EE175A/B Final Report
Impedance-based Battery Management System

EE 175AB Final Report
Department of Electrical Engineering, UC Riverside

Project Team Member(s)	Jack Gatfield, Jack Gu, Joseph Gozum
Date Submitted	06/10/2019
Section Professor	Professor Roman Chomko
Revision	2.0
URL of Project Wiki/Webpage	Video Demonstration: https://youtu.be/oQQVyO3CPuU
Permanent Emails of all team members	jackgu70@gmail.com joseph.f.gozum@gmail.com jack.gatfield@comcast.net

Revisions

Version	Description of Version	Author(s)	Date Completed	Approval
0.1	First draft - Not to be released	Jack Gatfield, Joseph Gozum, Jack Gu	03/01/2019	
0.5	Second draft - To be submitted for commenting and notes	Jack Gatfield, Joseph Gozum, Jack Gu	03/07/2019	
0.8	Final draft– edited with commenting and notes from previous version	Jack Gatfield, Joseph Gozum, Jack Gu	03/16/2019	
1.0	Final version – To be submitted for final grading	Jack Gatfield, Joseph Gozum, Jack Gu	03/18/2019	
2.0	Revised version – Revised through ENGR 181W	Jack Gatfield, Joseph Gozum, Jack Gu	06/10/2019	

Table of Contents

1 Executive Summary

Lithium-Ion batteries are becoming increasingly important in the automotive field, specifically electric vehicles. Lithium-Ion batteries come with concerns such as continued health and safety after production. One of the most prevalent problems is cell mismatch, which is batteries within a pack having different voltage levels. If all cells do not have similar voltages, they may become damaged. Current battery management systems (BMS) cannot easily determine a cell mismatch. The impedance-based battery management system (IBMS) that was designed in this project can determine cell mismatch without the time intensive current processes. The IBMS also requires less circuitry to dissipate heat and power compared to current BMS.

Along with decreasing testing time, this project had several other goals. The following are goals of the project:

- Lowering overall equipment size
- Lowering testing time
- Lowering power consumption
- Increasing overall safety
- Increasing accuracy in the impedance calculation
- Increasing cell mismatch detection accuracy

Many of these goals are in response to inefficiencies found in current BMS.

In terms of design objectives, the most important was to maximize the voltage response of battery while maintaining safety. The Lithium-Ion battery could overheat if exposed to large amounts of current. In order to keep the battery at safe temperatures, the IBMS was limited to injecting 20 mA of current. There is also a high wattage resistor used for excess heat dissipation. Some key features of this project are:

- Safe current injection
- Voltage data collection
- Singular and sweep frequency testing
- Impedance calculations

After testing, the IBMS produced results that followed trends similar to those found in the John Hopkins article [3]. Further testing is still needed to fully verify the accuracy of the impedance calculations.

The IBMS of this project was able to calculate the impedance of a lithium-ion battery. It collected the voltage and current of the battery safely and was able to reconstruct its sinusoidal waveform for the impedance calculation. The system was able to calculate impedance at multiple frequencies, either through a set level or a frequency sweep. This allowed for health trends to be graphed and observed.

2 Introduction

2.1 Design Objectives and System Overview

Lithium-ion batteries play a critical role in industry, powering machinery, tools, and electric vehicles. Over time and disuse cell mismatch occurs which is when individual lithium-ion batteries have different voltage levels than neighboring cells. Continued use of mismatched cells leads to over-charging/-discharging which affects the batteries overall health. To counteract this problem, battery management systems (BMS) are designed to monitor batteries to offset these varying voltage levels.

Current BMS only measure a battery's overall voltage and surface temperature. These are poor indicators of the battery's lifespan and overall condition and do not address the inherent mismatch of cells. These established systems can be improved by collecting the battery's impedance at multiple frequencies. An impedance-based BMS (IBMS) can determine battery conditions more accurately and in-depth than the conventional method.

This project is a BMS that uses impedance instead of surface temperature and voltage to indicate the condition of a battery. Impedance at various frequencies follow patterns that reveal internal issues such as cell mismatch and overcharging. Finding hidden issues with this system before an accident occurs increases the efficiency and safety of the batteries.

The project is divided into several subsystems: function generation, current generation, data collection, and data processing. Function generation modifies and generates the frequency of the injected current. The current injection subsystem injects the modified current into the tested battery to induce and measure a voltage response. During data processing, the voltage response is processed with the voltage response across a resistive load. From these two measurements, impedance is calculated for the specific frequency. The process is repeated for different frequencies and then processed for trend analysis and looked at overall. Users can either input specific frequencies or use a general frequency sweep. The system takes lithium-ion batteries as inputs and outputs impedance measurements.

Using concepts from signal processing and circuit design, the project is designed to have the final technical specifications, completed by the following team member:

- Jack Gatfield:
    - Easy to follow circuitry
    - Data processing through MATLAB
    - Safe power dissipation and overall safety
    - Frequency range for overall analysis: $f \in [1\text{ Hz}, 1\text{ kHz}]$
    - Short overall testing time: Testing Time < 45 minutes
    - Accuracy within 10% to experimental research data
    - Semi-low cost: Cost < $150

- Jack Gu:
    - Safe but effective injected current through the battery: $I_c \in [1\text{ mA}, 20\text{ mA}]$
    - Flexibility in testing circuit allowing for modification in future tests
    - Safe power dissipation and overall safety
    - Frequency range for overall analysis: $f \in [1\text{ Hz}, 1\text{ kHz}]$
    - Accuracy within 10% to experimental research data
    - Semi-low cost: Cost < $150

- Joseph Gozum:
    - Cross-platform data movement and processing
    - Accurate data collection to the third decimal place
    - Short overall testing time Testing Time < 45 minutes
    - Accuracy within 10% to experimental research data
    - Semi-low cost: Cost < $150

With the completion of the overall project, this device and future iterations will monitor the conditions of lithium-ion batteries to check for underlying issues such as overcharging and cell mismatch.

2.2 Backgrounds and Prior Art

Devices that use multi-cell Lithium-ion battery packs are designed for matched cells. Matched cells are cells that have the same capacitance, charge and discharge rates, and temperature variations. Matched cells ensure safe performance and efficiency. As battery packs age, little attention is given to verify their matched status. Using a mismatched cell will cause a cascading effect of issues including large heat variation, over-charging and discharging, and most notably diminishing power storage efficiency.

Current battery monitoring systems focus on measuring cell voltage and temperature. These measurements do not monitor cell mismatch, chemical and material health. Commonly available devices monitor temperature and voltage during a long charging/discharging cycle. These test cycles only ensure that cells do not exceed preset voltage and temperature limits. Examples of available BMS are shown in figure 2.2a through 2.2c.

Impedance measurement of a battery is not a new concept. There are experimental BMS that use impedance to monitor the health of a battery. These devices typically only use one frequency. One frequency does not allow for trends or flexible testing.

The following devices are references to commercially available BMS:

- At the research lab of Professor Cengiz Ozkan of the University of California, Riverside, a Bio-Logic Science Instruments Battery Cycler, shown in figure 2.2a, is used to determine cell mismatch.

Figure 2.2a: Professor Ozkan's research device

- The University of California, Riverside: College of Engineering – Center for Environmental Research and Technology (CE-CERT) uses the NHR Battery Pack Test System – 9200 series, shown in figure 2.2b. The large size is a safety requirement to allow for proper energy dissipation.

Figure 2.2b: CE-CERT monitoring device (NHR Battery Pack Test System – 9200 series)

- Wireless battery monitor, shown in figure 2.2c, are available for hobby battery testing systems. They are less precise as the other examples but they provide similar information about the tested batteries with a shorter test cycle.

Figure 2.2c: Wireless battery monitoring systems such as these are able to be purchased and used by almost anyone, but still fails to address the aforementioned issues

The advantages of this projects design are time, size, and power consumption compared to the available products. This project does not require the battery to go through a full-cycle of charging and discharging which reduces power usage, time and circuitry size.

2.3 Development Environment and Tools

The following software and hardware tools are used:

- Hardware
  - Oscilloscope (Techtronics TDS340**)**
  - Multimeter (FLUKE45)
  - Function Generator (HP 33120A)
  - Power Supply (HPE3630A)
  - Raspberry Pi
  - Arduino UNO
  - Arduino Mega 2560
  - Atmega1284P
  - Breadboards
  - Through Boards
  - Solder station, solder suction pens, solder tip cleaner
  - Circuit parts
    - LF353N OP-AMP
    - Q2N2222 BJT
    - Resistors
    - Capacitors
    - 5 V, 10 V voltage regulators
    - MAX1044 Voltage Converter
    - Barrel Jack
    - 18650 Battery holders
    - XR2206 function generator
    - AD5241 digital potentiometer
    - FTDI TTL-232 Serial Communication Cable
- Software
  - MATLAB
  - Atmel Studio 7 Integrated Development Platform
  - Arduino IDE
  - Python 3.7.2
  - Eagle PCB Design Software
  - Visual Studios 2013
  - Cloud9 Development Environment
  - RealTerm: Serial/TCP
  - PuTTY Terminal Emulator

2.4 Related Documents and Supporting Materials

[1] Atmel Corporation, "8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash", ATMEGA1284P Datasheet, Nov. 2009

[2] "Getting Started with Atmel Studio 7." Microchip Technology, Chandler, AZ, Jan-2018.

[3] J. Alveredo, "CERT C Programming Language Secure Coding Standard." Carnegie Mellon University, Pittsburgh, Pennsylvania, 10-Sep-2007.

[4] "UM10204 I2C-bus specification and user manual." NXP Semiconductors, Eindhoven, Netherlands, Apr-2014.

[5] "USART and Asynchronous Communication." Oregon State University, Corvallis, Oregon, Jul-2010.

2.5 Definitions and Acronyms

- BMS: Battery Monitoring System.
- IBMS: Impedance Based battery Monitoring System
- CE-CERT: College of Engineering - Center for Research & Technology
- OP-AMP: Operational amplifier.
- Digi-Pot: Digital potentiometer, an adjustable resistor set digitally.
- ADC: Analog-to-Digital Converter
- USART: Universal Synchronous Asynchronous Receiver Transmitter
- I2C: Inter-Integrated Circuit
- RS-232: Recommended Standard 232
- CSV: Comma-Separated Value
- BJT: Bipolar Junction Transistor.
- SAR: Successive-Approximation Register
- PCB: Printed Circuit Board
- SMT: Surface Mount Technology
- THP: Through Hole Part
- TBC: Through Board Circuit
- TXD: Transmit Data
- RXD: Receive Data
- DMA: Direct Memory Access
- IC: Integrated Circuit
- DC: Direct Current
- AC: Alternating Current
- FTDI: Future Technology Devices International
- TTL: Transistor Transistor Logic
- GUI: Graphical User Interface
- CPU: Central Processing Unit

3 Design Considerations

The designs constraints of this project are size, signal-type, current limitation, high-power rated load, accuracy, precision, data transfer speeds, and safe power dissipation.

- Size is considered to address the issue of currently available products which are large and not portable.
- Signal-type is limited to sinusoidal and square waveforms because both provide data with minimal computation towards impedance.
- Current limitation is related to the current injected to the battery. If too much current is injected to the battery it will heat up the battery and affect the internal chemicals.
- High-power rating loads are needed to dissipate produced heat.
- Accuracy is needed in all components that do signal readings. The accuracy of the differential probes and the ADC on the microcontroller are the biggest concerns. The data this project collects is in the millivolt range of voltage and any inaccuracy affects the results.
- Precision is considered in the same components as accuracy. Any loss of precision will show up as errors in the results.
- Data transfer speeds affect the overall testing time that this project is trying to minimize. The transfer speeds between the microcontroller and computer are set at 9600 bits per second. This speed allows for fast data transfer without loss in data.
- Safe power dissipation is related to high-power rating load. In order to safely dissipate power, the load must be able to handle it.

In order to design this system, the following points are taken into considerations:

- Small form factor function generator to produce sinusoidal and square waveform voltages
- Independent biasing circuit (DC Offset) for sinusoidal waveform
- Voltage divider for square waveform
- Limits on the current through the battery
- Low resistance but high-power rating load
- Reading the small voltage response across the battery after current injection
- Required to sweep through a large range from 1 Hz to 1 KHz
- Transfer the data collected from the microcontroller to the computer
- Safe power dissipation in the test circuit
- Maintain an input voltage for the current injection circuit
- Accuracy in reading of differential probes
- Proper and clean circuit implementation

3.1 Assumptions

The system this manual is based on is a prototype. The following are assumed:

- The batteries used are 18650 Lithium-Ion batteries.
- The data provided by the John Hopkins experiment give an accurate reflection on the condition of the battery.
- The BMS is given time to complete the full frequency sweep.
- The batteries are assumed to remain safely connected to the system until testing is completed.
- The testing is assumed to be conducted at room temperature.

3.2 Realistic Constraints

3.2.1 Voltage Supply

Safely supplying voltage is a circuit design and implementation constraint. Voltage needed to be supplied in environments with a simple wall outlet. This required the implementation of a barrel jack and DC power adapter to provide voltage to the system.

3.2.2 Microcontroller Processor and Memory Speed

Microcontrollers operate on a set processor speed. This speed had to be taken into account for developing the data acquisition code to ensure proper functionality. This speed also had to be taken into account when deriving the data sampling frequency of the ADC.

3.2.3 Battery Safety

Batteries are very volatile and unsafe if improperly handled. A safe testing environment required that less than 20 mA could be injected into the battery. The load resistor must dissipate the power generated.

3.2.4 Ability to Capture Induced Battery Voltage Response

Batteries are volatile and unsafe. The circuit is designed to ensure safety when testing batteries.

3.3 System Environment and External Features

3.3.1 Power

The Printed Circuit Board (PCB), Through Board Circuit (TBC), and the breadboard version of the circuit require external power. The PCB and the TBC are powered through a 12 V DC power adapter. The bread board circuit is powered through desktop power supplies.

3.3.2 User Interface

The user interface uses the PuTTY Terminal, Python Command Window, and MATLAB. The PuTTY Terminal establishes communication between the Microcontroller and the user's computer. Python formats the gathered data. MATLAB process the data generated and presents the results to the user.

3.3.3 Computer to Microcontroller interfacing

PuTTY and RealTerm are used as terminal interfaces for communication between the user computer and the microprocessor.

3.4 Industry Standards

[1] J. Alveredo, "CERT C Programming Language Secure Coding Standard." Carnegie Mellon University, Pittsburgh, Pennsylvania, 10-Sep-2007.

[2] "RS232 Interface Module." Solartron metrology, Bognor Regis, England, Jun-2010.

[3] "UM10204 I2C-bus specification and user manual." NXP Semiconductors, Eindhoven, Netherlands, Apr-2014.

[4] ISO/IEC 1475: UART. Rev. 2.40-28 April 2015

3.5 Knowledge and Skills

Joseph Gozum:

- Previously learned:
  - EE/CS120B - Intermediate Embedded Systems
    - Embedded system programming and peripheral interfacing
  - EE128 - Data Acquisition, Instrumentation, and Process Control
    - Embedded system programming, data acquisition, and communication
  - EE01A/B
    - Circuit design and circuit simulation
  - EE100A/B - Electronic Circuits
    - Circuit design
  - PuTTY Terminal Emulator
    - An interface to receive and interact with connected devices using different communication methods with a terminal-like design
- Learned:
  - Serial communication methods and standards
    - Learned how to properly implement the different methods
  - Digital logic for controlling peripherals
  - Python for data manipulation and automation
    - Learned about the ability of Python and supporting libraries

Jack Gu:

- Previously Learned:
  - EE100A/B: Electronic Circuits
    - Circuit design for the hardware components of the project
    - Understanding of tradeoffs in performance for different designs
  - EE123: Power Electronics
    - Power flow and consumption in the hardware components of the project
    - Use of LTSPICE for various circuit simulations
  - MATLAB (EE110A/B, EE105, EE141, and some self-taught)
    - Design algorithms in developing the phase calculating algorithm of our project
    - Debug MATLAB programs that are the main processing unit of our project
  - EE001A/B
    - Circuit design, implementation, and testing

- Learned:
    - PCB Soldering
        - How to safely solder components onto a test board and PCB board
    - LabVIEW
        - How to run LabVIEW with MATLAB for implementation of GUI in the future
        - Handle user interface and user inputs

Jack Gatfield:

- Previously learned:
    - EE/CS120B: Intermediate Embedded Systems
        - Embedded system program, setup, and interfacing
    - EE128: Data Acquisition, Instrumentation, and Process Control
        - Embedded system programing, integration and data acquisition, and code optimization
    - EE100A/B: Electronic Circuits
        - Circuit design, BJT operation, proper lab and testing procedure of BJT circuits
    - EE123: Power Electronics
        - Circuit design, power flow and consumption, using LTspice for circuit simulation
- Learned:
    - Python programming
        - Needed to developed code to handle data formatting from the serial communication to the MATLAB data read; eventually taken over by Joseph Gozum due to his larger knowledge base of Python
    - MATLAB (EE20, EE110A/B, EE105, EE141, and self-taught)
        - Data processing
    - PCB design and manufacturing
        - Learned how to use Eagle PCB Design Software to implement a PCB of the circuit. This required learning how to use the program, learning about best practices for PCB design, developing the full circuit to implement, and learning how to properly send it to printing.
    - AutoCAD 2019 Design
        - Learned how to develop a 3D model to be used as a case for the future PCB. This required learning the program and then properly implementing the box to the needed dimensions.
    - PuTTY Terminal Emulator
        - Learned a basic understanding of how the putty terminal works in order to obtain data from the circuit. This task was left mostly to Joseph Gozum due to his prior knowledge of the program.

3.6 Budget and Cost Analysis

The cost breakdown in Table 1.6.1 shows the cost of the final system without the PCB. With the parts below, and the needed PCB, one can fully implement the system.

Parts in final circuit design		
Part	**QTY.**	**Cost per unit**
12V DC Power Adapter	1	$8.95
Barrel Jack	1	$1.50
0.33uF non polarized capacitor	2	$1.50
0.1uF non polarized capacitor	4	$1.30
5V voltage regulator [L7805C]	1	$0.75
10V voltage regulator [BA17810]	1	$0.84
Voltage inverter [Max1044CPA+]	1	$3.40
Potentiometer (Digital [AD5242])	1	Digital: $2.68
Function generating chip [XR2206]	1	$7.95
2 level dip switches	2	$1.20
NPN BJT	1	$3.00
18650 battery holders [2n2222a]	1	$1.95
18650 battery	Test Batch	$5.00
Op amp [LF353]	1	$1.00
Microcontroller [Atmega1284p]	1	$5.00
Programmer and cable	1	$75.00
Program header [IEEE UCR programmer header]	1	$1.00
Serial communication cable [FTDI Serial TTL-232 USB Cable]	1	$17.95
Resistors (1Ω to 20kΩ)	25	$1.20 to $3.00

Table 1.6.1

The final cost of the project is low when compared to current available BMS. The main cost of this design is the PCB. PCB printing and implementation can be minimized if the system is put into mass production, or a cheaper printing establishment is found.

3.7 Safety

This project has several safety hazards in the final prototype. The first concern is the danger from directly injecting current into a lithium-ion battery. If the amount of current entering a battery exceeds a certain threshold (20 mA), the battery could react negatively and potentially melt or even explode. Current entering the battery is controlled so that the current magnitude is large enough to induce a voltage response while low enough to not damage the battery. Transient responses (Unwanted spikes in current) is also addressed through switches and lowering the frequency sweep step size.

The second concern is the issue of heat buildup when the system is running. Lithium-ion batteries react negatively to constant exposure of heat. This constant heat generation stems from the inherent long testing time of our system as well as the need for safe power dissipation after the current passes through the battery. Methods implemented to address heat issue include heat sinks, high power rated components, and a power regulating network. The PCB implementation reduces the possibility of wire meltdown as well as allowing more airflow into the circuit, reducing heat buildup. The power regulating network allows for the system to run for long periods of time without extensive heat buildup from the power sources. The high-power rating components reduce the risk of component meltdowns as well as allow energy to be dissipated efficiency.

The final concern is the issue of unwanted transient responses. Transient responses in the system are defined as sudden spikes in voltage or current. Transient responses appear in the system when the battery is either connected or removed from the system and when the switch in frequency values is too large (> 50 Hz). Stopping transient responses was done in two ways: Reduction of frequency step sizes and removal of the battery from the circuit when switching frequencies.

3.8 Performance, Security, Quality, Reliability, Aesthetics etc.

This project requires precision and accurate reading of voltage signals in the millivolt range. The computation for impedance is used in this project is based off these readings. The components of main concern for performance are the differential probes and the ADC.

Next is security concerns. This product does not have any security and it does not require it in order to function. This project does have quality standards that it must meet.

The device must be able to product quality results that measure up to widely available products. This is related to its ability to gather precise and accurate data which is then used determine cell health and mismatch. The IBMS also has reliability standards.

This IBMS must be able to support or replace existing products. Therefore, its reliability must be greater than or equal to those same products. Another important concern for this project is its size.

The IBMS must have a small size and silent aesthetic to address the opposite seen in widely available products.

Finally, there is an importance on control for this project. The project involves working with analog signals but the actual reading is digital. Components that work with the analog signal can also be digitized to reduce error in using analog components. Analog potentiometers and switches that would be used can be replaced with digital potentiometers and switches.

3.9 Documentation

To maintain and generate documentation for the senior design work, a Google Drive folder was made that contains all technical specifications and instructions. During meetings and experiments, all events were logged. The file would be titled and placed with an appropriate name and in the proper sub-folder. For handwritten notes dedicated notebooks were used and later typed or scanned. Hand written notes were few so most of the documentation was typed and stored directly through documents on the Google Drive.

3.10 Risks and Volatile Areas

This project has a battery that is being actively injected with current. This process can be very dangerous if handled incorrectly. To help mitigate the safety risk for this system, precautions were taken with current levels and power dissipation. The current was kept below 20 mA and the load resistor has a high-power rating. In future iterations a way to disconnect the battery automatically would be a further step for system safety.

Another volatile area in the design is the BJT that converts the voltage signal to a current signal. If the frequency of the voltage signal is changed rapidly in magnitude the BJT will be damaged and cause a current surge. To mitigate this, a slow manual frequency stepping progression was used for data collection. In the future an automatic frequency sweep using a digital potentiometer controlled by the microcontroller will be used.

4 Experiment Design and Feasibility Study

4.1 Experiment Design

4.1.1 Current shaping viability experiment

Objective: Test whether it is possible to send a sinusoidal current through the 2N2222 transistor and check if the sinusoidal waveform is --intact when used with a test battery.

Setup: The following setup was used to test the shape of the current under various frequencies.

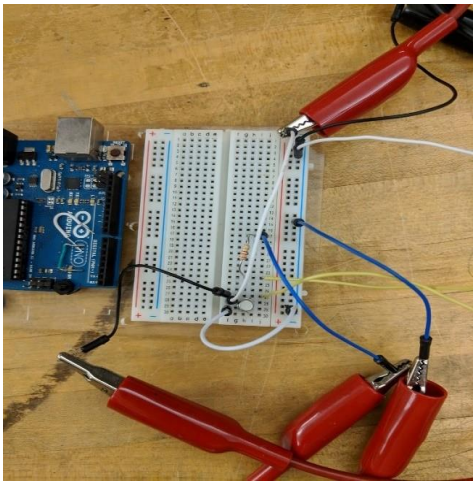

Figure 4.1.1a: Physical setup of the test circuit for the experiment

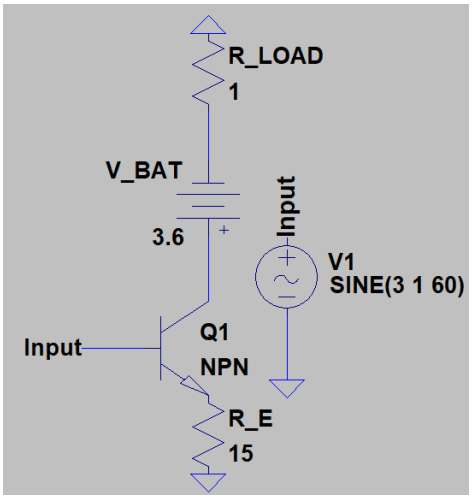

Figure 4.1.1b: Test schematic for the experiment in LTspice

Procedure:

1. Connect the function generator to the test circuit (Black wire in Figure 4.1.1a)
2. Connect the common ground to the circuit (Refer to Figure 4.1.1b).
3. Connect the negative terminal of the test battery and indicate the section where you would connect the positive terminal.
4. In those two terminals listed in step 3, connect the oscilloscope probes across the same terminals.
5. Turn on the function generator and slowly increment the frequency while retaining the voltage input with the input: $2 + \sin(\omega t)\, V$
6. Record data on the oscilloscope
7. Repeat for different frequencies until satisfied.

