	3035	3037	3122	3123	3234	3237	3285	3285	3460	3457	0.50
	2971	2973	2953	2954	3010	3012	3119	3120	3191	3192	3369	3368	1.00
19	3014	3015	2928	2931	2979	2981	3146	3148	3250	3251	3396	3396	1.50
	3134	3135	3039	3040	3124	3124	3228	3228	3284	3283	3469	3467	0.17
	2971	2974	2952	2955	3010	3015	3119	3120	3192	3193	3370	3371	2.33
20	3121	3123	3037	3039	3078	3079	3242	3239	3328	3330	3448	3447	0.50
	3148	3148	3049	3050	3133	3134	3237	3237	3290	3289	3473	3471	0.17
	2972	2975	2952	2955	3010	3012	3120	3122	3192	3194	3370	3370	2.00
21	2928	2930	2851	2853	2905	2907	3073	3074	3186	3187	3352	3351	1.17
	3089	3091	2994	2995	3080	3080	3189	3190	3249	3248	3442	3441	0.33
	2971	2973	2954	2956	3012	3013	3119	3120	3191	3192	3367	3368	1.33
22	3193	3196	3113	3116	3152	3154	3309	3311	3390	3391	3492	3492	1.83
	3136	3137	3039	3041	3124	3125	3242	3244	3287	3286	3472	3470	0.50
	2974	2977	2953	2955	3012	3013	3119	3120	3193	3195	3370	3371	1.67
23	2878	2877	2797	2800	2874	2875	3027	3029	3150	3151	3328	3327	0.83
	3153	3155	3055	3057	3139	3140	3243	3244	3300	3301	3481	3478	0.67
	2972	2975	2953	2956	3010	3012	3119	3120	3193	3193	3369	3370	1.67
24	2719	2720	2648	2650	2713	2716	2892	2895	3032	3031	3253	3252	1.17
	3108	3110	3012	3014	3098	3099	3206	3206	3263	3263	3454	3452	0.50
	2971	2974	2953	2956	3010	3012	3119	3120	3191	3192	3369	3370	1.83
25	2876	2878	2798	2800	2854	2855	3025	3027	3146	3146	3326	3324	0.83
	3094	3095	2998	3000	3084	3085	3193	3194	3252	3252	3445	3443	0.50
	2972	2974	2954	2956	3012	3014	3120	3122	3192	3194	3368	3369	1.83
												Avg.	1.03

Table F.2: Fault detection and DRDY high-low-high time for different system settings.

Test #	UV [V]	OV [V]	OT [°C]	Fault	Detected	DRDY Time [us]	Time / Cell [us]
1	0.70	4.15	55	UV	Yes	572.6	31.81
2	0.90				Yes	572.6	31.81
3	1.10				Yes	572.5	31.81
4	1.30				Yes	572.7	31.82
5	1.50				Yes	572.6	31.81
6	1.70				Yes	572.6	31.81
7	1.90				Yes	572.6	31.81
8	2.10				Yes	572.6	31.81
9	2.30				Yes	572.6	31.81
10	2.50				Yes	572.6	31.81
11	2.70				Yes	572.6	31.81
12	2.90				Yes	572.6	31.81
13	2.50	3.95	55	OV	Yes	572.6	31.81
14		4.15			Yes	572.5	31.81
15		4.35			Yes	572.5	31.81
16		4.55			Yes	572.6	31.81
17		4.75			Yes	572.6	31.81
18		4.95			Yes	572.5	31.81
19	2.50	4.15	40	OT	Yes	572.6	31.81
20			45		Yes	572.6	31.81
21			50		Yes	572.7	31.82
22			55		Yes	572.6	31.81
23			60		Yes	572.6	31.81

24			65		Yes	572.5	31.81
					100.00%	572.5913043	31.81

Table F.3: Cell voltages as a function of time to test cell balancing.

Time [min]	Cell 1 [V]	Cell 2 [V]	Cell 3 [V]	Cell 4 [V]	Cell 5 [V]	Cell 6 [V]
0	3.726	3.729	3.728	3.727	3.726	3.727
5	3.726	3.729	3.728	3.727	3.726	3.727
10	3.726	3.729	3.728	3.727	3.726	3.727
15	3.726	3.729	3.728	3.727	3.726	3.727
20	3.726	3.728	3.728	3.727	3.726	3.727
25	3.726	3.728	3.728	3.727	3.726	3.727
30	3.726	3.728	3.728	3.727	3.726	3.727
35	3.726	3.728	3.728	3.727	3.726	3.727
40	3.726	3.728	3.728	3.727	3.726	3.727
45	3.726	3.728	3.728	3.727	3.726	3.727
50	3.726	3.728	3.728	3.727	3.726	3.727
55	3.726	3.728	3.728	3.727	3.726	3.727
60	3.726	3.727	3.728	3.727	3.726	3.727
65	3.726	3.727	3.728	3.727	3.726	3.727
70	3.726	3.727	3.728	3.727	3.726	3.727
75	3.726	3.727	3.728	3.727	3.726	3.727
80	3.726	3.727	3.728	3.727	3.726	3.727
85	3.726	3.727	3.728	3.727	3.726	3.727
90	3.726	3.727	3.728	3.727	3.726	3.727
95	3.726	3.727	3.728	3.727	3.726	3.727
100	3.726	3.726	3.728	3.727	3.726	3.727
105	3.726	3.726	3.728	3.727	3.726	3.727
110	3.726	3.726	3.727	3.727	3.726	3.727
115	3.726	3.726	3.727	3.727	3.726	3.727
120	3.726	3.726	3.727	3.727	3.726	3.727
125	3.726	3.726	3.727	3.727	3.726	3.727
130	3.726	3.726	3.727	3.727	3.726	3.727

135	3.726	3.726	3.727	3.727	3.726	3.727
140	3.726	3.726	3.727	3.727	3.726	3.727
145	3.726	3.726	3.727	3.727	3.726	3.727
150	3.726	3.726	3.726	3.727	3.726	3.727
155	3.726	3.726	3.726	3.727	3.726	3.727
160	3.726	3.726	3.726	3.727	3.726	3.727
165	3.726	3.726	3.726	3.727	3.726	3.727
170	3.726	3.726	3.726	3.727	3.726	3.727
175	3.726	3.726	3.726	3.726	3.726	3.727
180	3.726	3.726	3.726	3.726	3.726	3.727
185	3.726	3.726	3.726	3.726	3.726	3.727
190	3.726	3.726	3.726	3.726	3.726	3.727
195	3.726	3.726	3.726	3.726	3.726	3.727
200	3.726	3.726	3.726	3.726	3.726	3.726

Appendix References

[A.1] http://www.fsaeonline.com/content/2016_FSAE_Rules.pdf