Expected Result: The signal across the battery is expected to be noisy but still maintain a sinusoidal shape.

Result: The following was obtained:

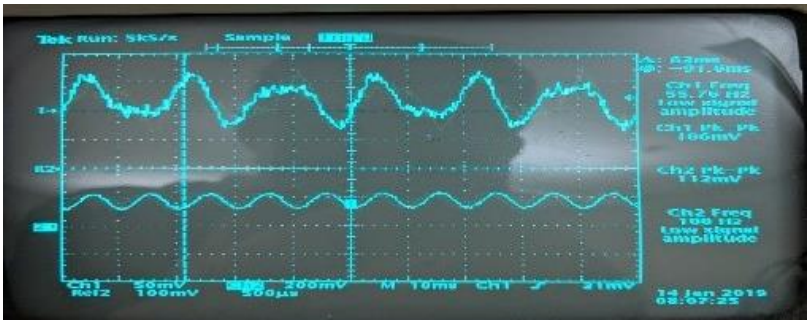

Figure 4.1.1c: Current flowing the battery (top) and the input voltage (bottom) at 100 Hz.

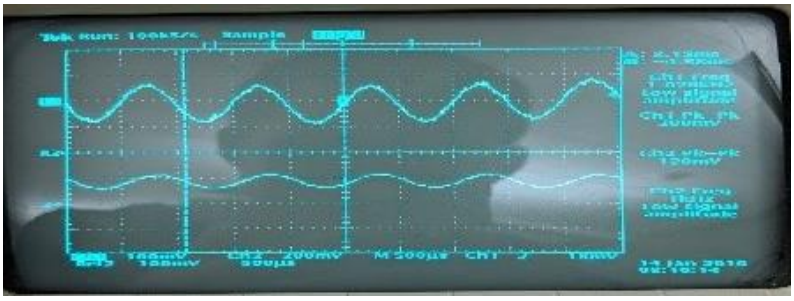

Figure 4.1.1d: Current flowing the battery (top) and the input voltage (bottom) at 1000 Hz

Looking at the figure 4.1.1c and 4.1.1d, the signal retains a sinusoidal waveform in the battery regardless of frequency.

4.1.2 Floating Node experiment

Objective: Examine the voltage of the input and output nodes for the current injection circuits BJT and check for floating points.

Setup: The following setup was used to check for floating nodes in our current injection module:

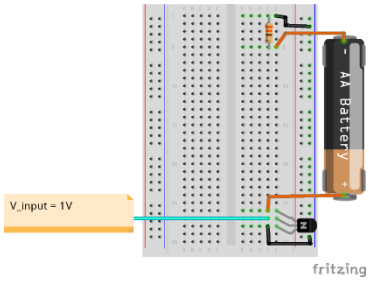

Figure 4.1.2a: Physical setup of our floating node experiment

Procedure:

1. Connect appropriate ground terminals
2. Connect differential voltage probes (Figure 4.1.2a blue wire) at the collector of the transistor and ground
3. Connect a power supply (Figure 4.1.2a black wire) which should be a DC input voltage. [0 V, 5 V]
4. Connect the battery in series in between the resistive load and BJT (Figure 4.1.2a yellow wire)
5. Record the voltage at the collector for all DC values in the sweep.

Expected Result: Simulations shown below suggest the possibility of a floating node voltage.

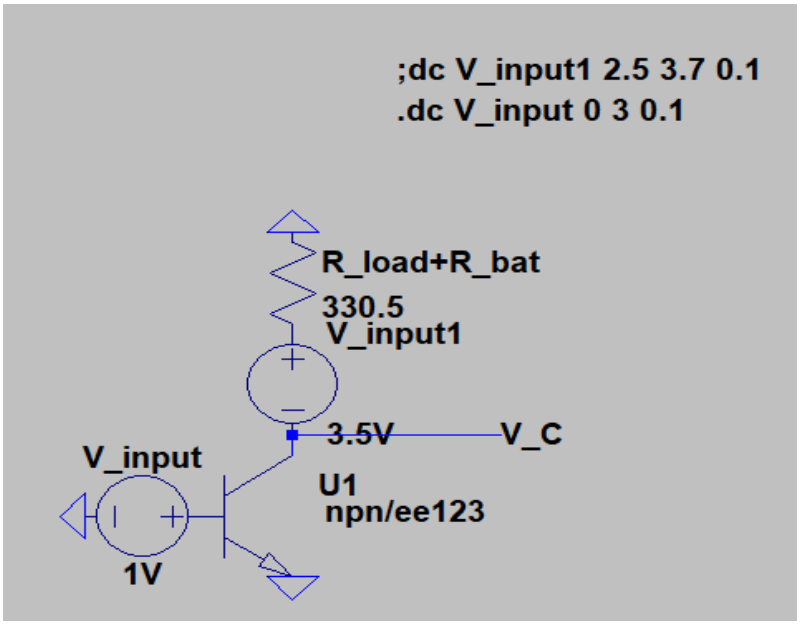

Figure 4.1.2b: LTspice schematic for simulations of a floating node

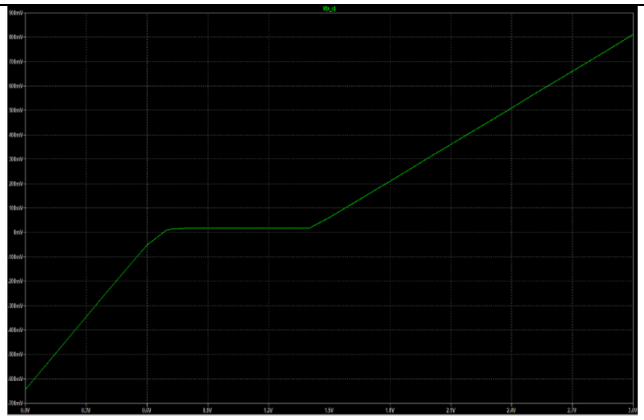

Figure 4.1.2c: Output for the collector voltage and current

A floating node voltage is expected at the collector component of the transistor.

Result: The graph below displays all induced responses from input voltages ranging from 0 to 3.5 V:

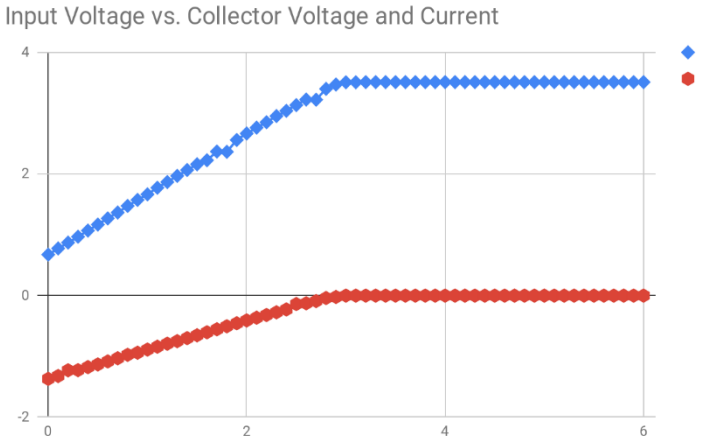

Figure 4.1.2d: Output at the collector for both voltage and currents

There appears to be no floating voltage nodes when looking at Figure 4.1.2d.

4.1.3 Amplifier accuracy test

Objective: Test the accuracy and precision of the differential amplifier module when applying various DC signals as inputs.

Setup: The following setup was implemented to test the functionality and accuracy of the amplifier:

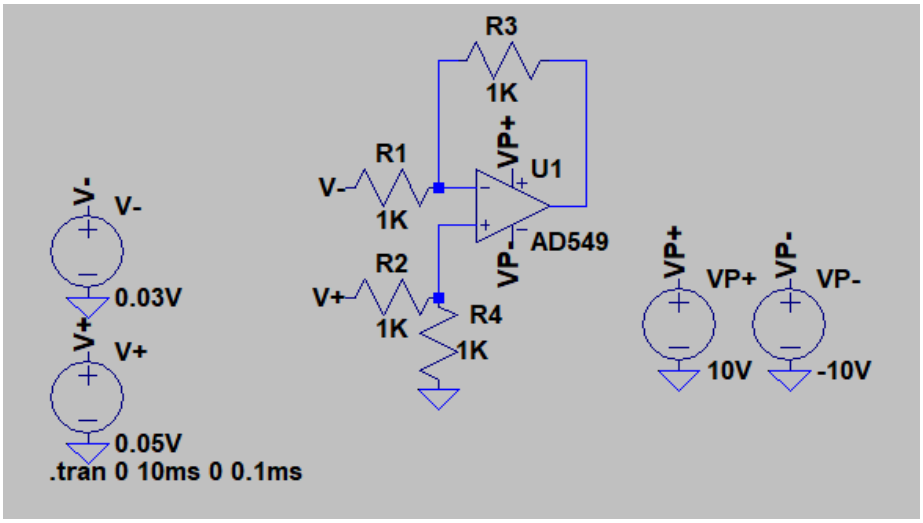

Figure 4.1.3a: LTspice schematic for simulation of the operational amplifier

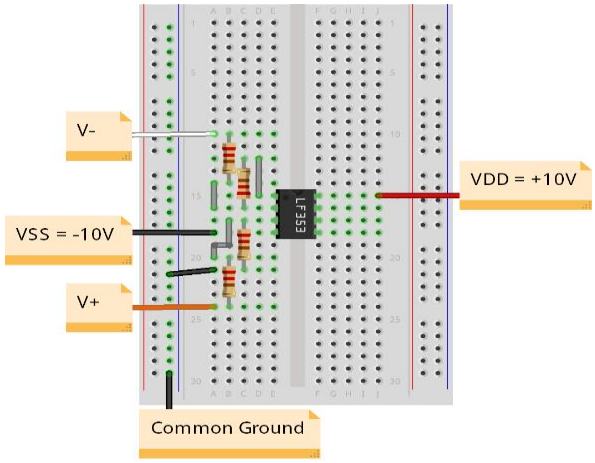

Figure 4.1.3b: Physical implementation diagram of the differential amplifier

Procedure:

1. Construct the circuit and connect all ground terminals.
2. Apply the appropriate power supplies and input voltages to the circuit.
3. Record the results of the output and test various DC values to be differentiated.

Expected Result: Simulations suggested accurate calculations of the amplifier. Slight errors were still expected.

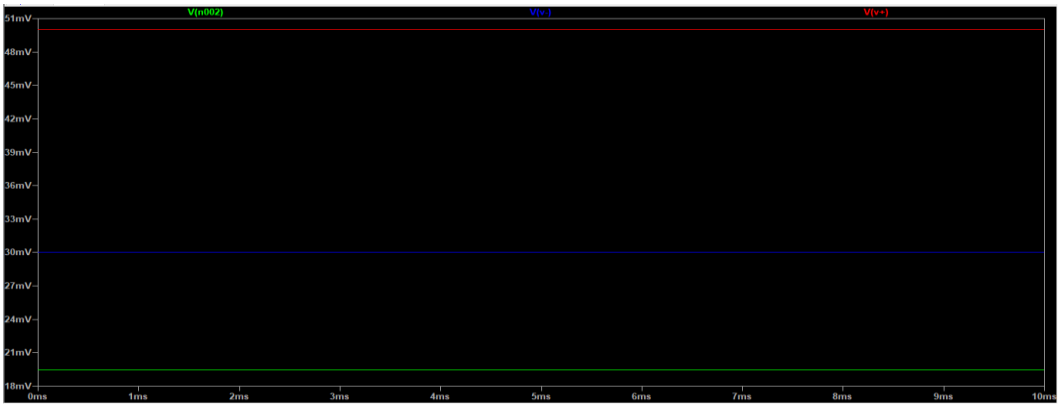

Figure 4.1.3c: Initial Test for $V_2 = 0.05\,V;\ V_1 = 0.03\,V;\ R_1 = R_2 = R_3 = R_4 = 1\,K\Omega;$

When looking at the LTspice simulations, an output was expected that follows the gain equation:

$$A = \frac{R_3}{R_1}(V_+ - V\_); R_3 = R_4; R_1 = R_2$$

Result: The output is close to what was expected in the gain calculation due to the high precision of our resistors (±2%). For example:

$$Output = (V_+ - V\_) = 1.017\,V;$$

$$V\_ = 1\,V; V_+ = 2\,V. NOTE: This\ was\ tested\ with\ seperate\ power\ supplies$$

4.1.4 Biasing Circuit Output Test

Objective: Test whether the biasing circuit successfully biases the inherent function generator chip correctly. Correct biasing allows for proper function of the current injection module.

Setup:

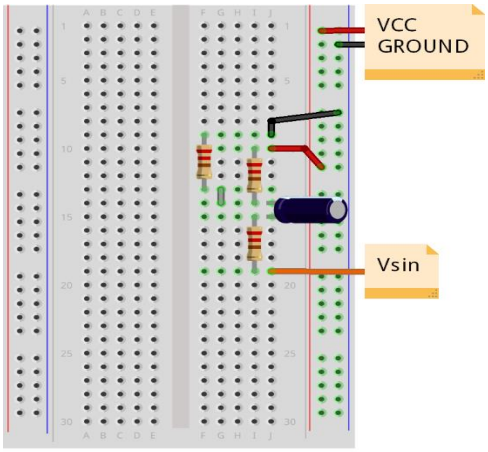

Figure 4.1.4a: Physical implementation of the biasing circuit to be tested

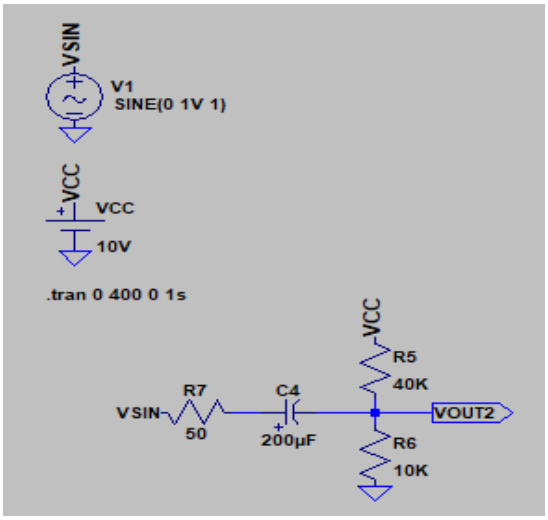

Figure 4.1.4b: LTspice Schematic for the biasing circuit to be tested

Procedure:

1. Setup the biasing circuit shown in Figure 4.1.4a without connecting the power supply and the input.
2. Connect the power supply and then the input voltage.
3. Record the output.
4. Change capacitances (C4 in Figure 4.1.4b) and repeat step 3, taking note of the shape and voltage swing.

Expected Result: In order to predict the results of the experimental system, simulations were performed and yielded the following results:

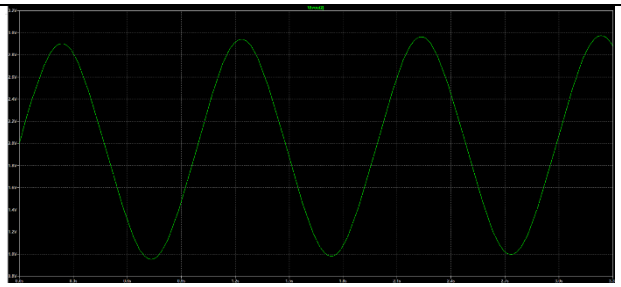

Figure 4.1.4c: LTspice simulation output for a 1 Hz biased signal.

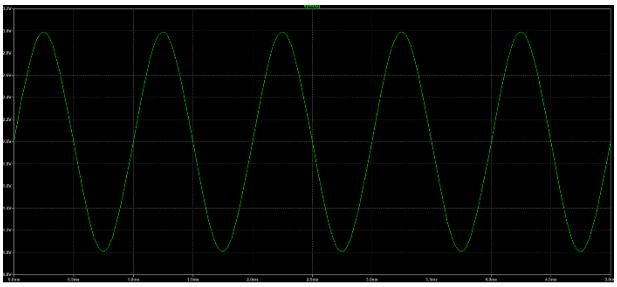

Figure 4.1.4d: LTspice simulation output for a 1 kHz biased signal.

The results from both LTspice simulations (Figure 4.1.4c and Figure 4.1.4d) suggest that biasing was successful across all frequencies. It should be noted that voltage swing decreases proportionally with the capacitance.

Result: Physical testing confirmed expected results. Because of tradeoff when looking at voltage and capacitance swing, the 100 µF capacitor served as the optimal capacitance value for the circuit. The capacitor results in a 10% drop in swing (1.8 V_{PP}) while maintaining the sinusoidal shape needed for all frequency ranges.

4.1.5 Frequency Sweep Test

Objective: Test whether the function generator chip can generate sinusoids within frequency range 10 Hz to 1 kHz.

Setup: The figures below show the schematic and physical implementation of the test circuit respectively.

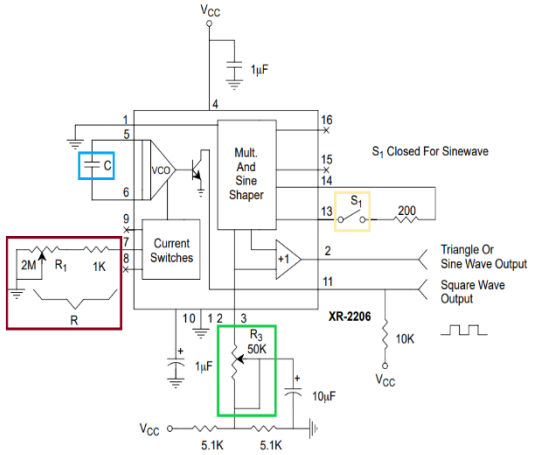

Figure. 4.1.5a: Schematic of our setup of the function generator chip

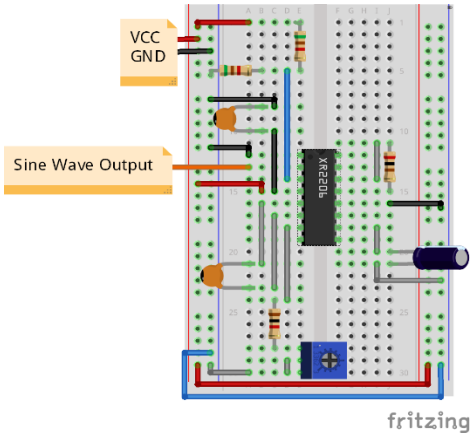

Figure. 4.1.5b: Physical implementation of our test setup

Procedures:

1. Assemble the circuit shown in Figure 4.1.5a
2. Replace the digital potentiometer (Red box in Figure 4.1.5a) with an analog potentiometer. This is done to simplify testing for manual frequency adjustment.
3. Adjust the potentiometer to the lowest resistance.
4. Record the output of the sine wave output pin.
5. Adjust potentiometer so that the next frequency is an increment of experiment choice.
6. Record the output of the sine wave output pin.
7. Repeat steps 5-6 as necessary, until all experiment frequencies are tested.

Expected Result: The function chip should produce the desired frequency range. Referencing the user manual for the chip states that the chip can handle the range for this experiment. The range of interest for this test is between 10 Hz to 1 MHz.

Result: The results reflect expectations, sinusoids within the desired frequency range were produced. However, to produce the whole frequency range one capacitor value needed to be changed.

4.1.6 Initial Data Reading Experiment

Objective: Replicate the experiment in section 4.1.9 but replace the Arduino Uno with the Atmega1284P that uses new code developed for it.

Setup: The following is the test circuit for Initial Data Reading Experiment

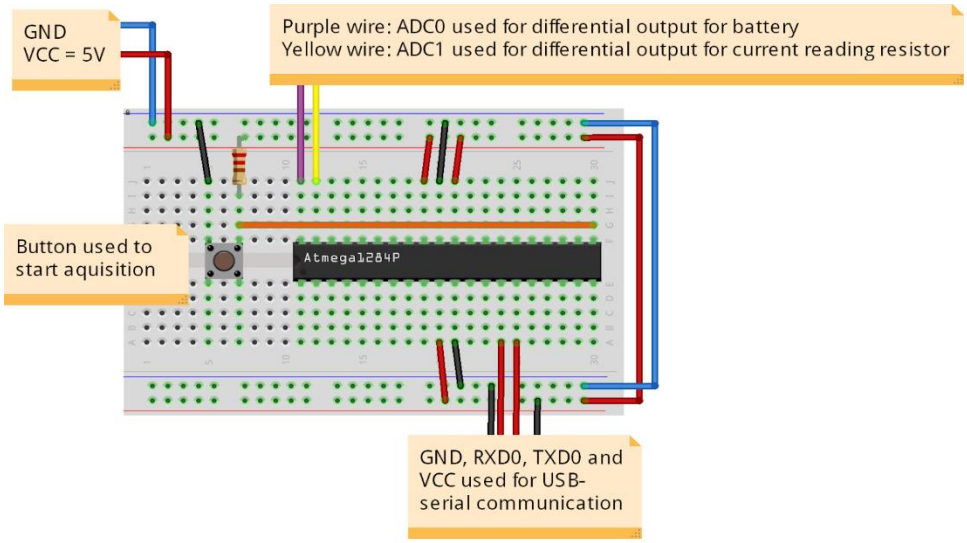

Figure 4.1.6a: Physical implementation of test circuit

Procedures:

1. Assemble Circuit above
2. Set frequency of function generator to desired value
3. Set parameters for serial communication (e.g., speed, polarity, data, etc)
4. Press button to start data acquisition
5. Save terminal output to text file
6. Run formatting Python script
7. Manually analyze the data

Expected Result: Similar results to the results from section 4.1.9

Result: The voltage response of the battery was measured after current was injected; the values of the test can be found below in Figure 4.1.6b.

Voltage Reading
3.78299120234604
3.67546432062561
3.63147605083089
3.6119257086999
3.6119257086999
3.6119257086999
3.6119257086999
3.6119257086999
3.6119257086999

Figure 4.1.6b: Voltage equivalent value of captured ADC values at 13 Hz

The results are expected, the value hovers around the nominal voltage of the test battery. Changes in the voltage response across the battery changes with the battery.

4.1.7 MATLAB Python Cross-platform Experiment

Objective: Have a MATLAB script call a python script that will format the collected data into a numerical format.

Setup: Raw data from the terminal in text files in the set file location with the Python script and MATLAB in the same directory.

Procedure: Code and properly set up the windows environment to be able to have MATLAB call the python script.

Expected Result: Have the MATLAB script run the data formatting script in order to reduce time in data processing.

Result: The python script was not able to run in the Windows environment. MATLAB did indeed call the script properly but the python script would fail. Debugging was attempted for many hours. It was settled that the data had to be formatted by the Python Script in a Linux environment and then transferred to MATLAB.

4.1.8 Current Maximization Experiment

Objective: Maximize the current flowing through the test battery while maintaining safe power dissipation and battery safety.

Setup: The following is a schematic of the test circuit and a physical implementation.

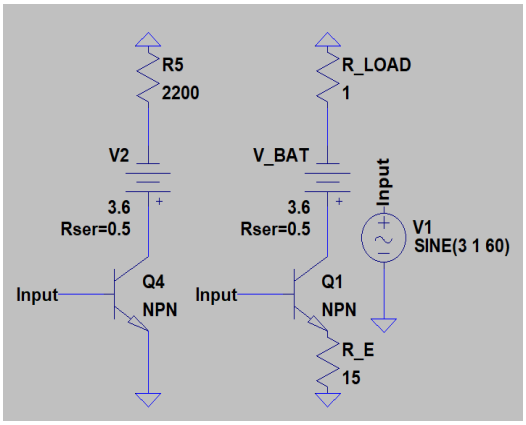

Figure 4.1.8a: Schematic setup of two current injection circuits used for testing and comparisons of input currents

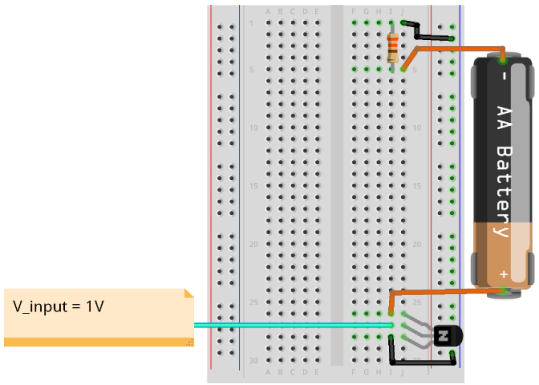

Figure 4.1.8b: Physical implementation of one current injection circuit. Limited materials prohibited simultaneously building and testing two current injection circuits.

Procedures:

1. Assemble the original current injection circuit (Left schematic of Figure 4.1.8a).
2. Prepare the battery for testing (Refer to Figure 4.1.8b's yellow wires).
3. Using the function generator, send a voltage with an arbitrary frequency through the BJT (Figure 4.1.8b's black wire). For the magnitude of the voltage, send in a voltage with an offset of
4. 2 V and an independent swing of 1 V.
5. Record the current flowing through the current injection circuit (Figure 4.1.8b's blue wires)
6. Safely disconnect the circuit and assemble the new experimental current injection circuit (Figure 4.1.8a's right schematic). It is highly recommended to keep the same wiring format so that the same color coding can be used to maintain clarity and safety.
7. Repeat steps 1-4 for the new current injection circuit. If the current recorded is still not satisfactory for user's needs, repeat with new schematic setups.

Expected Result: Several systems were simulated to test for maximum current values.

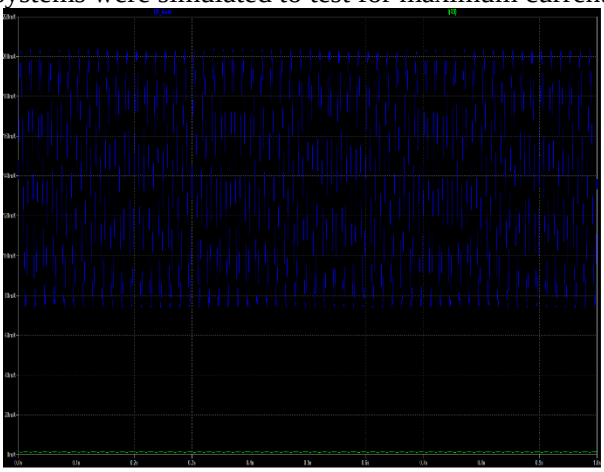

Figure 4.1.8c: Output of the original current injection circuit (green) and the experimental setup (green).

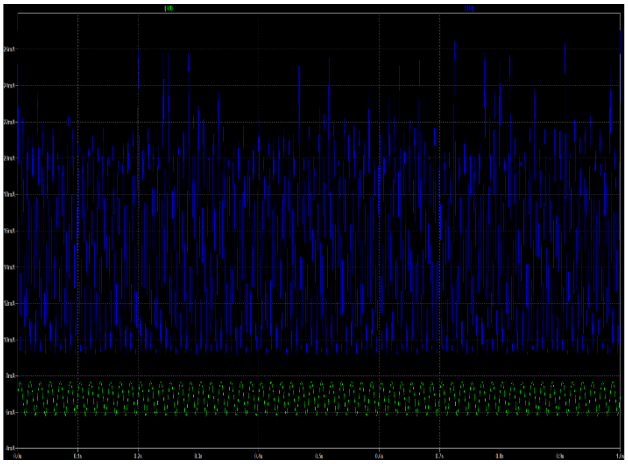

Figure 4.1.8d: Output of the other experimental setups that never made it past simulations. The Darlington configuration (Blue) and the Wilder current mirror output (Green).

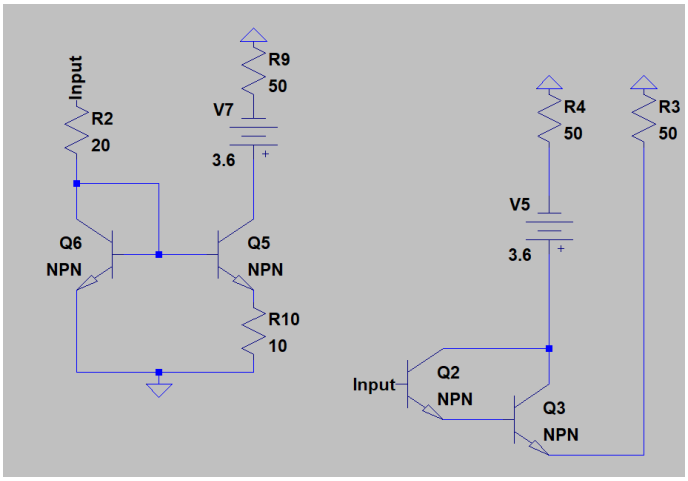

Figure 4.1.8e (Bottom): Wilder current mirror schematic used for simulation (Left) and the Darlington current amplifier schematic used for simulation (Right)

The expected results are current in the 10 to 15 mA range and a sinusoidal output waveform. Two different current sources were simulated to see if they met the specifications set for this project. The schematics can be seen in Figure 4.1.8e and the outputs can be seen in Figure 4.1.8d. The Darlington and Wilder current sources were not physically tested because they did not meet required specifications. The Darlington circuit cannot maintain a sinusoidal output waveform and the Wilder did not produce the current magnitude needed. However, the circuit schematic shown in Figure 4.1.8a produced the output of Figure 4.1.8c and called for physical tests. While the output current from circuit in Figure 4.1.8a in simulations has incredibly high current magnitude, in the physical implementation the current magnitude is lower.

Result: Testing of schematic in Figure 4.1.8a produced current in the desired magnitude range. Initially, the current output of the current injection circuit was in the µA range [100 µA, 600 µA], this magnitude cannot induce a voltage response from the battery. The low current was caused by the circuit setup and current readings being done incorrectly. After fixing the issues, current varied in the mA range [9 mA, 15 mA].

4.1.9 Arduino ADC Testing

Objective: Test the output of the prototype to see if the responses induced produced valid test data that reflect accurately the condition of the battery.

Setup: Figure 4.1.9a is a diagram of the Arduino UNO setup.

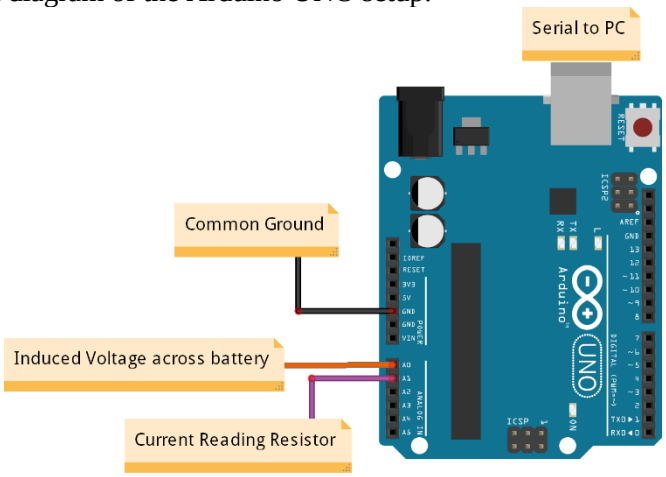

Figure 4.1.9a: Schematic of our testing setup

Procedures:

1. Setup test circuit as shown in Figure 4.1.9b.
2. Connect the corresponding serial communication cable (also powers Arduino Uno)
3. Adjust the frequency and note the value of said frequency.
4. Connect the testing battery through to the current injection circuit module.
5. Record the average voltage across the battery
6. Record the average current through the battery branch
7. Repeat steps 3-6 as desired.

Expected Result: We expected the outputs to be minimal in difference but still sinusoidal in nature. This is because the impedance of the test battery is naturally low, so as a result of that, we expect the readings being processed to be slight deviations from the nominal battery voltage. And after running it through a excel macro that calculates the magnitude of the battery's impedance, we expect it to be incredibly minimal.

Result: The following is the impedance calculated at specified steps in the frequency range of the project. Figure 4.1.9b is the impedance versus frequency plot while Figure 4.1.9b is a table of the impedance at the corresponding frequency.

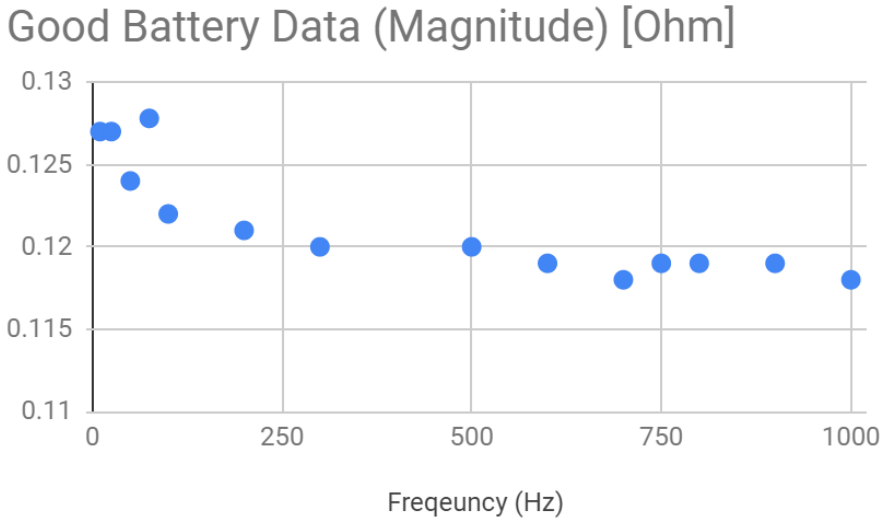

Figure 4.1.9b: Output of the test battery impedance magnitude over the entire frequency range.

Impedance Calculation of a "Good" Battery	
Frequency (Hz)	Impedance
10	0.127
25	0.127
50	0.124
75	0.1278
100	0.122
200	0.121
300	0.120
500	0.120
600	0.119
700	0.118
750	0.119
800	0.119
900	0.119
1000	0.118

Figure 4.1.9c: Hand calculations of impedance placed in a spreadsheet.

The results are inconclusive. While the ADC data acquisition works perfectly, there are still flaws in the results. While the trend of the magnitude (Figure 4.1.9a) follows the trends of the experimental data that this project is based; frequency and impedance being inversely correlated with each other, the impedance magnitude is significantly different.

4.1.10 Arduino to MATLAB Data Acquisition and Processing Experiment

Objective: Test to make sure what we did by hand in Arduino ADC testing can be automated and produce equivalent if not better results as well as define our testing procedures.

Setup: The following is the test circuit using the Arduino Uno in place of the Atmega1284P to make sure the steps from data acquisition to processing is able to produce expected impedance values.

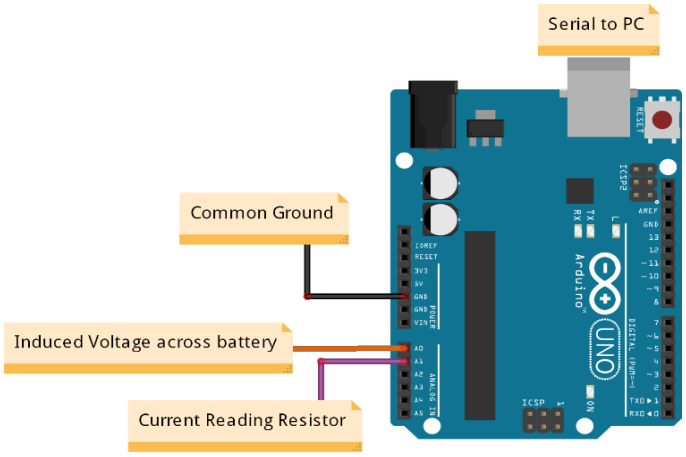

Figure 4.1.10a: Arduino setup used for testing

Procedure:

1. Connect differential amplifier output for battery to A0 as shown in figure 4.1.10a.
2. Connect differential amplifier output for current reading resistor to A1 as shown in figure 4.1.10a.
3. Set desired frequency for testing
4. Connect serial/power cable
5. Open communication port using PuTTY
6. Save the output to a text file
7. Convert from ADC values to voltage equivalent and save to spreadsheet
8. Read into MATLAB and run scripts

Expected Result: The outputs to be sinusoidal in nature with less than 5% offset to actual value. Calculated impedance should match data in the reference study within 5%.

Result: The collected data was processed and the final output can be seen in figure 4.1.10b. Data collection, transmission, and processing worked properly.

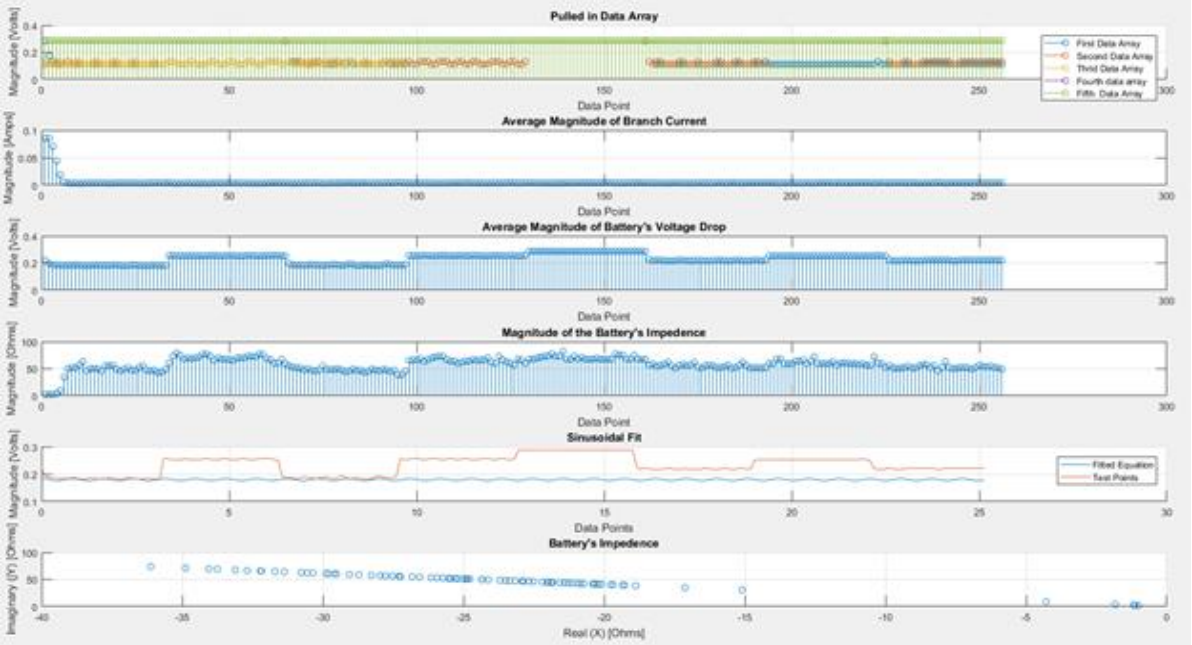

Figure 4.1.10b

The collected data was sinusoidal as shown in the top graph of figure 4.1.10b. The data gathered by the Arduino was communicated and processed. The data was processed to obtain the battery's impedance as shown in the bottom graph of Figure 4.1.10b. This calculated data was similar to the data in the reference study.

4.1.11 Arduino IDE and MATLAB Tool Box Experiment

Objective: Test whether the Arduino toolbox for MATLAB is suitable for our data capture needs as well as see if the Arduino Analog to Digital Converter (ADC) will meet our needs.

Setup: Figure 4.1.11a is a diagram of the setup used to perform this experiment.

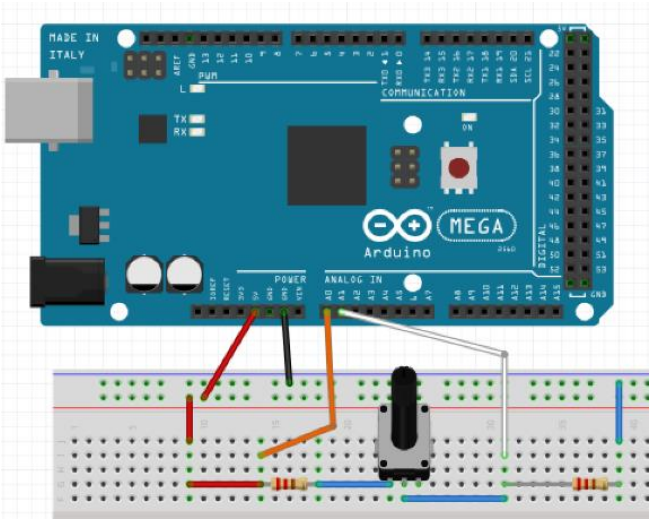

Figure 4.1.11a: A simple variable resistance branch that goes from 5V to ground with a potentiometer. Pins A0 and A1 of an Arduino Mega 2560 are connected in order to capture the voltage drop.

Procedures:

1. Assemble the circuit shown in Figure 4.1.11a
2. Check the voltage level of the circuit using a multimeter
3. Check corresponding Arduino Serial Terminal and MATLAB variable to ensure accuracy of ADC and that communication is established.
4. Check speed of data acquisition and MATLAB data processing
5. Check speed of ADC, communication, and MATLAB data handling

Expected Result: Results from the Arduino ADC and communicated through the serial port would be correct but the communication would be slow but maybe manageable.

Result: The results from the Arduino ADC were correct but the communication was extremely slow. Communicating the data to MATLAB and having MATLAB build the data arrays was too slow and inefficient to satisfy the project requirements.

4.1.12 MATLAB Data Processing Code Test

Objective: Using test data files the object was too

- Dynamically pull in the data to be stored in MATLAB arrays
- Be able to average the multiple arrays while not eating up processing time and memory
- Be able to output the multiple arrays and the extrapolated data in a MATLAB Figure

Setup: Excel files full of dummy data and a MATLAB script in the same file location.

Procedure: Using test data similar to figure 4.1.12a, create and debug code to accomplish the objectives.

1	4.15852902
2	4.09070257
3	4.85887999
4	5.7568025
5	5.95892427

Figure 4.1.12a: Example data

Expected Result: Produce a MATLAB script that pulled in data from an Excel file.

Result: Created a MATLAB function, with a snippet shown in figure 4.1.12b, that dynamically made file names and then pulled in the data from the corresponding file in order to manipulate and process it. The function architecture was chosen due to its handling of intermediate variables being better for memory management.

```
filename = sprintf('%s_%d','TestData',FrequencyRange(FrequencyCount));
filename = strcat(filename, ".csv")

%Pulls columns in from Excel Sheet that the serial terminal will popualte
Voltage1 = xlsread(filename,"A2:A257"); %Gets data for voltage calculations
Voltage2 = xlsread(filename,"A258:A513");
Voltage3 = xlsread(filename,"A514:A769");
Voltage4 = xlsread(filename,"A770:A1025");
Voltage5 = xlsread(filename,"A1026:A1281");
```

Figure 4.1.12b: Snippet of the final developed code

4.1.13 PCB Development

Objective: Learn and eventually implement circuitry in Eagle PCB Design software

Setup: Installed Eagle PCB Design. Watched YouTube videos and read articles about proper Eagle use and proper PCB design procedures.

Procedure:

1. Start on dummy circuit to gain an understanding of the program and proper implantation methods. One circuit is shown in figure 4.1.13a. Another is shown in figure 4.1.13b.
2. Send the design, shown in figure 4.1.13c, to a PCB printer to test design practices
3. Start on project implement, shown in figure 4.1.13e, to develop the full project circuit.
4. Send the design to a PCB printer

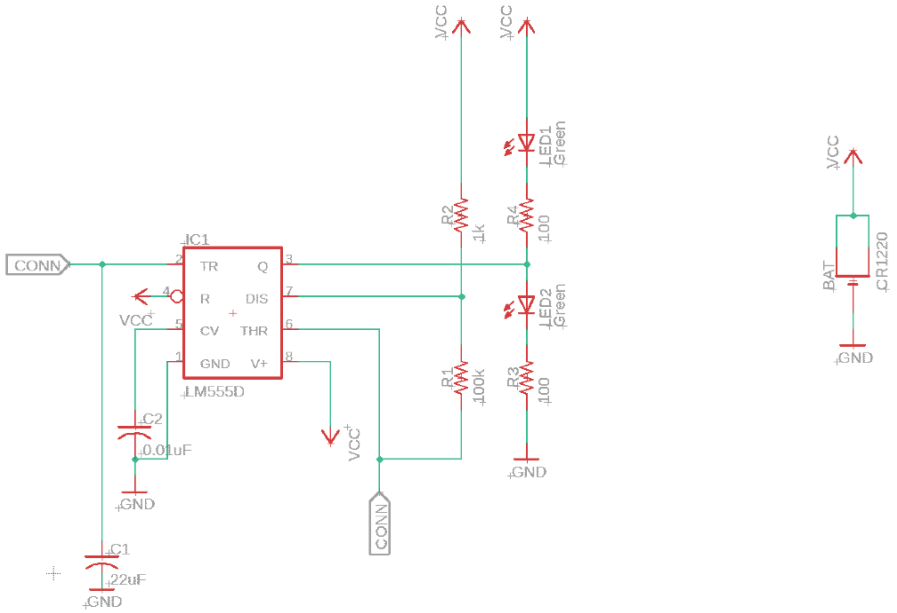

Figure 4.1.13a: First circuit implemented in Eagle. Developed with the help of a tutorial video

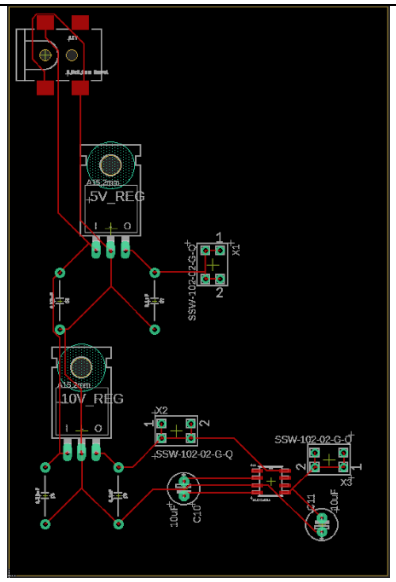

Figure 4.1.13b: Test circuit in Eagle to test printing capabilities and needs

Figure 4.1.13c: Top layer of test circuit printed by EE shop in Chung Hall

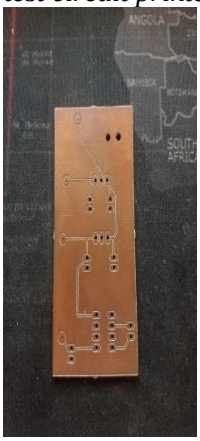

Figure 4.1.13d: Bottom layer of test circuit printed by EE shop in Ching Hall

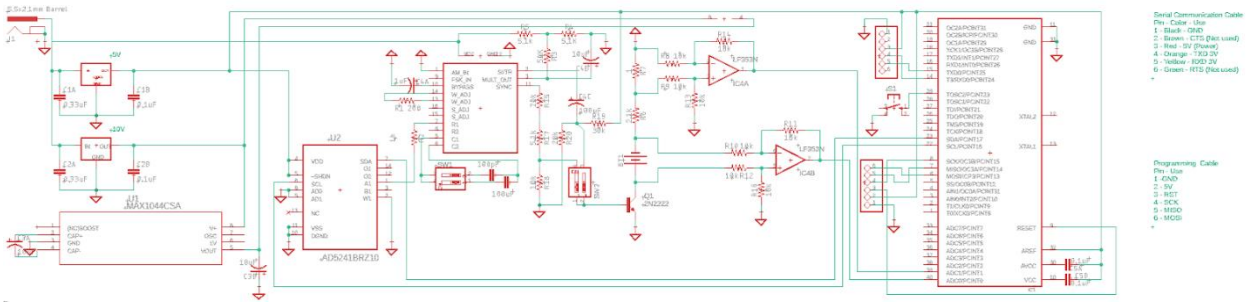

Figure 4.1.13e: Final project circuit currently implemented in Eagle.

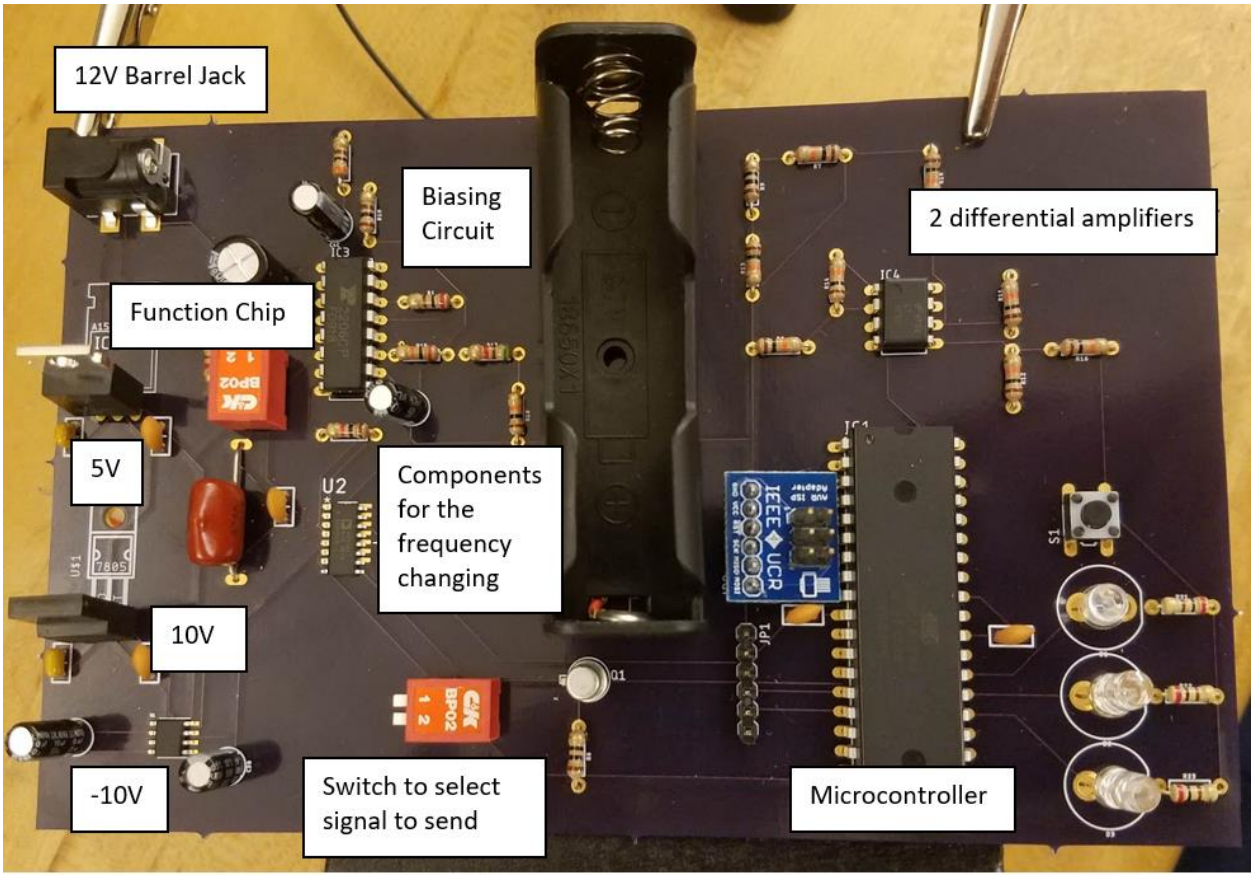

Figure 4.1.13f: Final printed circuit for the project

Expected Result: Quick and easy learning of the Eagle program that would lead to a quick and easy implementation of the project circuit.

Result: The circuit of Figure 4.1.13a provided a basic understanding of the PCB program. The circuit of 4.1.3b was the first test design for the project. It was designed to implement a power system for the circuitry. When printed, shown in figure 4.1.13c and 4.1.13d, the shortcomings of the EE shop printing for our system was shown. The complexity and sensitivity of our circuit made printing difficult. Figure 4.1.13e shows the final circuit implemented in Eagle with Figure 4.1.13f showing the printed version of the system. Implementation of a prototype PCB board for the project was achieved the development process was very difficult with printing being a major difficulty due to the complexity of the circuitry and mixed part types.

4.2 Experiment Results, Data Analysis and Feasibility

The project was proven to be feasible through the performed experiments. The experiments show that that the data collection system gives similar values to the experiment this is based on. The hardware modules are reliable, safe, and communication can be established between the microcontroller and user's computer.

Experiments 4.1.1, 4.1.2, and 4.1.4 proved that the required current could be safely generated and injected into the battery. Experiment 4.1.5 further proved current generation's feasibility because it proved that the frequency range was possible to generate. Experiment 4.1.3 proved that the differential amplifier configuration worked properly and its precision was enough for our purposes. Experiments 4.1.6, 4.1.7, 4.1.9, 4.1.10, 4.1.11, and 4.1.12 proved that data collection, formatting, and processing was possible. They also proved that theoretical and experimental data was similar through development. Experiment 4.1.13 took the results of the previous experiments and developed the final system. It developed the full circuity and then designed and implemented a PCB to finalize the project.

The current injection induces a voltage response by the test battery. This response can be accurately captured, stored, and transmitted to allow for processing. This processing allows for the battery health trends to be discovered and explored. This is the basis of the project. The experiments proved this process is possible and this project is feasible.

5 Architecture and High-Level Design

5.1 System Architecture and Design

Figure 5.1.1 shows the high-level block diagram of the system. It shows how the various subsystems interact with each other and their basic connections.

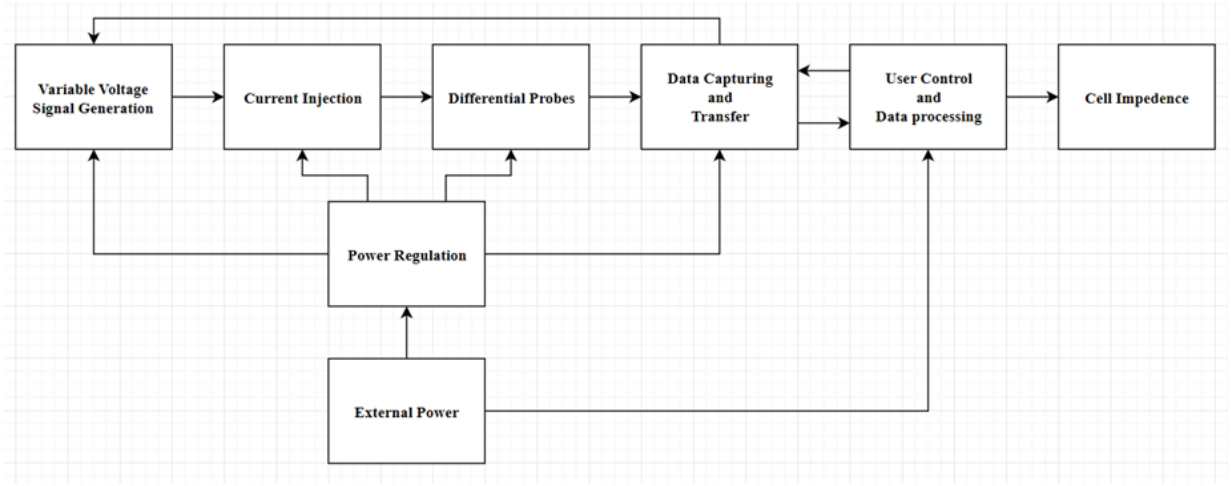

Figure 5.1.1 Block diagram for project circuitry

HARDWARE:

Variable frequency Sine and Square wave generation: The variable frequency sine/square wave generation component is used to power the current generation component of the system. The ability to switch frequencies within f ∈ [1 Hz, 1 KHz] allows for various impedance measurements.

Current generation and injection: The current injection induces a voltage response at a set frequency within the battery. This response is captured by the differential reading component. The current injection module needs to be able to provide a current of the appropriate sinusoidal shape and magnitude i ∈ [1 mA, 20 mA].

Differential reading: Differential probes are placed in specific parts of the test circuit in order to measure the appropriate data. The differential reading component gathers data on battery voltage and value of the injected current. The current value is measured via the voltage of a small load resistor in series with the overall module. These voltage readings are then processed in order to obtain impedance data.

Power distribution and voltage level division: Power is taken from a 12 V power adapter and is then placed into the circuit. This provides a 12 V line that powers the function chip and a line that can be manipulated to get other voltage levels. A 5 V voltage regulator was used to produce a five-volt voltage line for the microcontroller, biasing circuitry, and digital potentiometer. A 10 V regulator was used to produce a 10 V voltage line for the differential amplifiers and the voltage inverter. A negative inverter also uses the 10 V voltage line to provide a -10 V supply voltage to the differential amplifiers.

Microcontroller: The microcontroller is the initial data processor. Using the built-in 10-bit ADC, the voltage values from the differential probes are quantized. Immediately after being quantized, the values are then stored in an array waiting to be transmitted to a computer connected by Universal Serial Bus (USB).

The USB cable used contains a Future Technology Devices International (FTDI) FT232RL USB/Serial chip that takes in Transistor-Transistor Level (TTL) signal logic values that the microcontroller outputs when using USART. The chip then produces a signal based on the USART initialization on the microcontroller that the computer can read.

SOFTWARE:

Serial Communication: The microcontroller reads the ADC data and builds an array of values. It then sends the array via USART to a terminal and is captured on the processing computer.

Data Formatting: The data sent by the microcontroller and captured through the terminal cannot be understood directly by MATLAB. The file has to be formatted using a Python script and a visual basic macro. The formatting code converts the microcontroller data into a voltage value that can be used by MATLAB to determine impedance.

Cell Impedance Calculation: The system provides data for the MATLAB code to calculate the battery impedance. The code takes out the nominal voltage of the battery, corrects the data to account for the differential amplifier gains, and then averages the trials to reduce noise influence on the results. Once the average battery voltage values and current values are computed, the impedance can be calculated. First, a sinusoidal approximation of the voltage array is created in order to calculate the phase shift of the voltage. After this calculation is performed the magnitude of the impedance is calculated and the real and imaginary parts of the impedance are extracted via the phase calculation. After all the calculations it outputs a graph of the data at the different frequencies for the user to see.

5.2 Hardware Architecture

The hardware of this project consists of circuitry, a microcontroller, and a computer. The circuit is comprised of the Variable Frequency Voltage Generation module, Current Generation and Injection module, and the Differential Readings module produces the analog data. The microcontroller captures the analog data, converts it to digital values, and transmits it to the computer. The computer processes the data and shows the results to the user. The microcontroller and computer also communicate user parameters to change system specifications.

Figure 5.2.1 shows the general block diagram of the system. The circuitry is divided into four blocks: Variable Frequency Voltage Generation, Current Generation and Injection, Differential Readings, and Power Distribution and Voltage Division.

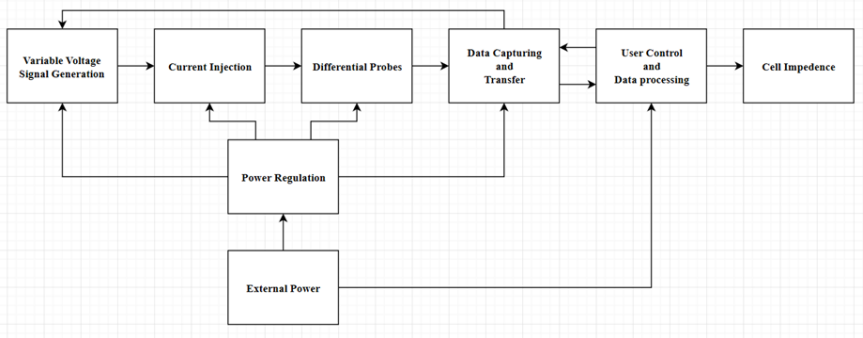

Figure 5.2.1 Block diagram for project circuity

The Variable Frequency Voltage Generation module produces a sinusoidal and square wave voltage signal. Either the sinusoidal or square wave is selected and sent to the Current Generation and Injection module. The frequency is set by a resistor-capacitor network. This network has a digital potentiometer to allow for the network value to be changed digitally. The digital potentiometer value is set by the microcontroller for testing. The Current Generation and Injection module injects the current into the battery producing a response which is captured by the Differential Readings module. The Differential Reading module captures the response of the battery and a dummy resistor, amplifies the response with a predetermined gain, and then presents the response to the microcontroller for digitization.

The microcontroller is the transition between the circuitry and the computer. It interfaces with the Differential Readings module to digitize the collected data. It also interfaces with the Variable Frequency Voltage Generation module to allow for the frequency of the input signal to be changed through user input from the computer. This frequency changing allows for the testing frequency sweep to be accomplished.

The computer interfaces with the microcontroller to obtain the needed data for processing. The data goes through multiple processing levels once transferred to allow for the required computation. Once the computations are completed the results are presented to the user. The computer also allows the user to set parameters in the circuitry.

The following sections development was led by Jack Gu:

- Variable frequency Voltage Generation
- Current generation and injection
- Differential reading

The following sections development was led by Jack Gatfield:

- External Power Supply
- Power Distribution and Voltage Division
- Computer
- Cell Impedance

The following sections development was led by Joseph Gozum:

- Microcontroller

The generic block diagram shown in Figure 5.2.1 is implemented in a through board prototype is shown in Figure 5.2.2 and a printed circuit board in figure 5.2.3.

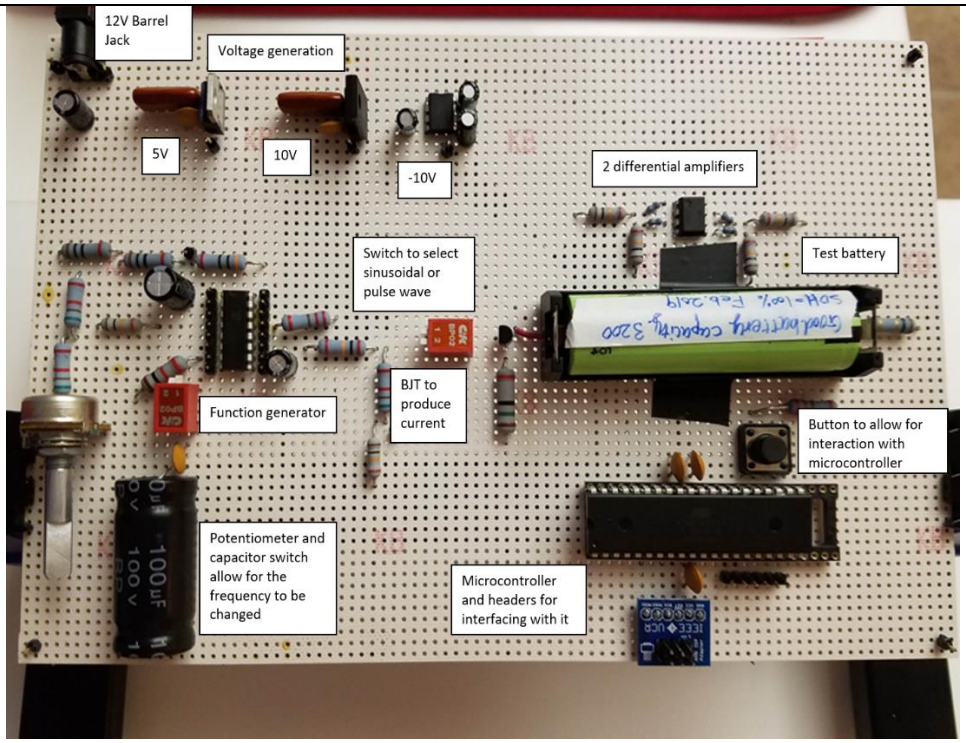

Figure 5.2.2 Through Hole version of the circuity

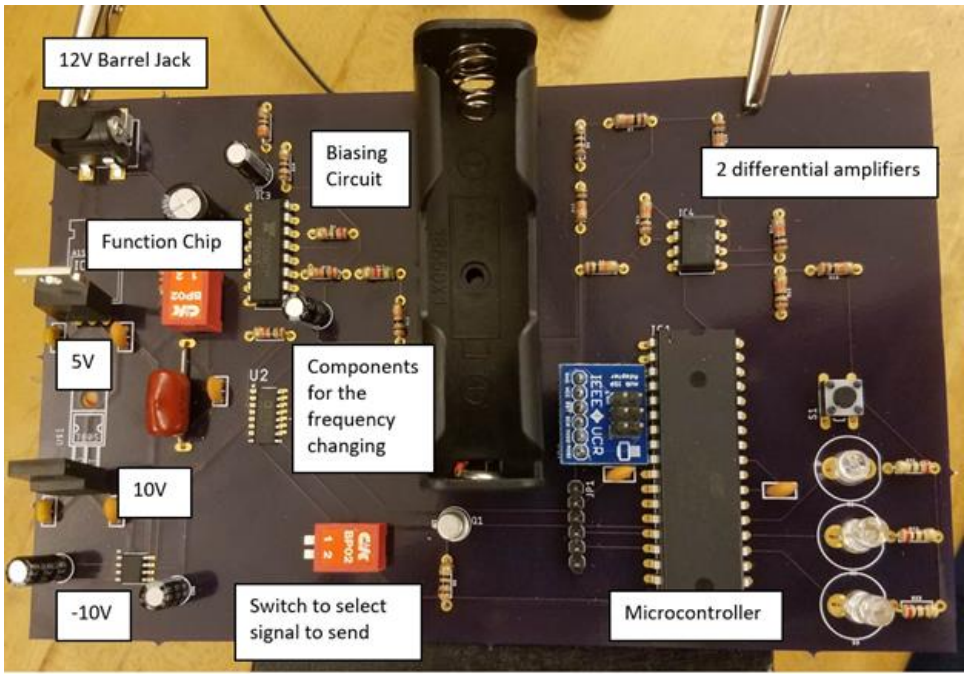

Figure 5.2.4 PCB version of the circuity printed and assembled

Table 5.2.1 shows how the system block diagram shown in figure 5.2.1 was implemented. This implementation was shown in figure 5.2.2 and figure 5.2.3.

Block Diagram Module	Executed System
External Power Supply	12 V Barrel Jack
Power Distribution and Voltage Division	5 V, 10 V, and -10 V voltage regulators
Variable Frequency Voltage Generation	Function generator, potentiometer, and capacitor switch
Current Generation and Injection	Signal selection switch and BJT
Differential Reading	Two differential amplifiers
Microcontroller	Microcontroller and support pin headers
Users computer	User's Computer

Table 5.2.1 Block diagram execution

The External Power Supply and Power Distribution and Voltage Division module are designed to safely power the system. The Variable Frequency Voltage Generation, Current Generation and Injection, and Differential Reading modules are designed to safely produce and capture a voltage response in the battery. The microcontroller is designed to digitize the captured response and user-defined system parameters. The computer uses the digital data and calculates the impedance.

5.3 Software Architecture

For software, the project uses a microcontroller (Atmega1284P) programmed in embedded C, whose state machine is shown in Figure 5.3.2 and a computer. There is also a computer that formats and processes data that has its own state machine as seen in Figure 5.3.1. The microcontroller must communicate with and to the computer. The computer keyboard provides parameter inputs that tell what parameters to set in software for capturing data. The one parameter that needs setting is the digital potentiometer resistance. Once the value is set, the microcontroller collects voltage data. The computer then receives data from the terminal in hex format and then formats it to the corresponding decimal voltage equivalent.

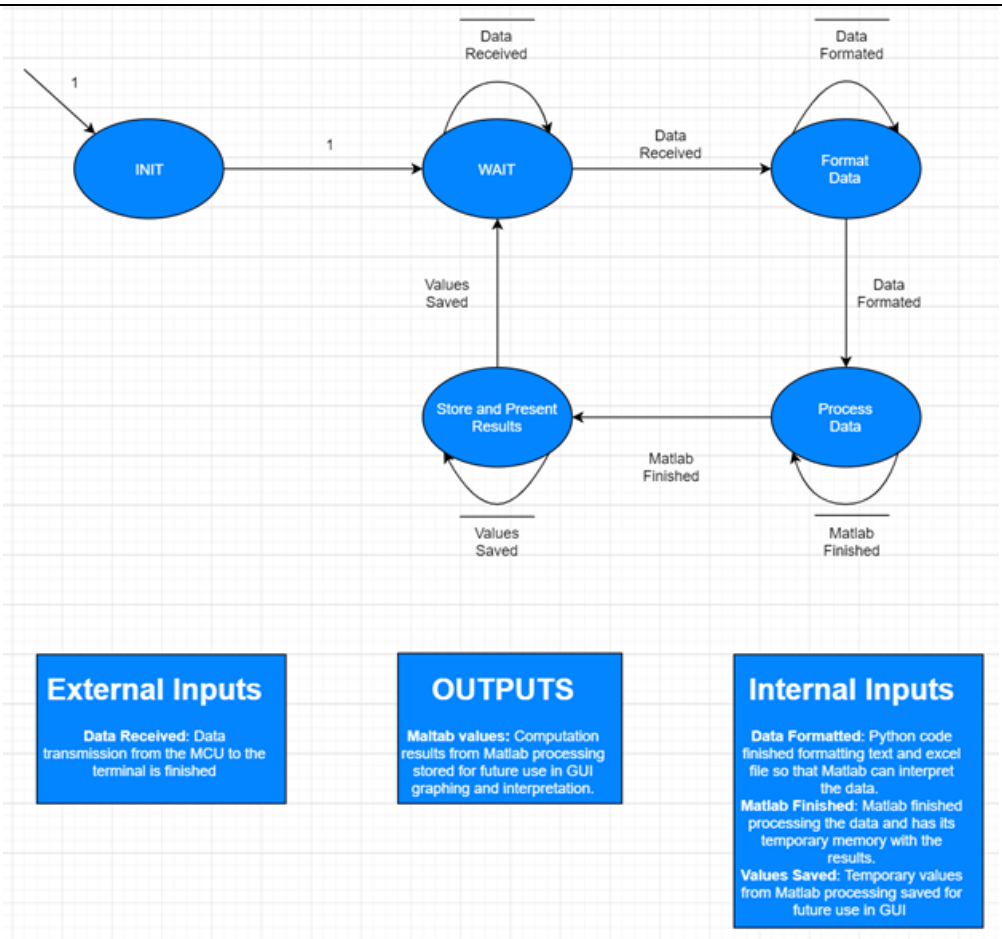

Figure 5.3.1 State machine for computer processing

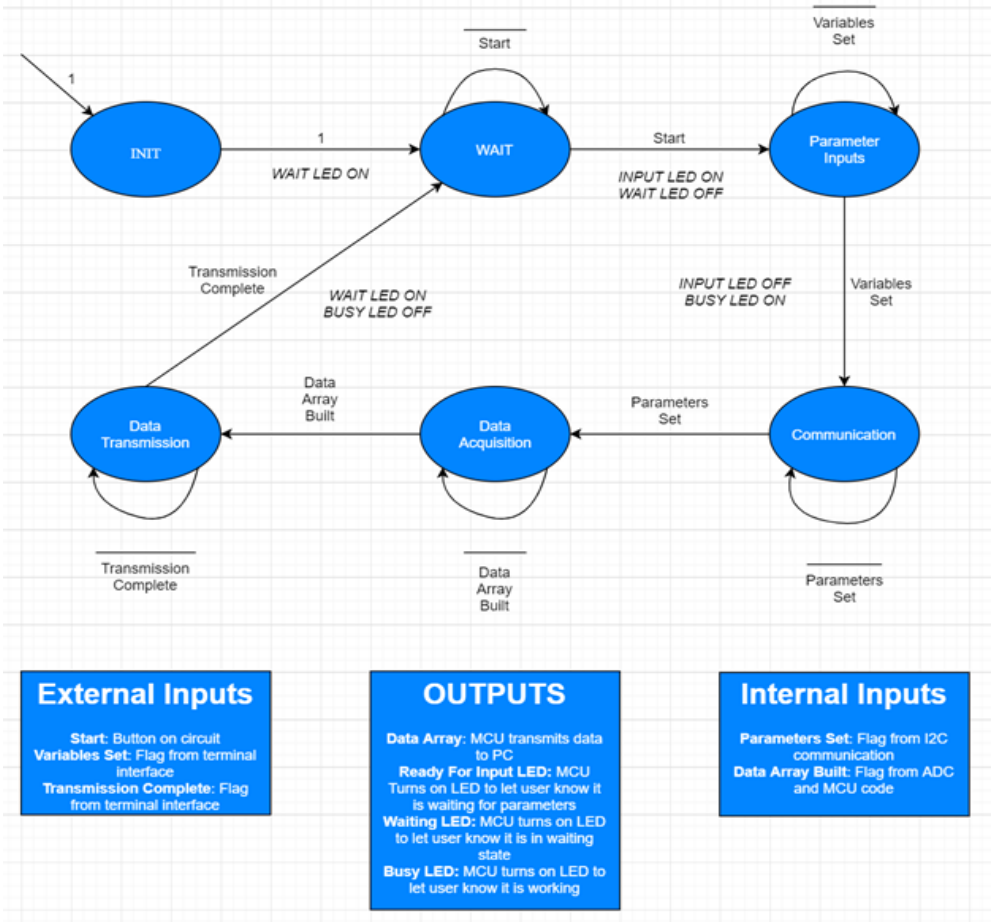

Figure 5.3.2 State machine for microcontroller processing

5.4 Rationale and Alternatives

The rationale of the hardware architecture focuses on the most direct approach to acquire the data and calculate impedance. To consider other architectures or approaches would be unnecessary unless some other functionality was needed.

A Finite-State Machine representation was chosen for the software architecture because the device is only in one specific state of operation at any given time. Changes based on external inputs are in a pre-defined sequence, making a Finite-State Machine optimal choice. This architecture is also not computationally intensive nor deterministic, perfect for implementation on the microcontroller. This architecture is also heavily event-driven which is required for the system because certain components need to act based on certain external event/stimuli.

The main rationale for the project is optimizing the system to be able to easily implement while still obtaining accurate data. For the hardware modules, the most straightforward and efficient circuitry was taken to allow for easy data acquisition and minimal error occurrences. For the software programs, a desire for compatibility on different operating systems ability to interact with the microcontroller was prioritized.

6 Data Structures

Internal:

- N/A

Global:

- Arrays
  - ADC values from across the battery and current reading resistor
  - Formatted ADC values for MATLAB to process.
  - Calculated Impedance values
  - Test Frequencies
- User-defined structure
  - Contain input terminal value corresponding with one of 256 wiper positions on the digital potentiometer, testing delay, etc.
- Floating Point Integers
  - Battery nominal voltage level
  - Differential amplifiers gain
  - Test Frequency Array

Temporary:

- Text files containing the terminal output after microcontroller sends data.
- Excel files containing the formatted terminal output
- MATLAB
  - Arrays to read in the data files for the current frequency
  - Various arrays to allow for manipulation and computation to calculate the impedance

6.1 Internal software data structure

- N/A

6.2 Global data structure

- Arrays containing ADC value across the battery and current reading resistor.
- User-defined structure containing the input terminal value corresponding with one of 256 wiper positions on the digital potentiometer.
- Batteries nominal voltage level
- Differential amplifiers gain
- Test Frequency Array

6.3 Temporary data structure

- Text files: Created from the terminal output that contains raw data sent from the microcontroller, afterwards converts to a voltage equivalent and stored in a CSV file.
- Arrays: Created during the data processing as intermediate values in order to obtain the impedance data.

6.4 Database descriptions

- N/A

7 Low Level Design

7.1 The Current Injection Circuit

When designing the current injection circuit, the main goal was to design a current source that continuously generated a sinusoidal current without performance issues, regardless of frequencies.

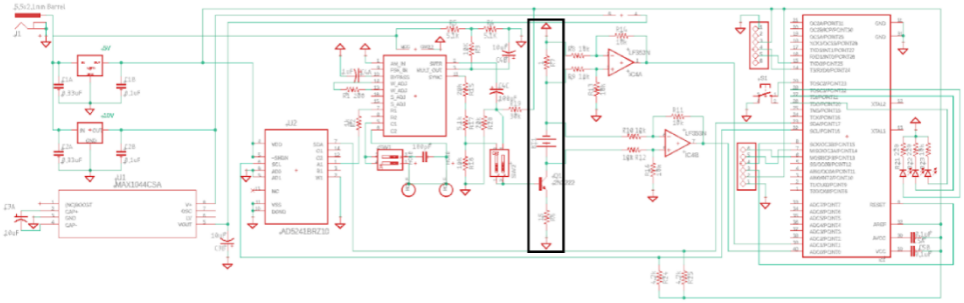

Figure 7.1.0: Module of the current injection source in the overall schematic

The current module is placed between the function generation module and the two differential modules, as seen in Figure 7.1.0. The input to the module is the voltage output from the function generation module. The output to the module is the voltage reading taken by the differential amplifier modules.

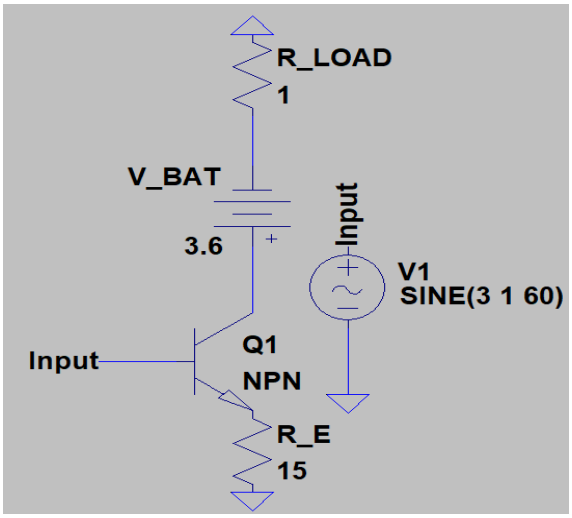

Figure 7.1.1: Final schematic for the current injection circuit that injects sinusoidal current into the test battery for data acquisition.

The final chosen schematic for the module is shown in Figure 7.1.1. The model takes the most direct approach while accounting for safety and current requirements discovered in the design process.

7.2 Processing Narrative for Current Injection Circuit

Safety was the main concern in the initial design, resulting on a design focused on safety and current stability rather than providing a strong current (i > 1 mA). This approach was taken due to the volatile nature and response when current is directly injected into a battery.

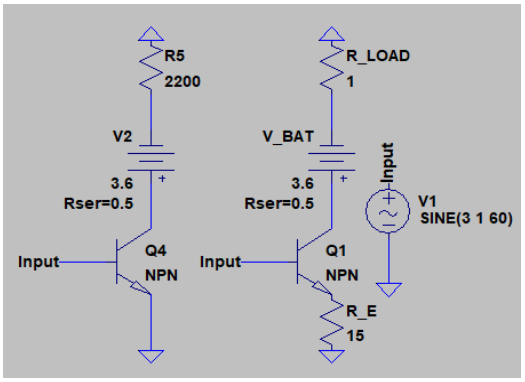

Figure 7.1.3a: LTSPICE schematics for both the previous and final implementation of the current injection circuit. Note that the input is only an external sinusoid for the test schematic and in implementation comes from the function generator chip (Module 4)

The first circuit design was the current mirror schematic (Figure 7.1.3). The basic current mirror and Wilder current mirror appeared as the most viable design choices. After receiving feedback to keep the current injection circuit design simple, the simple current mirror was chosen as the final design choice.

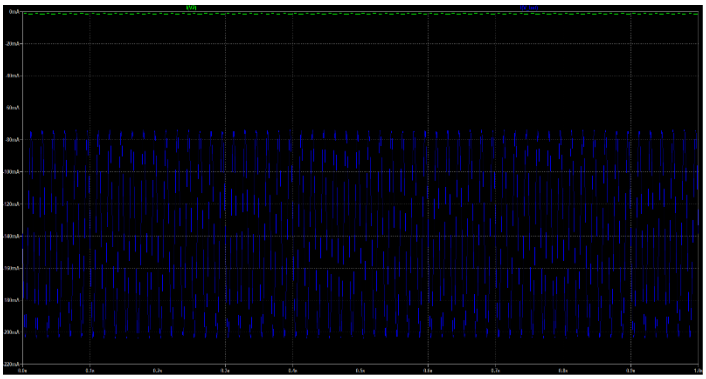

Figure 7.1.3b: Comparison of output currents across respective batteries. The green current indicates the left model in Figure 7.1.3b while the blue current indicates the current from the right model in 7.1.3b

Initial simulation and testing of the module resulted in no issues. However, issues appeared when multiple modules were tested together. The first issue occurred attempting to induce a response from the battery with the differential probes (Module 3). With the current output being in the μA range (Green graph in Figure 7.1.3b), no discernable differences from the nominal DC voltage could be measured. Current amplification was needed before the injection process. Amplification resulted in lowering the load resistance as much as possible while concurrently preventing power issues and circuit meltdowns. Thorough testing resulted in a current that wouldn't exceed 18 mA. An emitter resistance was added to enhance this current as well as dissipate power. Testing found that the best emitter resistance values for the module falls within the range of [15 Ω, 20 Ω], with the higher end for increased stability but lower current output and vice versa.

The second issue was the unexpected impedance when connecting the function generator chip module (Module 4) to the current injection circuit. Connection results in an unexpected drop in the input voltage when connecting the voltage output to the current injection circuit. This change reduces the DC offset and drops the input voltage from an input of 2 + 0.9 sin (wt.) to just simply 0.9 sin (wt.). Increasing the output of the bias was needed to compensate for this loss. Without the bias, the BJT only produces a half-wave sinusoid of the respective frequency since the nature of the BJT rejects negative voltage input. After slowly increasing the bias of the biasing circuit to test until the input to the current injection circuit no longer looks like a half-wave, the best bias value found was 4 V, rather than the initial 2 V.

7.2.1 Current Injection Circuit Interface Description

The current injection circuit comprises of a BJT (model 2N2222) to supply a sinusoidal current that is injected into the test battery. The input for testing purposes comes from an external sinusoidal voltage source, but in implementation it comes from the biased output of the biasing circuit.

In regard to output in relations to other modules of the system, it outputs the voltage responses that is read by the differential probes. These responses are measured from the test battery $V_{battery}$ and the current measuring resistive load. The output of the $V_{battery}$ should be the nominal DC value with a slight deviation over time for the voltage response. The V_{Load} across the battery should be similar in magnitude to the desired injected current of [9 mA, 15 mA], meaning the V_{Load} falls in the range of [9 mV, 23 mV], with the extra voltage due to slight extra resistance in the load resistor.

7.2.2 Current Injection Circuit Processing Details

The current injection circuit starts off with the BJT which serves as the core component of the current injection circuit. The 2N2222 BJT model was chosen for being readily available and low-cost. Branching off the emitter branch of the BJT is the emitter resistor, a high power (2 W) 15 Ω resistor. The current injection set up induces a current to flow from the collector to the emitter, meaning that a sinusoidal current within [9 mA, 15 mA] flows through the test battery, current measuring resistor, and emitter resistor. The test battery is the standard 18650 Lithium-Ion battery and the current measuring resistor is another high-power resistor (2 W) with a resistance of 1 Ω.

With this setup, when sending a sinusoidal input voltage (V_{input} with amplitude [1 V, 3 V] and frequency [1 Hz, 1 KHz]), the current injection circuit induces a current ($I_{collector}$ with range [9 mA, 15 mA]) and frequency similar to the input frequency.

This setup does have issues. Power dissipation and safety was the largest of the issues and also served as the largest constraints. The amount of measured voltage response is proportional to the magnitude of the injected current. However, increasing the current brings up the issues of power dissipation in the circuit. This constraint resulted in optimizing the circuit to have minimal overall impedance while concurrently maintaining a circuit that could run without issues. While current resistor values help maximize performance as well as maintain a safe operating system, the issue of transient response is not solved with these resistors. To tackle this issue, it is advised to not switch frequencies too quickly when performing manual measurements at specific frequencies. For frequency sweeps, a mechanism is needed to disconnect the battery after testing a frequency before switching frequencies in the sweep. This prevents transient responses from occurring that could potentially damage the battery and the test circuit. However, this delay in switching frequencies causes performance issues, specifically in elongating the testing time for frequency sweeps.

7.3 The Function Generator Biasing Circuit

The biasing circuit module is implemented to correct the function generation module's output. The output of the bias is corrected because the natural bias provided the function generation module is not suitable for the current injection module.

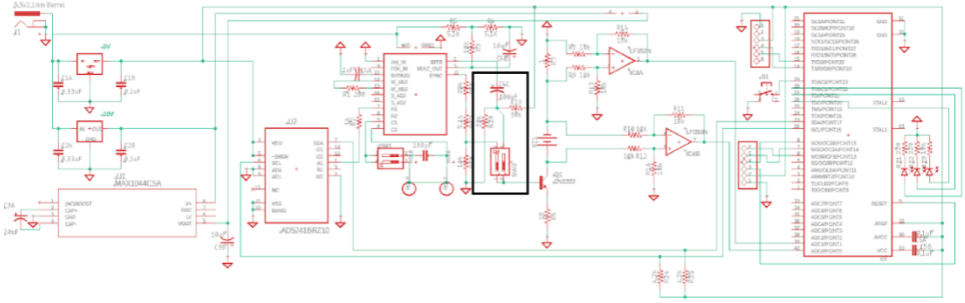

Figure 7.2.0: Module of the biasing circuit in the overall schematic

The position of the module in relation to the overall circuit is shown in Figure 7.2.0. The input of the biasing module is the output from the function generation module. The biasing module fixes the bias of the voltage output and outputs it to the current injection module.

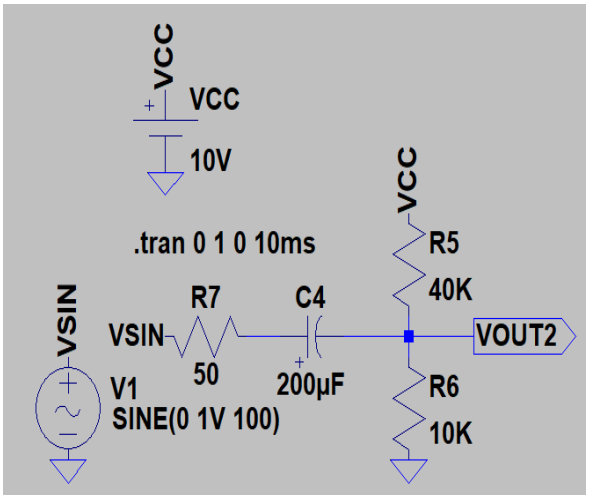

Figure 7.2.1: Test circuit for the biasing circuit that biases the output of the function generator chip.

After extensive testing, the final design and component values can be seen in Figure 7.2.1. Higher capacitance results in better biasing but lowers the sinusoidal magnitude (and vice versa) while resistor values just changes the values of the new bias.

7.3.1 Processing Narrative for the Function Generator Biasing Circuit

Initially this module was not supposed to exist in the final system implementation. However, this implementation was needed due to the inherent flaws of the function generator chip (XR-2206). Specifically, the chip automatically biases the voltage with an offset that's approximately half of the power supply used to power the chip, there would be a minimum offset of 5 V since the chip requires a power supply of at least 10 V (power supply range of [10 V, 26 V]).

To tackle this issue, a biasing circuit was implemented. A common emitter amplifier was proposed initially, but a simple biasing circuit was enough for the project's needs. When building the biasing circuit, the resistor network was simple as it was just a voltage divider following the formula:

$$V_{out} = \frac{V_{CC}R_6}{R_5 + R_6}(DC) + \sin(\omega t)$$

The difficulty in designing this module was the choice of the capacitor. While a lower capacitance preserved more of the voltage swing from the input, a lower capacitance means that the biasing itself is poorly filtered, resulting in a flat DC signal as the output. Having a higher capacitance alleviates this issue but in return causes a decrease in voltage swing, with the worst case in being half of the original swing (2 V_{pp} to 1 V_{pp}). After extensive testing and consideration of the test bias circuit (module three), a 100µF polarized capacitor is chosen as it only causes a 10% swing decrease (2 V_{pp} to 1.8 V_{pp}) while preserving the bias for the entire frequency range.

7.3.2 Function Generator Biasing Circuit Interface Description

The function generator biasing circuit takes an input voltage and re-biases it according to the parameters of the circuit components. Input for the biasing circuit is V_{SIN} and the input range for voltage is approximately 6 + 1 sin (wt). To power the biasing circuit, a power supply of 10 V is provided by the 10 V power regulator from the power network.

The output for the biasing circuit is a voltage that's approximately 4 + 1 sin (wt) and is fed into the current injection module. Output in the test schematic is shown as V_{OUT_2} and represents the connection between the biasing circuit module and the current injection module.

7.3.3 Function Generator Biasing Circuit Processing Details

The biasing circuit starts by filtering out the DC bias provided by the input (The output of the function generator chip). This is done with a capacitor that goes in series with the input. Note that there is an inherent tradeoff with the capacitor value. This constraint forces us to choose between DC filtering performance and swing reduction. Higher capacitance values perform a better job at cancelling the initial bias of the input signal but in return shrinks the voltage swing of the output. Although in the schematic the capacitor value is stated to be 200 µF, the best and most appropriate value for the capacitor should be 100 µF. This allows for appropriate filtering of the input bias at all frequencies in the sweep while concurrently causes minimal reduction in the output swing (~0.1V or 10% decrease in the overall swing).

The next part of the biasing circuit is the voltage divider component of the module responsible for the magnitude of the re-biased input. The offset of the output can be found from the voltage of the center of the divider, meaning that resistor values must be adjusted accordingly to achieve the desired bias. For testing purposes, a general 10 V power supply was used, but in implementation a 5 V power supply from the 5 V regulator chip in the power network was used. The main constraint for this part of the circuit is the available power supply, as the power supply determines the maximum possible bias of the output. Fortunately, the output with the regulator chip's power is sufficient for the current injection module.

7.4 The Differential Probes

The differential probes measure the voltage response from the current injection module. The module is comprised of two differential amplifiers, one for the test battery and one for the resistive load. The differential amplifier then amplifies the response by a factor of nine.

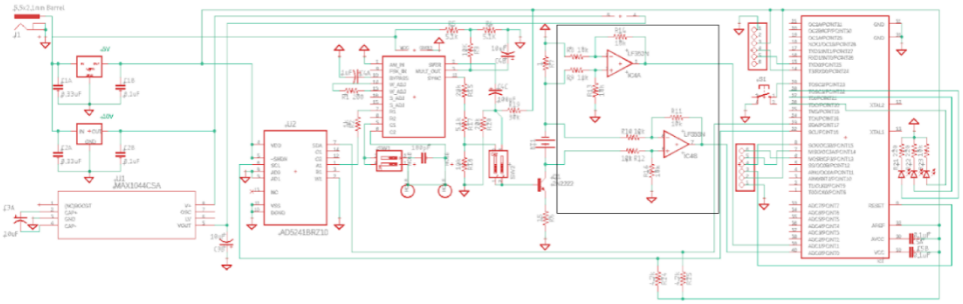

Figure 7.3.0: Module of the differential probes in the overall schematic

The differential probes are placed to the left of the microcontroller and to the right of the current injection module, as seen in Figure 7.3.0. The input of the differential probes are the voltage responses from the test battery and resistive load from the current injection module. The differential probes then feed the output to the microcontroller to be further processed.

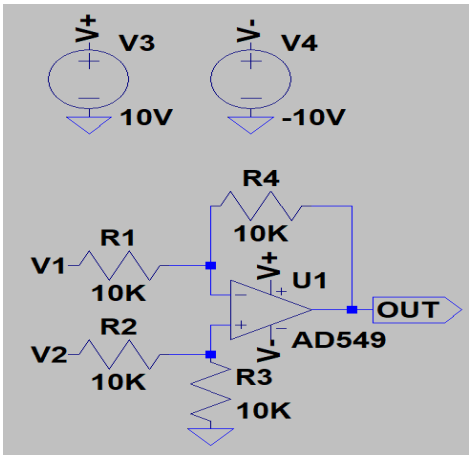

Figure 7.3.1: Test circuit for an individual voltage differential probe

The final model for an individual probe is shown in Fig 7.3.1. Note that two separate models make up the module in the final design.

7.4.1 Processing Narrative for the Differential Probes

A direct approach was used for the design of the differential probes, leading to a differential amplifier. Although the signal was a sinusoidal with an offset, an OP-AMP based differential amplifier is sufficient since only the magnitude of the voltages from the test battery and current measuring resistor/resistive load matters. In other modules, the values of the gathered differential voltages are used for reconstructing a sinusoidal figure for calculation.

The design in straightforward, using a simple differential amplifier circuit for both voltage differential measurement probes. The resistor network used in the differential probes are not important so long as they are in a set ratio for the desired gain (Unity or 10). The gain for both measurement probes were initially all one, but during testing it was hard for the voltage across the resistive load to be measured due to low magnitudes [9 mV, 23 mV]. Amplifying the signal of the difference by a magnitude of 10 allowed for better pickup while also staying within the range of measurable voltage ranges for the data acquisition module (Module 6), [0 V,5 V]. Future implementations can result in higher gains for clearer data acquisition, but is presently unneeded and can cause potential safety risk or unwanted noise.

7.4.2 Differential Probes Interface Description

There are two differential probe circuits in the overall module, resulting in two inputs: the voltage differences across the current measuring resistor and the test battery. Both components measured are found in the current injection circuit module. Voltage difference for the battery is expected to be around the nominal DC value of the battery with deviations from the forced response from the battery. Similarly, the voltage across the current measuring resistor to be around the range of [9 mV, 23 mV].

There are also two outputs. The first output just passes the voltage difference across the battery to the data acquisition module (Module six). The second output requires an amplification in order to help improve the data acquisition and processing. This means that the values of the resistors used in the second probe circuit must be modified. The test circuit resistor one and two values are reduced by 10 to a value of 1 KΩ. The output is then multiplied with a gain of 10, improving output range for the second probes to [90 mV, 230 mV]. Both outputs are then sent to the data acquisition module that reads these values at a more precise level than conventional multimeter or oscilloscope measurements.

7.4.3 Differential Probes Processing Details

The probes offer a flexible circuit that could be easily adjusted. There are two types of differential probes: probes that simply read the voltage difference (Gain of 1) and probes that amplified the voltage difference by a factor of 10 (Gain = 10). The first probe circuit was used to measure and extract the voltage difference over time for the test battery and has no amplification (Gain of 1). This extracted voltage difference is the first output to the data acquisition module with values within the range of the nominal battery voltage with slight deviations. Amplification is required for the second probe before outputted to the data acquisition module. This is done by modifying the values of the resistors of the circuit, based around the formula:

$$A = \frac{R_3}{R_1}(V_+ - V_-); R_3 = R_4; R_1 = R_2$$

This amplification of the voltage difference results in the range [90 mV, 230 mV]. This then feeds into the second output which is also part of the data acquisition modules. The simplicity of the differential probes results in few constraints. Resistor precision is an incredibly important factor, especially since precision and accuracy is important in data calculations. High-quality, incredible precise (±1%) resistors are used to alleviate this error. Power dissipation isn't an issue for the module, as the current that reaches the voltages are inherently low from the nature of the OP-AMP.

7.5 The Function Generator Chip Circuit

The function generator module comprises of a function generation chip (XR-2206) and the components are chosen according to the spreadsheet provided for the chip. The chip is tuned to the following parameters:

- Frequency Range [1 Hz, 1KHz]
- Voltage swing of 1 volt: Vpp = 2V
- Sinusoidal shape

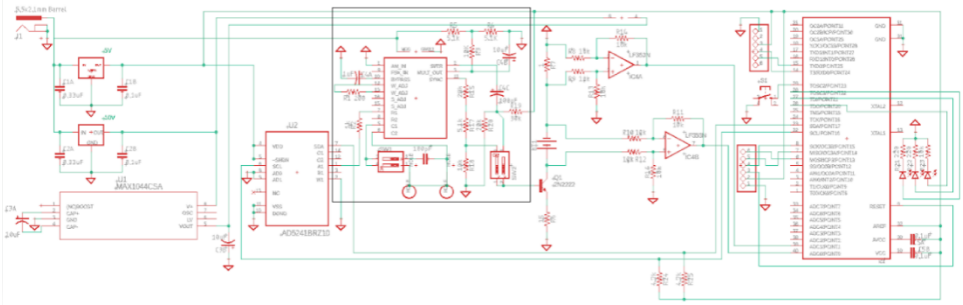

Figure 7.4.0: Module of the Function generator chip circuit in the overall schematic

As seen in Figure 7.4.0, the function generator chip module is placed in between the power inverter and the biasing circuit module. The function generator chip module takes voltage from the power network as input (+10 V and +5 V).

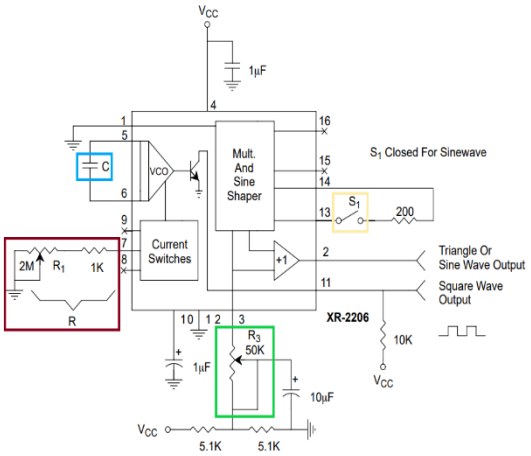

Figure 7.4.1: Internal circuit for the function generator chip circuit

Figure 7.4.1 shows the components modified around the chip. The capacitor in blue and the resistor network in maroon determines the frequency of the output voltage while the resistor network in green determines the amplitude of the output.

7.5.1 Processing Narrative for the Function Generator Chip Circuit

There are several approaches to designing the function generation module. This module needs to easily switch frequencies while also maintaining the amplitude of the voltage needed for the current injection module. Before using the function generator chip, several timers were considered.

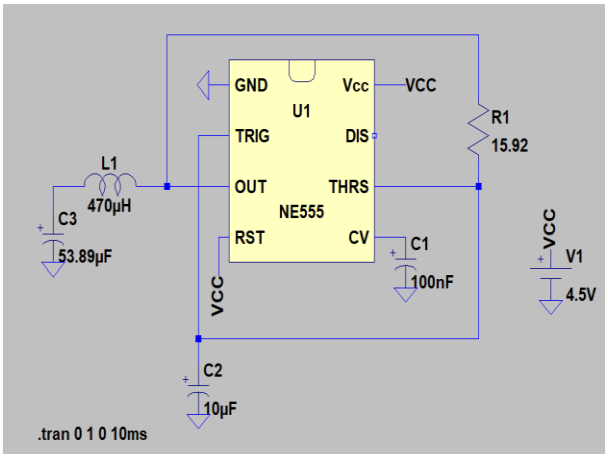

Figure 7.4.1: Schematic for the 555 Timer circuit

The first approach was to use a 555 timer (shown in Figure 7.4.1), but this resulted in noticeable errors in the timer, such as signal slewing as well as the frequency being off. Signal slewing is unacceptable in the final output due to the difficult nature in measuring the output frequency.

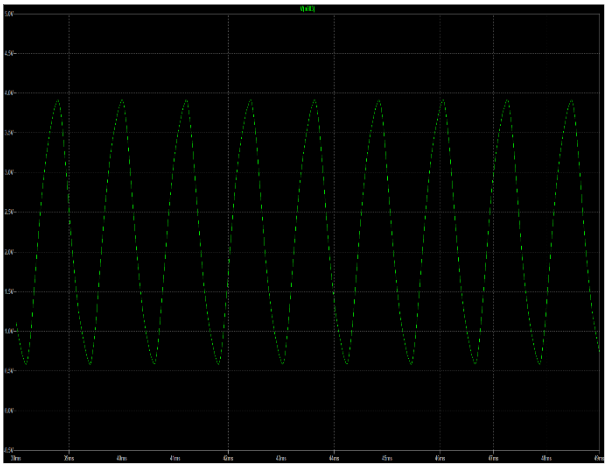

Figure 7.4.2: Output of the 555 Timer.

Figure 7.4.2 demonstrates the signal slewing of the 555 timer. What did not help the timer's cause was the need for more than two capacitances, suggesting the use of a variable capacitance. This adds complications to the timer, leaving more room for error. Browsing around for alternatives lead to the function generator chip used in the final implementation.

7.5.2 Function Generator Chip Circuit Interface Description

The function generator chip circuit module takes in two inputs: a signal indicating which capacitor branch (Figure 7.4.1 Blue box) the module should be connected to and an I2C signal that adjusts the Digi-pot (Figure 7.4.1 maroon box) for specific frequencies. Input signals are matched and coordinated to generate a desired frequency. Refer to the lookup table included for what digital inputs to send for a certain frequency.

The module is powered by a 12 V power supply from the power network, indicated as V_{CC} in the schematic Figure 7.4.0. The output of the function generator chip circuit is pin 2 in Figure 7.4.0 and is a sinusoid due to wiring configurations of pin 13 and pin 14 in Figure 7.4.1. The sinusoid is a signal with DC offset of 6 V, due to the inherent nature of the function generator chip. The AC component has been adjusted to have a swing of 2 Vpp. The output is then sent to the function biasing circuit module (Module two). This output in total is: $6 + \sin(\omega t)$ and is sent to the current injection circuit module.

7.5.3 Function Generator Chip Processing Details

The main constraints lie within the chip parameters itself. The nature of the XR2206 function generator chip does not allow for resistances larger than 1 MΩ to be used while the capacitance can only accept capacitance inputs of the range [0.1µF, 100 µF]. Implementation of the capacitance branch is needed to simplify the digital resistor network. This causes delays in overall processing time, due to the need for capacitors to discharge before switching branches for safety.

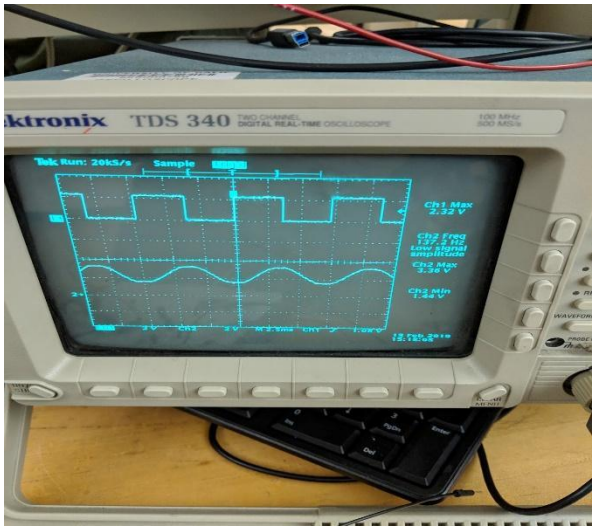

Figure 7.4.3: Output of the function generator chip after biasing (High frequency range)

There is also the performance issue regarding the internal biasing, since it inherently biases the output to way above desired voltage range. The function biasing circuit (Module two) is added to resolve this issue. The frequency range is unchanged and is sinusoidal throughout the entire range, as shown in both screenshots of Figure 7.4.3.

7.6 Digital Potentiometer and I2C

Figure 7.5.0 is the input and output specifications of the digital potentiometer used in this project.

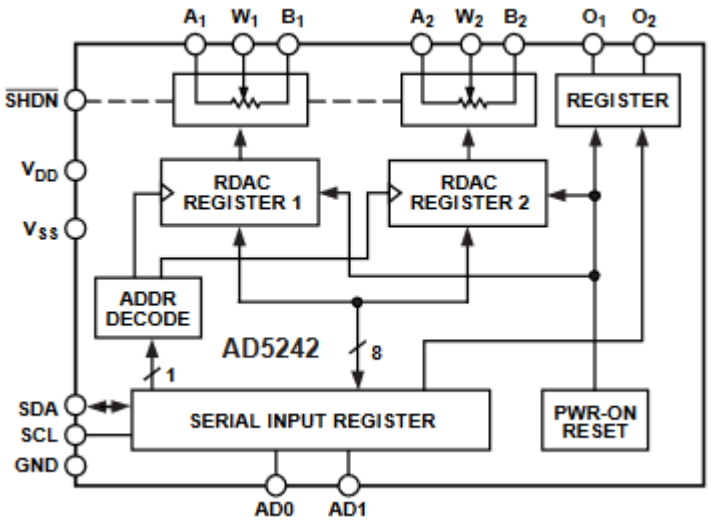

Figure 7.5.0: Input terminals of AD5422 programmed via I2C

7.6.1 Processing Narrative for I2C Communication

During the design phase of the project, after it was found that the XR2206 function generator IC could be used to generate variable frequency waveforms, there needed to be a way to control the frequency. In the datasheet of the XR2206, it shows a 1 MΩ analog linear potentiometer varying the frequency but in order to automate the project, a digital potentiometer (AD5422) is a better replacement. The digital potentiometer allows for a quicker and more precise change of resistance than an analog equivalent.

The digital potentiometer chosen has its resistance set by sending a specific 8-bit value over I2C. In order to properly control the digital potentiometer, I2C communication was implemented on the microcontroller.

7.6.2 I2C Interface Communication Description

Figure 7.5.1 is the physical implementation of the test bench for I2C communication.

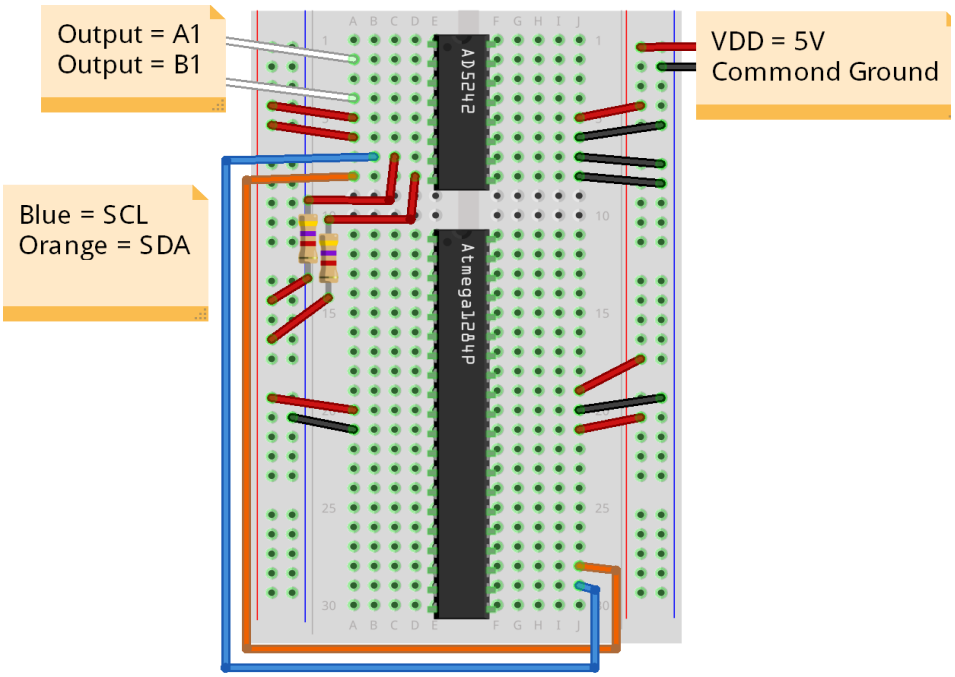

Figure 7.5.1: Physical implementation of the I2C communication circuitry

The digital potentiometer has 13 inputs but only 8 are used in the implementation and two outputs which are the opposite ends of the potentiometer.

Of the 13 inputs, two are AD0 and AD1, these inputs set the last bits of the address of the digital potentiometer to be used during I2C transmission. Both are must be physically set by attaching to an appropriate voltage or grounding them. In the implementation in this project, both inputs are grounded to represent zeros. Therefore, according to the datasheet the address of the Digital potentiometer is 0101100, the first 5 bits are set in manufacturing.

Another 2 inputs are SDA and SCL, these are the data and clock signals respectively. VDD and VSS are power inputs and these inputs are both set to VDD (5 V).

The final input utilized in the Battery Management System device is W1, also known as the wiper of the digital potentiometer. The position is set by a digital signal, containing the position for a specific resistance that is sent through I2C.

7.6.3 I2C Processing Details

Figure 7.5.3 shows an example of an I2C signal being sent on the SDA line and the clock signal on the SCL line.

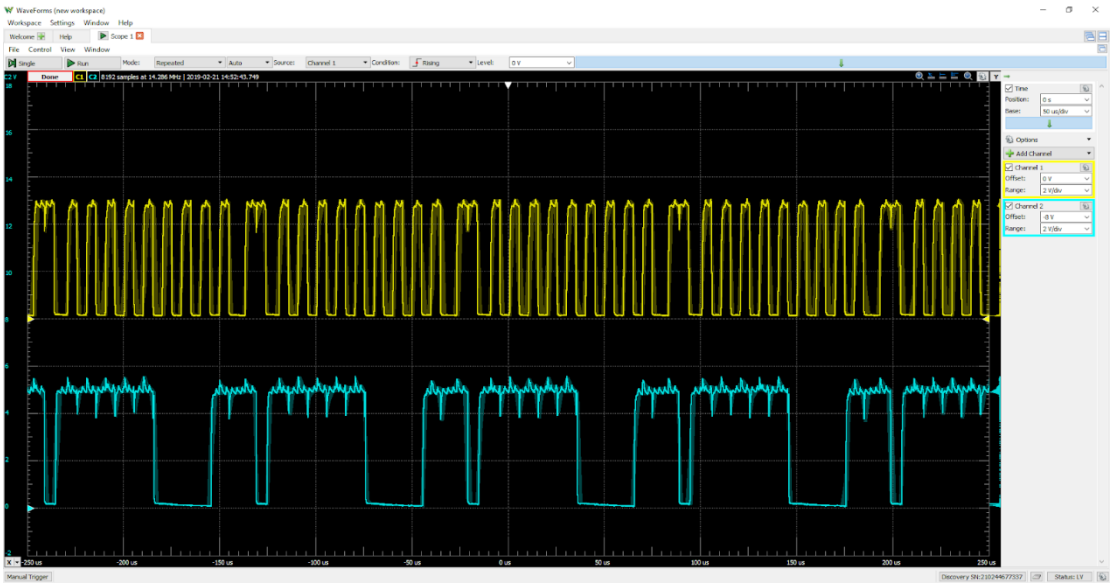

Figure 7.5.3: Output of I2C processing

During testing of I2C communication to the digital potentiometer, the specifications for the I2C standard were difficult to implement. There are many acknowledge signals that need to be generated and acknowledged on both sides, eventually a library was used to implement I2C. The figure above shows the signals generated over the SDA (blue waveform) and SCL (yellow waveform).

The waveforms look correct as the test data to be sent was 0xF8 (i.e., 1111 1000) and that was sent repeatedly, the acknowledge signal is seen before the data that takes up about two clock cycles. Then the data that follows is correct.

I2C is implemented as specified and works as desires. During actual implementation of I2C on the breadboard and PCB, it is important to keep the distance short as I2C is highly affected by capacitance of wire interconnects. If I2C is used more, then it is important to include more pullup resistors (of at least 4.7 kΩ) along the signal lines.

7.7 USART

7.7.1 Processing Narrative for USART Communication

During initial discussion on the physical implementation, the need to send data from microcontroller to an external computer came up. It was understood that the calculations needed to find impedance is too intense to run on a microcontroller. It was decided that processing needs to occur on a computer. RS232 is well-known as the standard for communicating with a PC from a microcontroller. To properly implement the communication standard by hand takes a lot of time and debugging. To save time, a USB cable that includes ICs that implement the RS232 standard were purchased. The cable takes in a USART signal then converts it into an appropriate RS232 signal that can be understood by the computer.

Implementing USART communication is well-documented in the Atmega1284P datasheet and even included examples written in the C programming language. Writing code for this module implementation was straightforward.

7.7.2 USART Interface Communication Description

Figure 7.6.1 shows the wiring configuration of the USB cable that transmit data from the microcontroller to a computer.

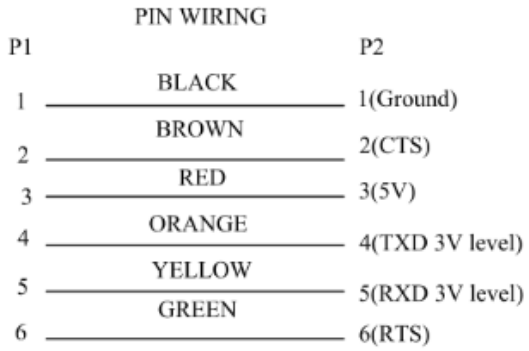

Figure 7.6.1: USART Pin output used in the project

The header terminated end of the USB cable has 6 inputs but 3 can be grounded in this implementation. Ground, CTS, and RTS are connected to ground. CTS and RTS are used for data flow control purposes but are unnecessary in this project. The most important inputs are TXD and RXD, while in the figure above it says they operate at 3 V levels, it can actually handle the 5 V signals outputted by the microcontroller. The TXD (transmit pin) input it connected to the RXD output on the microcontroller and the RXD input connected to the TXD output on the microcontroller. This allows the USB cable to receive the transmitted data from TXD on the microcontroller and transmit to the RXD (receive pin) on the microcontroller.

USART implementation details:

- 9600bps (Baud rate)
- 8 data bits
- 1 stop bit
- No parity checking
- No flow controls

The main reason for the 9600 bps is that it provides the least amount of error during transmission compared to higher baud rates, the higher rates are difficult for small microcontrollers to reliably send data. In this device, the quality of the data is the most important aspect. The baud rate during testing only leads to several second transmission times of hundreds of bytes. 8 data bits was the easiest to send and receive without having to add extra software control. Parity checking was not implemented due to time constraints and only being useful across noisy mediums. Flow control was also left out because it is just not needed for this device.

7.7.3 USART Processing Details

Figure 7.6.2 shows an example of the output from the microcontroller being read on a computer via a terminal emulator.

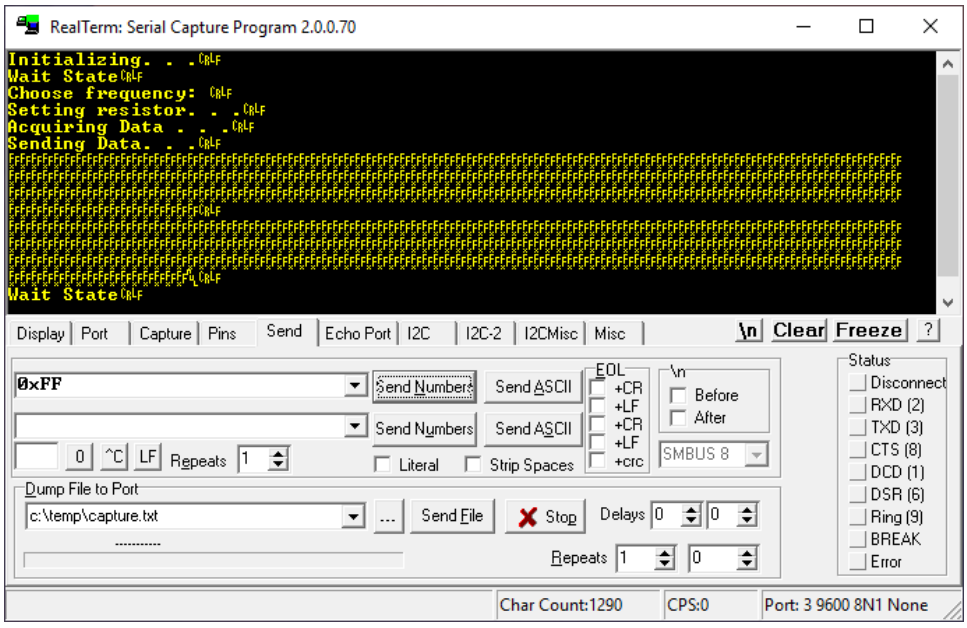

Figure 7.6.2: USART Interface using RealTerm for debugging

The implementation works just as desired. Data is able to be acquired on microcontroller and be successfully transmitted to the computer. The only limitation of this module is that only 8 bits can be sent at any given time by USART from the microcontroller, causing a time delay, but the overall transmission time occurs only over several seconds.

7.8 Data Acquisition

7.8.1 Processing Narrative for Data Acquisition

During the initial planning of the device, it was apparent the ADC would be an important component of the design. The ADC quantities the output of the differential probes which is the data needed for impedance calculations. The internal 10-bit SAR-based ADC in the Atmega1284P will be used, it can read changes in the desired range (i.e., the mV range). A higher bit ADC provides higher precision of data calculated but for the current iteration the internal ADC works well.

7.8.2 Data Acquisition Interface Description

The implementation of the ADC is simple and only involves setting the bits of four different 8-bit registers that are memory-mapped on the Atmega1284P microcontroller. After setting the registers, the ADC is set to Free-running mode. In this mode, the ADC is continuously quantizing values from the selected input channel. The selected channel is based on certain bits set in one of the four setup registers. One channel is used for the differential reading of the battery and is set to 1x gain. Another channel is used for the differential reading around the current reading resistor at 10x gain.

7.8.3 Data Acquisition Details

In Figure 7.7.3a the equivalent voltage values of the 10-bit ADC values after being sent and processed on a computer are shown.

Voltage Reading
3.78299120234604
3.67546432062561
3.63147605083089
3.6119257086999
3.6119257086999
3.6119257086999
3.6119257086999
3.6119257086999
3.6119257086999

Figure 7.7.3a Output reading of the ADC

The Atmega1284P microcontroller comes packaged with a 10-bit SAR-based ADC that allows the quantization of the differential probes.

7.9 MATLAB

The circuitry gathers data for the user computer to process through MATLAB. The complex impedance and the magnitude of the impedance are calculated for one frequency and for the full frequency sweep. This project was focused on finding the impedance of the battery, but the data can be further extrapolated to give more answers.

7.9.1 Processing Narrative for Data Processing

Data processing is a crucial component of the system. The circuitry was developed to provide the necessary data to allow for the impedance to be calculated. Due to the development team's prior knowledge, it was decided to use Python, Excel, and MATLAB to perform the necessary data formatting. Python and Excel format the data from the raw USART files. Python takes the USART files, converts the data from raw hex strings to individual integer values and stores the values into an Excel file. The Excel file removes remnant elements from the Python processing and saves the data for MATLAB to access. MATLAB reads in the file and build data arrays for them impedance calculations.

7.9.2 Data Processing Interface Description

The implementation of the MATLAB code has many supporting steps. These steps are needed to allow for the data to be transferred, using USART, and formatted, using Python, to allow for processing once these steps have been completed, Excel files of readable data are saved in the proper directory for the MATLAB function to manipulate it.

Originally the code was to be executed through a script but this was improved to be a function call.

```
function FrequencySweepData = FrequencySweep(BatteryVoltage, CurrentResistor,CurrentGain)
```

Figure 7.8.2a Frequency Call Header for frequency sweep data processing code

The function call gives the project further versatility in developing a GUI in the future but it also improves program speed. MATLAB functions handle intermediate variables differently than scripts, so the program became less intensive. In future work, optimization of the code is needed in order to further reduce execution time.

7.9.3 Data Processing Details

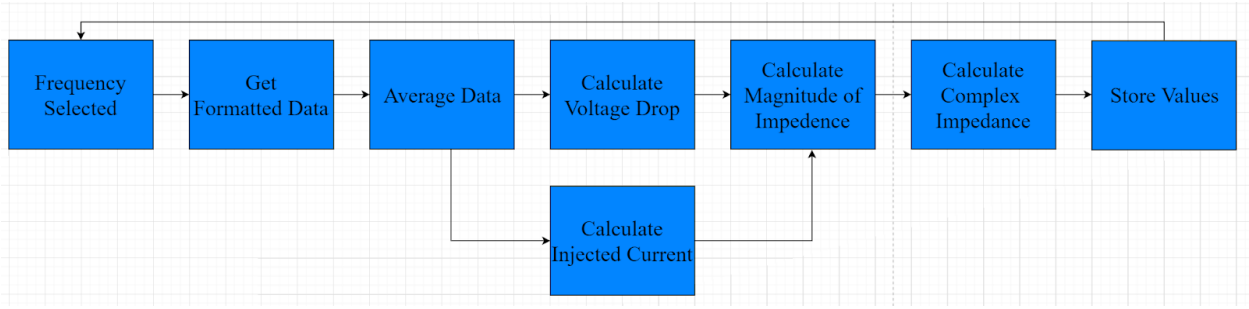

Figure 7.8.3a High level block diagram of data processing

The MATLAB program pulls in the file values and builds arrays to allow for the needed calculations. First, the program averages the multiple trials per frequency to reduce noise and outliers. Then the nominal voltage of the battery is removed from the average battery voltage to get the voltage drop. The current resistor value is manipulated to get the current through the battery branch. With the voltage and current values, the magnitude of the impedance can be calculated.

The complex impedance is also wanted so the phase shift of the data is needed. A default sine wave at the proper frequency and a sinusoidal approximation of the voltage data are created to calculate the phase shift. With this phase shift the complex impedance can be calculated and stored. The values of the magnitude and complex impedance at each frequency are computed and then output to the user.

This process can be executed at a single frequency (this was used for testing and development) and have those results outputted but the frequency sweep allows for trends to be seen giving more information about the batteries health.

7.9.4 Data Processing Important Code Snippets

Below are a few important snippets of code and a description of its functionality.

```
filename = sprintf('%s_%d','TestData',FrequencyRange(FrequencyCount));
filename = strcat(filename, ".csv")
```

The above code is used to create a dynamic file name. This is needed because the code has to loop through the frequencies and process the proper data. To obtain the data it has to be able to access the corresponding file. Dynamic name generation does require the saved files to follow a naming convention but it allows the processing code to operate more efficiently.

Figure 7.8.4a Code snippet of the file name creation system for the frequency sweep

```
myfit = NonLinearModel.fit(PhaseRunner,Voltage_AvgMag,
                           'y~b0+b1*sin(b2*x1+b3)';
                           [Average_Battery_Phase,
                           Amplitude_Battery_Phase,
                           Peak_Battery,
                           Phaseshift_Battery]);

Z_PhaseShif_radians = acos(dot(myfit.Fitted,TestSin)/(norm(TestSin)*norm(myfit.Fitted)));
Z_PhaseShif_degrees = Z_PhaseShif_radians * 360 / (2*pi);
```

The above code is a portion of the code used to calculate the phase shift between the injected current and the current experienced by the battery.

Figure 7.8.4b Code snippet of the phase calculation

```
AvgVoltagePull(FrequencyCount)              = mean(Voltage_AvgMag);
AvgCurrentPull(FrequencyCount)              = mean(Current_AvgMag);
AvgBatteryImpedence(FrequencyCount)         = mean(BatteryImpedence);
AvgComplexBatteryImpedence(FrequencyCount)  = mean(ComplexImpedence);
```

This takes the final computations and does a final averaging of the data points data and then saves them to another array in order to output the information at the end of the frequency sweep.

Figure 7.8.4c Code snippet of the final data saving to be used in the final output

7.10 The Insufficient Current Problem

During the initial testing, the microcontroller was unable to extract data from the voltage reading across the battery. The differential reading component was only reading the nominal DC value of the battery rather than the induced change caused by the current. Upon looking further into the problem, the issue behind this was that the current injected was insufficient in inducing a response within the battery.

7.10.1 Solving the Insufficient Current Problem

To solve the problem, the current injection component of the circuit was modified to increase the current injected (200 µA increased to 15 mA). This was done by adding a resistor to the emitter as well as reducing the load in the test circuit from 2,200 KΩ to 1 Ω. Through extensive testing, the best resistance to be added to the emitter was found to be $R_E \in [15\ \Omega, 20\ \Omega]$. Any lower resistance added to the emitter causes a safety hazard, with the increased injected current being too high, causing the circuit to breakdown and melt. Any higher resistance added to the emitter leads to insufficient change in the circuit.

7.11 How/Where do we Process the Data that we Collect Problem

During initial planning of the project, it was discovered the data collected from the ADC isn't practical for calculations, conversions, and processing on the microcontroller itself. The microcontroller is limited in memory, ability to handle floating point numbers, and just overall computational power required to find the impedance of the battery.

7.11.1 Use Python/MATLAB to do Conversions and Processes the Data

To compensate for the limited computational power of the microprocessor it was decided to us a separate personal computer. The data is transmitted to the user computer and the computer handles the processing to extract the impedance from the data. It was found that the communication between the microcontroller and the computer produces file unreadable by MATLAB. This unreadability was solved through using a Python script. Python takes the raw data and convert it from the 10-bit value acquired by the ADC to the floating-point voltage equivalent, then it placed into a CSV file that is used by MATLAB. From here, MATLAB takes the data and processes it to find the Magnitude and Phase of impedance, break it up into its real and complex components, and then display this in graphs to the user.

7.12 The Injected Current Cutoff Problem

When measuring the output of the function generator circuit after biasing, it was found that the biased voltage and swing was sufficient and close to ideal specification for the current injection circuit. However, when testing components together, the effect the current injection circuit had in power dissipation was not accounted for. This initially resulted in the biased signal to be insufficient, and it resulted in the input voltage and therefore the injected current to be cutoff. This cutoff was due to it crossing the operational threshold of the transistor causing it to shut down.

7.12.1 Solving the Injected Current Cutoff Problem

To fix this issue, the biasing of the function generator chip was increased. Additionally, sources of extra resistance were found and minimized without affecting the systems performance. While the exact reason for this unexpected drop in the input voltage for the current injection circuit is unknown, the biasing circuit was adjusted to account for this unexpected modification to the input.

This readjustment accomplishes the task of preventing cutoff that would skew the data while concurrently keeping the voltage range low enough to where the current injection circuit could operate.

7.13 The Power Dissipation and Battery Safety Issue

With the problem described in 7.9 the issue of power dissipation arose. With a higher current flowing through the circuit a higher power was being experience by all the circuit parts. This higher power was causing parts, especially the BJT and the emitter resistor, to heat up after short testing periods. This heat would cause the BJT to become compromised, creating a major safety hazard. It could cause a huge current (greater than 10 A) to flow through the circuit and potentially compromise the rest of the circuit and most dangerously the battery. Power dissipation created the need to balance power ratings with the resistor values. This balancing was needed in order to maximize current magnitude but maintain safety.

With the problem described in 7.13 changing the frequency of the signals caused unsafe transient responses. Having large frequency steps during the sweep created large transient current responses that could compromise the safe handling of the battery and circuit parts. Lowering the injected current was shown to be a feasible solution to the unwanted transient responses but this led to not inducing a detectable voltage response from the battery.

7.13.1 Solving the Power Dissipation and Battery Safety Issue

The power dissipation resistors used in the current injection modules had power ratings of two Watts. Resistance values for the emitter and resistive load were chosen based on power ratings and current maximization. High injection current plays a critical role in generating raw sufficient data.

Transient response when switching frequencies is another safety issue. Preserving the manual input and frequency sweep feature, delays were introduced to allow the system to settle before switching frequencies. This delay varies depends on the type of input, with frequency sweep containing low delay times while manual inputs called for much higher delay times. Frequent delays go against lowering overall testing time, but is a necessary tradeoff to ensure system reliability.

7.14 The Frequency Range Generation Problem

When originally designing the function generation frequency RC network it was discovered that the full frequency range was not easily obtainable with a single capacitor. The values needed for the capacitor and the potentiometer were not available in commercial products. The plan was to have one non polarized capacitor value of 0.1uF and to use a digital potentiometer to change the RC network value. It was determined that with the original capacitor value a potentiometer range of 0.1 Ω to 10 MΩ. This range is not feasible in hardware.

7.14.1 Solving the Frequency Range Generation Problem

Two capacitor branches with a digital switch to select the branch was implemented. The switch allows for the capacitor value to be changed, allowing the function generator chip to output the entire frequency range.

Frequency	Resistance (Ω)	Capacitance(F)
1 Hz	10K	100µF
7 Hz	1.25K	100µF
...	...	...
1 KHz	10K	0.1µF

Table. 7.18.1: Table of lookup values for specific frequencies. Refer to the full table included for other frequencies

Through experimentation and calculations, it was determined to use a 100 uF capacitor for lower frequencies and a 0.1 uF capacitor for higher frequencies. These capacitor values allow the resistor of the RC network to stay at values available to the potentiometer. The relationship between frequency, resistance, and capacitance can be seen in table 7.18.1. This solution added onto the complexity of the function generation module, but it is needed to reasonably implement the frequency sweep.

8 User Interface Design

Microcontroller:

Data Acquisition at a Single Frequency

The main user interface for the microcontroller is through a terminal window. Communicating via USART, the microcontroller sends strings telling what state of operation the microcontroller is in (e.g., Initialization, Wait, Setting Digital Potentiometer, Acquisition, etc) after each statement of the state of operation. During the choose frequency state, the microcontroller requests a HEX value that has been mapped to a specific frequency.

Data Acquisition of Frequency Sweep from 1 Hz to 1 KHz

The user interface will be a terminal window. The terminal will print out the name of the state of operation the microcontroller is currently in and then a newline. The terminal will also print out the specific frequency that is currently being used for data acquisition and then the data that was sent over on the next newline.

Computer:

Python: A standard command shell/terminal interface.

MATLAB: MATLAB command window.

8.1 Application Control

Microcontroller:

Data Acquisition at a Single Frequency

Typical behavior is a static terminal window (RealTerm terminal emulation software was used here), where sentences appear on separate new lines that state what mode of operation the microcontroller is in. In the choose frequency state, the microcontroller will halt operation (and no newlines will occur) until user input is given. During the sending data state, the microcontroller sends all acquired data from the ADC to the terminal, this is required in order to capture the data and use it later for processing. The most important aspect is that all the data from a single frequency acquisition period must all appear on one line of its own. This is the basic configuration and design of the user interface with the microcontroller and can be extended as desired.

Data Acquisition of Frequency Sweep from 1 Hz to 1 KHz

The typical design of the application control will be a terminal window. The user only needs to press a specified button and the microcontroller will start acquiring data from every frequency in the predetermined range. Once the frequency is set, at each frequency a hardcoded amount of time for data acquisition and appropriate resistor values will be set automatically. The most important aspect is that all the data from a single frequency acquisition period must all appear on one line of its own.

Computer:
Python: Command shell/terminal in order to run the formatting script.

MATLAB: MATLAB command shell to run the corresponding formatting script.

8.2 User Interface Screens

Microcontroller:

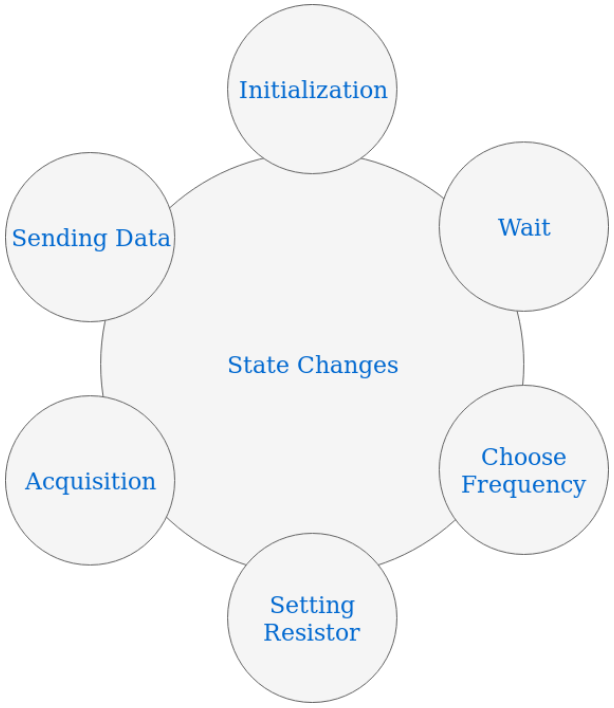

Figure 8.2a Microcontroller state diagram

Data Acquisition at a Single Frequency

The microcontroller will have a singular, long-running screen. At each instance of a new state, the microcontroller sends the name to the terminal window. The microcontroller can take input at specified points. After all, states have executed, the system will repeat at the Wait State to start a new set of acquisition, this configuration can be seen in Figure 8.2b.

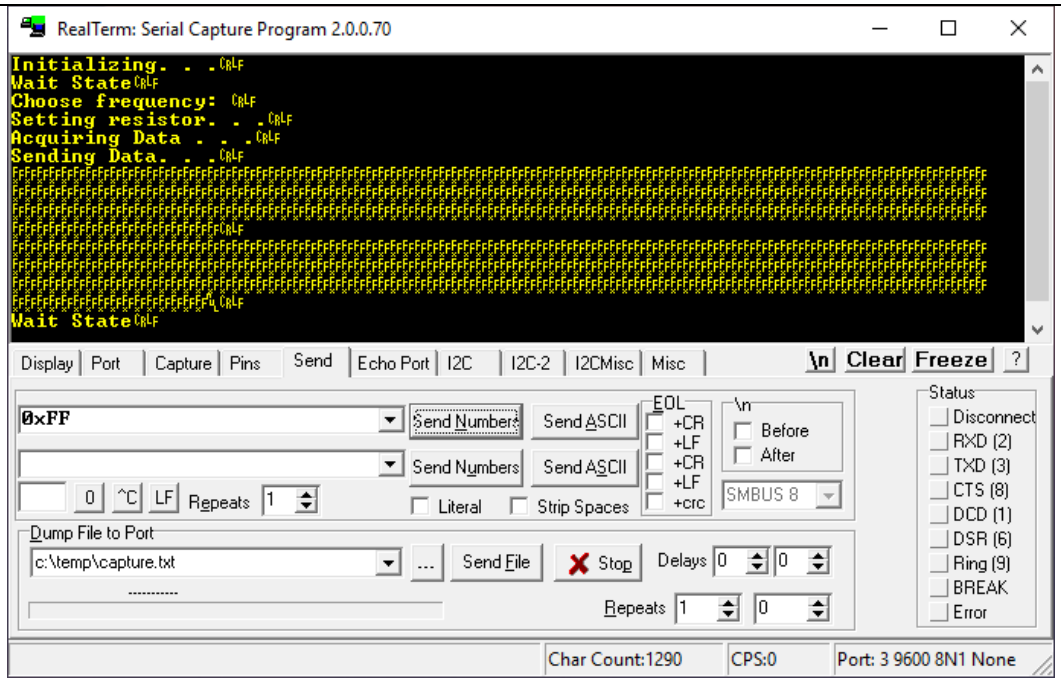

Figure 8.2b Generic design of a microcontroller user interface

Data Acquisition of Frequency Sweep from 1 Hz to 1 KHz

The same format as above (Data Acquisition at a Single Frequency) will be followed but repeated for each frequency within the sweep range.

Computer:

Data Acquisition

To start the circuit in collecting the proper data a simple run button would be had with a brief summary of the process that will occur when the program is running.

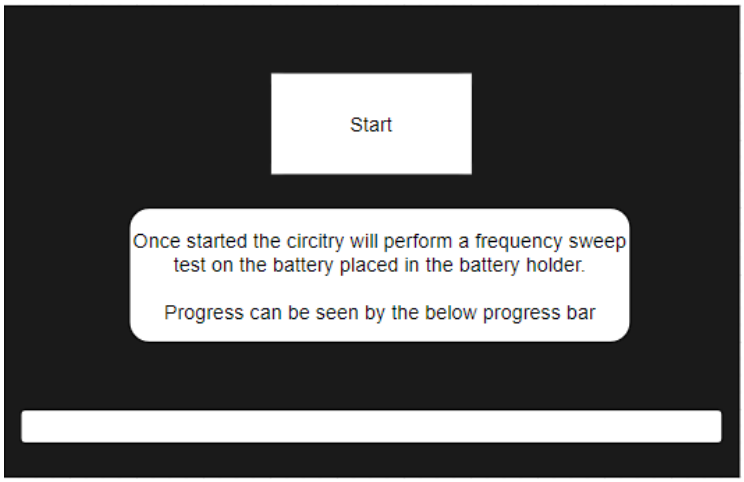

Figure 8.2c Example of a program begin and running menu of a LabVIEW window

Data processing

Once the data acquisition stage of the system is complete. The computer will transition to data processing functionality. The progress bar will still be progressing as it processes the data. Once the program finishes a new window will appear showing the results and a brief summary of their meaning.

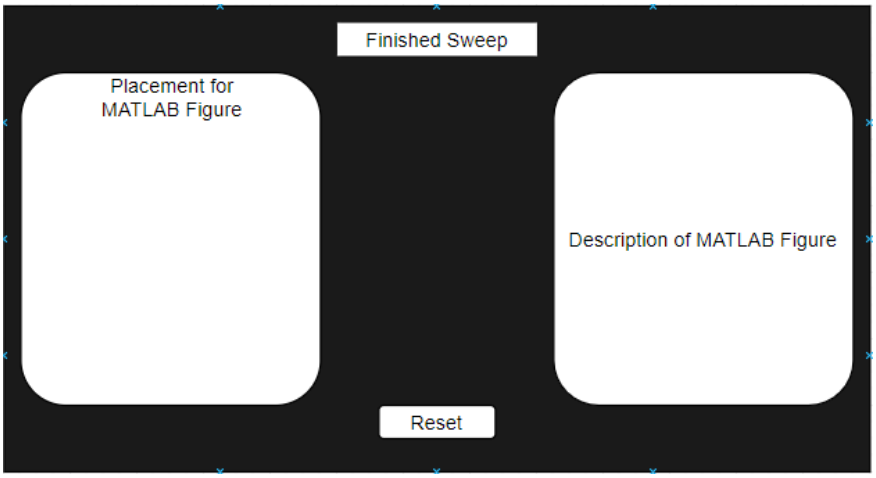

Figure 8.2d Example of a program output GUI in a LabVIEW window

LabVIEW has the capability to run all the needed programs. Using LabVIEW one GUI could be developed to completely execute the data acquisition and data processing systems. However, LabVIEW requires a paid license and then it takes time to develop the GUI based in LabVIEW.

9 Test Plan

9.1 Test Design

9.1.1 Testing for Floating Node Voltages

1. **Objective:** Test for floating node voltages at the transistor's collector component.
2. **Function tested:** The current generation module.
3. **Design objective involved:** Injecting a sinusoidal current into a test battery.
4. **Experiment Setup:** The setup simply involves placing a test battery in series with a resistor and the transistor. A high valued resistor is recommended to fully dissipate power generated from the current running through the test battery.

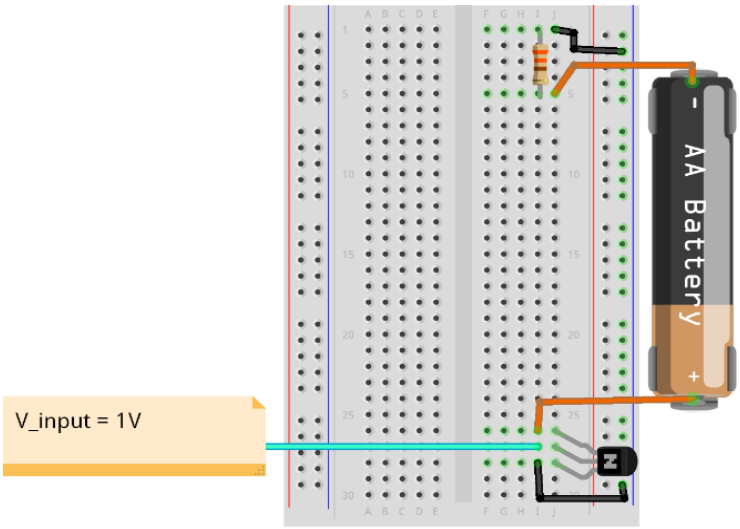

Figure 9.1.1: Setup for floating node voltage testing

Important note for the setup is to follow the setup in Figure 9.1.1 exactly. This is because if placed in the wrong terminal or order, the current won't be dissipated properly, leading to a potential circuit meltdown.

5. **Procedure(s):**
    A. Setup the experimental setup shown in Figure 9.1.1.
    B. Connect a function generator as the input (Figure 9.1.1 light blue wire)
    C. Connect the test battery in series with the circuit (Figure 9.1.1 bottom orange wire)
    D. In increments of 0.1 volts, input various DC voltages from zero to three volts.
    E. For each increment, record the voltage at the collector (Figure 9.1.1 bottom orange wire)

6. **Expected Results**: Running a quick initial test of the output indicates the possibility of a floating node voltage.

9.1.2 Testing the Injected Current Circuit's Output

1. **Objective**: Test whether the output of the injected current through the battery is able to maintain a sinusoidal waveform through the test battery.
2. **Function tested**: Current injection module.
3. **Design objective involved**: Injecting a sinusoidal current into the test battery.
4. **Experiment Setup**: Similar to experiment 9.1.1, the setup of the test circuit is extremely important. The model of the transistor is the 2N222 BJT.

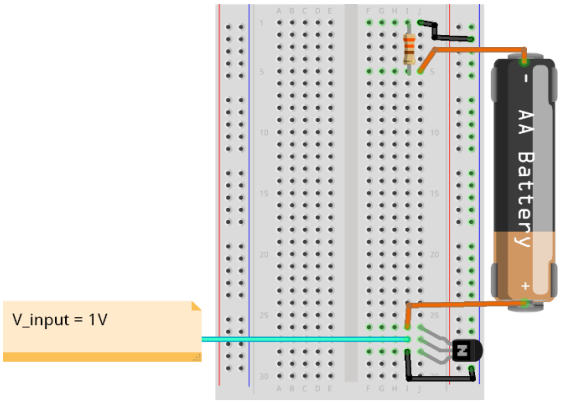

Figure 9.1.2: Setup for testing output current waveform

Resistor values are arbitrary, depending on the personal preference of a higher or lower current output. The only important component of the resistor is the location, as shown in Figure 9.1.2.

5. **Procedure(s)**:
   A. Setup the experimental setup in Figure 9.1.2.
   B. Connect a function generator as the input (Figure 9.1.2 light blue wire)
   C. Connect a test resistor (arbitrary value) in series with the circuit
   D. Connect a voltage supply to the end of the test resistor.
   E. With the function generator, input a sinusoidal voltage with an offset of two volts and a swing of two volts (frequency is arbitrary).
   F. Record the waveform shape across the test resistor using an oscilloscope.
   G. Disconnect all voltage supplies.
   H. Replace the DC voltage supply with a ground terminal.
   I. Replace the test resistor with a test battery.
   J. Place the sinusoidal input into the test circuit and record the waveform across the test battery.

6. **Expected Results**: When looking at simulations, there was no issue with maintaining a sinusoidal current through the battery, so similar results should be expected.

9.1.3 Testing the Output of the Function Generator Circuit

1. **Objective**: Test whether the function generator is able to generate the entire frequency range.
2. **Function tested**: Generate input voltage of varying frequencies.
3. **Design objective involved**: Injecting a sinusoidal current of varying frequencies into the test battery.
4. **Experiment Setup**: Test setup for this experiment is primarily based on the XR-2206 datasheet. Refer to the datasheet for which resistor and capacitor values to use based on the desired frequency.

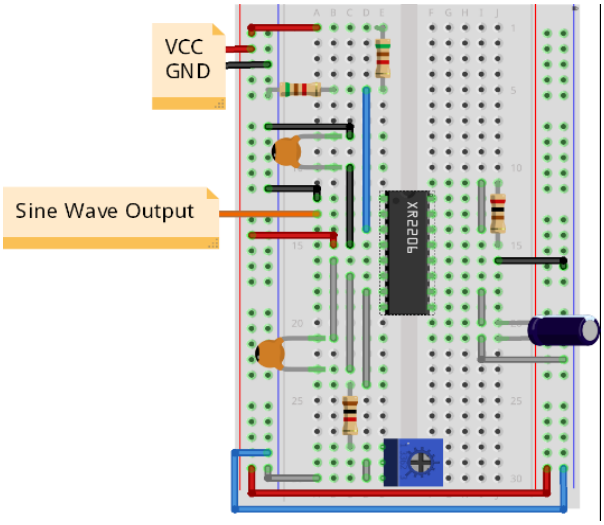

Figure 9.1.3: Test setup for the function generation circuit

The chosen amplitude of the output waveform is one volt. Looking at Figure 9.1.3, the potentiometer is shown in the setup is an analog potentiometer with a range of [10 Ω, 1 MΩ].

5. **Procedure(s)**:
   A. Setup the experimental circuit shown in Figure 9.1.3.
   B. Provide the appropriate power supplies (+12 V for the function generation chip, + 5 V for the biasing module)
   C. Connect an oscilloscope to the output of the biasing module.
   D. Slowly rotate the analog potentiometer, and note the shape of the waveform as well as the frequency value.
   E. Test for all frequencies within the objective range.

6. **Expected Results**: The entire frequency range could be generated without issues since it is well within the limits of the function generator chip.

9.1.4 Testing the Accuracy of the Differential Probes

1. **Objective**: Test the accuracy of the differential probes and see if it is good enough for the project requirements.
2. **Function tested**: Raw data extraction of the induced voltage response across the battery.
3. **Design objective involved**: Extract the induced voltage response from the current being actively injected into the battery.
4. **Experiment Setup**: The test setup for this the LF353N op-amp in differential amplifier configuration. All the resistors are the same resistor value (suggested: 10 kΩ) to provide a 1X gain.

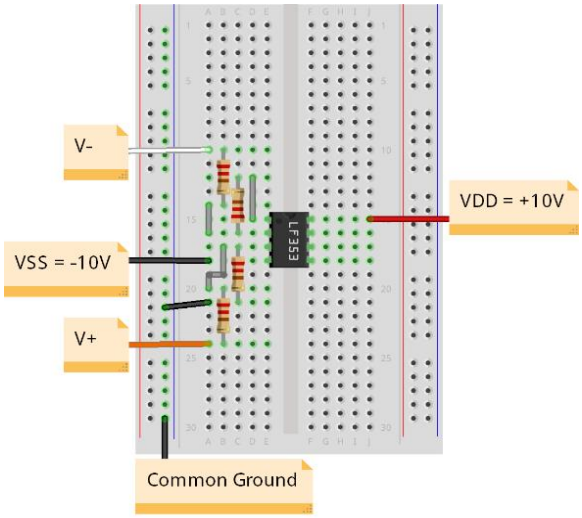

Figure 9.1.4: Test setup for the differential probes

5. **Procedure(s):**
   A. Setup the differential probe circuit shown in figure 9.1.4
   B. Feed two DC voltages with a small difference (~0.01 V) between the two of them.
   C. Record the output voltage of the differential amplifier.
   D. Repeat for different voltage values as much as needed.

6. **Expected Results**: The probes are expected to work and give the voltage difference between V+ and V- as previous LTspice simulations have shown.

9.1.5 Testing the Data Extraction Feature

1. **Objective**: Test the ability of the data extraction feature (the ADC) and the ability to transfer the data between the collection (microcontroller) and processing unit (computer).
2. **Function tested**: Data collection module.
3. **Design objective involved**: Collect the raw data and format the data to be used for processing.
4. **Experiment Setup**: The setup up is the Atmega1284P in the basic configuration to turn on. The serial communication lines are then connected to be used for USART to a computer and the ADC is also connected with the output from the differential amplifier.

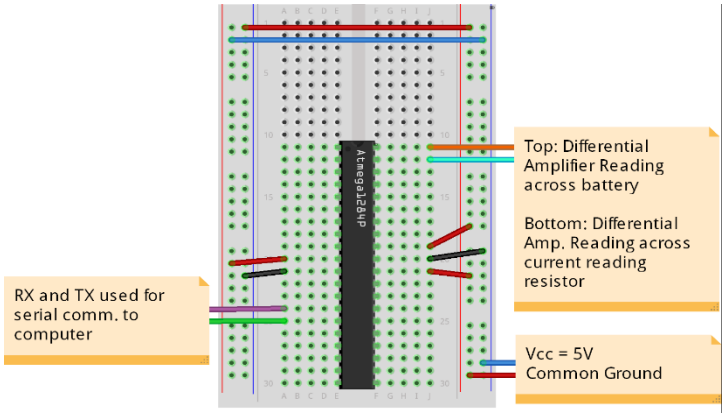

Figure 9.1.5: Test set up for data transmission between microcontroller and computer

5. **Procedure(s)**:
    A. Set the frequency of input voltage
    B. Inject battery with the produced current
    C. Read induced voltage across the battery and current reading resistor via ADC
    D. Send results from ADC to the terminal via USART serial communication
    E. Format the data received into equivalent voltage values
    F. Manually verify that results are in the correct range

6. **Expected Results**: Converted voltage value equivalents that vary around the nominal battery voltage.

9.1.6 Testing the Output of the Biasing Circuit

1. **Objective**: Test the output of the biasing circuit by changing the bias of the function generator circuit's output
2. **Function tested**: Biasing of the input voltage.
3. **Design objective involved**: Generating a sufficient input voltage for current injection.
4. **Experiment Setup**: The setup for this test consists of three resistors and a capacitor to produce a DC offset that could be added to an input sinusoid.

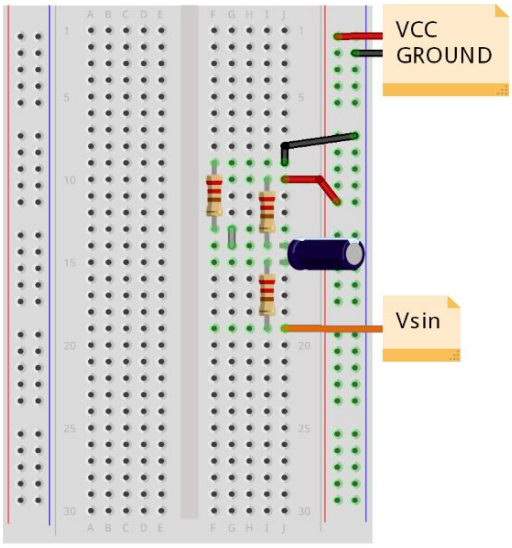

Figure 9.1.6: Testing circuit for the biasing circuit

5. **Procedure(s)**:
   A. Setup the biasing circuit shown in figure 9.1.6.
   B. Place two oscilloscope probes: One at the output of the function generator output and one at the biasing output.
   C. Start at the lowest frequency (1 Hz) and input the necessary power supplies.
   D. Record the output of the biasing circuit.
   E. Adjust the voltage divider as necessary after disconnecting the output of the function generator output after each acquisition.
   F. Repeat steps C through E until desired bias is reached.
   G. Adjust frequency to the highest in the range (1 kHz) and repeat steps C through F.

6. **Expected Results**: The input sinusoid is DC offset in the positive direction.

9.2 Bug Tracking

A database, shown in table 9.2, is used to track defects found while performing the test cases. All defects are logged as they are discovered. Defects are then assigned to individual members to investigate.

Test	Defects
9.1.1	N/A
9.1.2	Circuit heats up if left on too long, suggesting power issue
9.1.3	Function generator heats up if frequency switches too fast
9.1.4	N/A
9.1.5	Was not getting the full 10-bit ADC value and the ADC was not in free running mode
9.1.6	Slewing of slopes of output as well as inconsistency in startup

Table 9.2 Database for bug tracking

9.3 Quality Control

The test described in section 9.1 either pass or fail the designed specifications. The results of these tests are listed in table 9.3. This table had the date of the test and any seen deviations from the expected results.

Test (Iteration)	Pass/Fail (Fix Time)	Deviations
9.1.1 (1)	Fail (Tues. 10/16/18)	Node voltage is inconsistent and isn't following any static value or pattern
9.1.1 (2)	Pass (Thurs. 10/18/18)	N/A
9.1.1 (3)	Pass (Fri. 10/19/18)	N/A
9.1.2 (1)	Pass (Fri. 11/23/18)	N/A
9.1.3 (1)	Fail (2/2/19)	Slewing of the slopes of sinusoids, especially at lower frequencies (~100 Hz)
9.1.3 (2)	Pass (2/2/19)	Slight slew at really low frequency (~ 1 Hz), but still sinusoidal enough for data collection.
9.1.4 (1)	Pass (2/10/19)	A slight deviation in gain (~0.95)
9.1.5 (1)	Pass (1/30/19)	Data was gathered but Arduino is not the final implementation
9.1.5 (2)	Pass (2/10/19)	Using the Atmega, data was not massively sinusoidal but it was being captured and processed
9.1.5 (3)	Pass (2/10/19)	N/A
9.1.6 (1)	Fail (Tues. 1/29/19)	Output swing shrinks with frequency, also inconsistent.
9.1.6 (2)	Pass (Thurs. 1/31/19)	Still slight swing decrease with frequency, but at max 0.1 V so deemed acceptable

Table 9.3 Table for test results

9.4 Identification of critical components

When testing the system, there are several components that need to be focused on to ensure that the battery testing process is accurate and safe for use:

- Sufficient current (>1 mA) is flowing through the test battery in the current injection circuit module. For reference, the current values when properly injected should fall in the range of [9 mA, 15 mA]. Conversely, the current should not exceed 20 mA or else safety issues with the overall current injection circuit arises.

- Accuracy in the gain of the differential probes. The discrepancy between the claimed accuracy and actual accuracy will skew data due to the change in the differential op-amps gain.

- The proper supply voltage is given to each module of the project. This is especially important for the biasing circuit as the supply voltage plays a large role in determining the bias of the circuit. Improper voltage levels will not allow sections of the system to operate properly.

- The input voltage to the current injection circuit should be sinusoidal and never cross the zero thresholds. If this is not maintained the current will become non-sinusoidal.

9.5 Items Not Tested by the Experiments

- Overall test times for a frequency sweep. Due to issues incorporating the digital potentiometers during testing, testing with the digital potentiometer is impossible at the moment. Recording testing times with an analog potentiometer is inaccurate compared to an automated test.

- The needed discharge times. Due to risks with testing, minimal testing time to switch frequencies is not advised for safety reason. Generous time is given between testing periods, preventing safety issues but increasing the testing time.

- Accuracy, setting time, and effectiveness of the digital potentiometer.

10 Test Report

10.1 Floating Node Voltage

Iteration one results: The results of the experiments suggests the probability of a floating node voltage. A floating node voltage is a persistent voltage at a specified node that could potentially interfere with measurements and overall results.

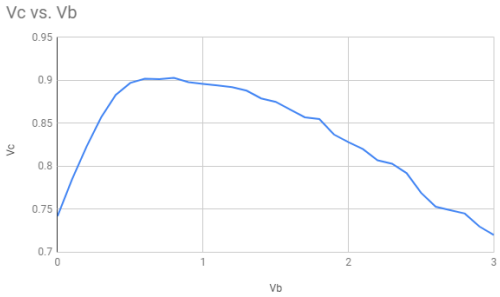

Figure 10.1.1: Table of the node voltage at the collector (Vc) vs. the input voltage (VB)

Figure 10.1.1 shown above shows the change in the node voltage value with different input voltage values. Inability to explain the cause of these results or patterns in the voltage values highly encourages a second iteration for verification.

Iteration two results: The results for the second attempt of the experiment yielded different results. The results seem to suggest that there is no floating node voltages at the collector, suggesting that the results of the first experiment are due to faulty equipment and human error.

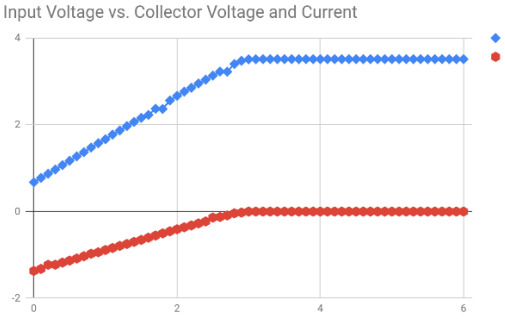

Figure 10.1.2: Input voltage vs. Collector voltage (Blue) and current (Red)

Figure 10.1.2 show both the current and voltage values at the collector. The current is inherently negative due to the properties of current mirrors. Collector voltages change proportionally with collector current, suggesting that there are no floating node voltages. To verify the validity of these results, a third test will be performed.

Iteration three results: The results are identical to the second iteration, suggesting the first iteration to be wrong and should be ignored.

10.2 Testing the Injected Current Circuit's Output

Iteration one results: When looking at the expected results, the current through the battery should retain the sinusoidal shape although some slight distortion might occur.

Figure 10.2.1: Oscilloscope output for the current (Top) and input voltage (Bottom).

The results shown in the oscilloscope in Figure 10.2.1 verify the expected results. The current through the battery maintains the sinusoidal waveform with slight distortion. A second iteration is needed for verification.

Iteration two results: The results are identical to the first iteration, confirming the idea that current injection through a battery maintains a sinusoidal shape.

10.3 Testing the Output of the Function Generator Circuit

Iteration one results: Assuming correct setup of the function generator chip, the square and sinusoidal waveforms should be properly generated. There should be no distortion as the frequency range used in the final implementation (f ∈ [10 Hz, 1 KHz]) is well within the limits of the function generator chip.

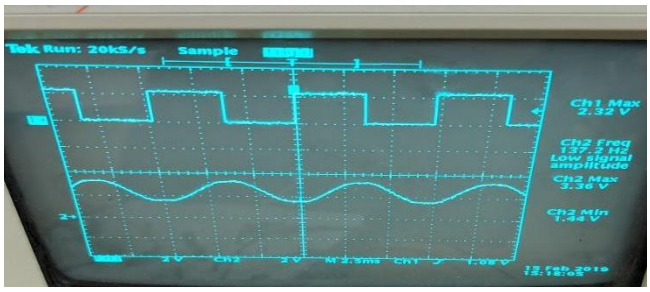

Figure 10.3.1: Oscilloscope outputs for the sinusoidal and waveform nodes of the function generator chip

Both outputs in Figure 10.3.1 are shown to be clean and lacking in noise. The amplitude and frequency values are also similar to the values depicted in the manual. A second iteration will be performed after adjusting the parts just for verification.

Iteration two results: The results are identical to the first iteration. No issues were expected for both iterations as circuit assembly followed the function generator chip's manual to an exact degree.

10.4 Testing the Accuracy of the Differential Probes

Iteration one results: To test the accuracy differential probes, two input voltages were given: ten volts and nine volts. The expected output should be one volt, but instead was found to be 0.9 V. The differential probe is for the most part accurate, with deviations due to deviations in the resistor values in the circuit. A second iteration with different resistors will be used for verification.

Iteration two results: Identical results suggest that the deviations in the probes stem from low precision resistors. It is suggested that for the final implementation that incredibly high precision resistors are used for maximum accuracy.

10.5 Testing the Data Extraction Feature

Iteration one results: The first data extraction test is conducted using the Arduino Uno/Mega 2560. The results create data that appears sinusoidal, similar to what was expected. The values itself are similar in magnitude to the nominal battery voltage (DC value). Further data extraction tests with different microcontrollers are suggested to optimize the data extraction process.

Iteration two results: The second data extraction test is conducted using ATMEGA 1284P. The recorded values are flat and static, with the magnitude incredibly similar to the nominal voltage of the battery. The issue is believed to be from improper optimization of the ADC, meaning a second test is needed.

Iteration three results: The ATMEGA1284P is used for another test, this time with a properly optimized ADC. The data extracted this time is inherently sinusoidal with slight deviation, as expected.

10.6 Testing the Output of the Biasing Circuit

Iteration one results: The biased circuit outputs voltage values that are different from what was expected. It's believed the discrepancy occurred due the capacitor blocking the original bias while also reducing our output swing. Using a capacitor of 200 µF, the circuit successfully biased the signals at all frequencies, but the output amplitude is reduced by half at high frequencies. A second test with different capacitors and different capacitance values is recommended.

Iteration two results: Switching the capacitance to 10 µF fixes the amplitude issues at high frequencies. However, the issue now occurs at low frequencies (~10 Hz). Switching the capacitor to an average value in between 10 and 200 µF is recommended.

Iteration three results: Switching the capacitance to 100 µF balances the output at all frequencies. The tradeoff is that there is a consistent 0.1 V decrease in the amplitude. However, the circuit successfully biases the function generation chip output to be appropriate for the current injection module.

11 Conclusion and Future Work

11.1 Conclusion

After thorough design and testing of the device, it achieves most of the design specifications initially set. The device is able to generate, capture, and analyze a voltage response from the test battery when injected with a low sinusoidal current. Although system accuracy is unable to be fully verified, the trends from multiple tests follow the data patterns presented in the experiment on which the device is based on [3].

Objective	Result	Notes
Ic∈ [1 mA,20 mA]	Implemented	N/A
f∈ [1 Hz,1 KHz]	Implemented	A full frequency sweep was not possible for the implementation so frequency stepping of about 20 Hz was implemented
Flexibility in testing circuit	Implemented	Subsystems work completely independently so future modification is possible
Cross platform data movement	Partially Implemented	Code and development environments ran into difficulties when transitioning between Windows and Linux Operating systems.
Accurate Data Collection	Implemented	N/A
Overall accuracy of 10%	Implemented	Can be further improved to a higher accuracy range
Low Cost	Implemented	Final implementation is well within cost objective
Safe Power dissipation	Implemented	Basic level implemented can be improved
Short testing time	Implemented	Can be further improved on but still much less than conventional methods
Easy to follow Circuitry	Implemented	PCB size not fully minimized but currently smaller than conventional systems
Process data	Partially Implementing	Data is processed but the results have not been checked against conventional system data

Table 11.1 Implementation Status of Design Objectives

For the partially implemented objectives a further explanation is found below:

- Cross platform data movement was hindered by Pythons interaction with the Windows operating system and Microsoft Visual Studio. When in a Linux environment the Python script operated properly and formatted the data for MATLAB's use. Many hours of debugging went into trying to get the conflict between windows and python to work but could not be fixed.

- For the MATLAB processing it works properly. The only thing that has not been completely finished is an overall checking validation of the calculated data to data gathered from a conventional system.

Throughout this project a lot of team and individual learning was required in order to be able to implement a working system. Individual members learned:

- Jack Gatfield:
    - Team communication
    - Proper documentation of the design process
    - Communication between multiple levels of management
    - Methodical methods for hardware and software debugging, testing, and documentation
    - Understanding engineering optimization, trade-offs, and methodologies
    - Organization of circuits, documentation, and software functions
    - Understanding of working under deadlines.

- Jack Gu:
    - Team communication
    - Communication between multiple levels of management
    - Being able to communicate ideas to others both textually and verbally
    - Methodical method for hardware debugging, testing, and documentation
    - Research in developing and testing components of a project
    - Understanding of engineering optimization and tradeoffs
    - Understanding of working under deadlines.

- Joseph Gozum:
    - Team communication
    - Proper documentation of the design process
    - Understanding engineering optimization, trade-offs, and methodologies
    - Understanding of working under deadlines.
    - Methodical methods for software debugging, testing, and documentation.
    - Interfacing with multiples devices using different communication standards
    - Being able to communicate with different levels of management

11.2 Future Work

The IBMS improves efficiency and viability of lithium-ion cells through frequency-based testing methods to catch poor health indicators. The current iteration of the design works, there are several key areas that can be improved upon to improve the overall system.

- Overall size of the device. It can sit nicely on a desktop as it is now but can be further minimized by designing a PCB utilizing SMT components.

- The PCB can also be further improved by separating the power generation, signal generation, and data acquisition sections. This helps to further reduce noise. Also having a dedicated ground for power, analog devices, and digital devices further reduces noise.

- Precision could be greatly improved by incorporating a higher bit SAR ADC or even using an ADC based on the Sigma-Delta method. While the Sigma-Delta method is slower than a SAR ADC, it provides higher resolution, and the conversion speeds are satisfactory for the data collection process. The fastest frequency tested is 1 kHz this requires a sampling frequency at least twice that much which a Sigma-Delta ADC can satisfy. Another important benefit is that Sigma-Delta ADCs provide the best noise isolation which is important with the small signals of the system.

- Pushing the current through the battery to its more upper limits may also prove to be useful in determining the voltage response required in the impedance calculation, injecting more current may yield more noticeable results that can be more clearly captured but must be balanced with the safety risks.

- Utilizing a microcontroller with DMA to its registers may prove helpful in the quality of the data capture as it reduces the time between each ADC reading. Currently with the SAR ADC built into the Atmega1284P, it takes a certain amount of time between each reading but a DMA could set it so that it takes care of transferring the ADC data while CPU has more time to focus on the actual acquisition.

- The quality of input impedance calculation can also be improved by utilizing higher-tolerance and more wiper position digital potentiometers. Currently, every whole integer frequency within the specified range cannot be produced because of the limitations of the resistance selection with the digital potentiometer. Analog potentiometers also have the inability to set exact resistances which is an issue impossible to ignore.

- The biggest improvement on this prototype is to implement the ability to test multiple battery cells at once; this could be accomplished through a multiplex selecting different batteries.

- Utilize LabVIEW to create a signal GUI to interface with the device and processing. (Joseph Gozum) *Note: Very expensive option unless large additional funding

Students of future classes should definitely look into improving this design. The time spent on this project has been fulfilling and great learning experience that really pushed this group to the limits of the knowledge learned while attending the University of California, Riverside.

The future market for this device is large. As the electric vehicle industry increases and a push for more renewable resources is implemented lithium ion batteries will become more used. This increased usage needs a better battery management system such as the one designed in this project to improve longevity and efficiency of the battery banks.

11.3 Acknowledgement

- Professor Mihri Ozkan and Cengiz Ozkan and Yige Li
    - Provided the core project idea
    - Helped with understanding test data
- Professor Roman Chomko
    - Provided design concepts and feedback
- Impedance-Based Battery Management System for Safety Monitoring of Lithium-Ion Batteries
    - By Bliss G. Carkhuff, Plamen A. Demirev, and Rengaswamy Srinivasan
        - Gave background information
- Method and device for determining the impedance of an energy storage element of a battery
    - By: Marco Ranieri and Vincent Heiries
        - Helped develop implementation process
- Active Battery Cell Balancing
    - By: Kevin Scott and Sam Nork
        - Helped develop implementation process

12 References

[1] Atmel Corporation, “8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash”, ATMEGA1284P Datasheet, Nov. 2009

[2] Analog Devices, “I2C-Compatible, 256-Position Digital Potentiometer”, AD5241/AD5242 Datasheet, May 2014

[3] Bliss G. Carkhuff, Plamen A. Demirev, and Rengaswamy Srinivasan, “Impedance-Based Battery Management System for Safety Monitoring of Lithium-Ion Batteries”, *IEEE Trans, 2017*

[4] Exar Corporation, “XR-2206 Monolithic Function Generator”, XR-2206 Datasheet, Feb. 2008

[5] Maxim Integrated, “Switched-Capacitor Voltage Convertors”, MAX1044 Convertor Datasheet, Nov. 2017

13 Appendices

13.1 Appendix A: Parts List

Parts in final circuit design		
Needed Part	**QTY.**	**Used Part**
12V DC Power Adapter	1	12V DC Power Adapter
Barrel Jack	1	Barrel Jack
0.33uF non polarized capacitor	2	0.33uF non polarized capacitor
0.1uF non polarized capacitor	4	0.1uF non polarized capacitor
5V voltage regulator	1	L7805C
10V voltage regulator	1	BA17810
Voltage inverter	1	Max1044CPA+
Potentiometer (Digital for PCB, analog for through hole board)	1	AD5242
Function generating chip	1	XR2206
2 level dip switches	2	2 level dip switches
NPN BJT	1	2n2222a
18650 battery holders	1	Surface 18650 battery holders
18650 battery	Test batch	4 test batteries
Op amp	1	LF353
Microcontroller	1	Atmega1284p
Programmer and cable	1	Atmega 1284p variations
Program header	1	IEEE UCR programmer header
Serial communication cable	1	FTDI Serial TTL-232 USB Cable
Resistors (1Ω to 20kΩ)	25	1Ω, 15Ω, 100 Ω, 220 Ω, 300Ω, 1kΩ, 4.7kΩ, 5.1kΩ, 10kΩ, 20kΩ

13.2 Appendix B: Equipment List

Equipment	Use
Oscilloscope	Check signals throughout circuits
Desktop power supply	Power circuit
Multimeter	Check different aspects of the circuit
Computers	Interface with the microcontroller, process the data, and allow for document making
Arduinos	Development and testing systems
Solder station, board holder, and suction pen	Solder the through hole and PCB together
Breadboards	Foundation of prototypes and testing
Various resistors, capacitors, and ICs	Used for development and testing

13.3 Appendix C: Software List

Software	Use
MATLAB	Data Processing
Python Command Shell	Data Formatting
PuTTY	Serial Terminal for data capture and serial communication to microcontroller
RealTerm	Serial Terminal for data capture and serial communication to microcontroller
Atmel studio	Programming the microcontroller
Eagle PCB	PCB designing and file generation for printing
AutoCAD	Designing of housing for PCB
LTspice	Circuit simulation and design
Arduino IDE	Programming Arduino for testing throughout development
Arduino Serial terminal	Terminal communication throughout development
Microsoft Office Suite	Documentation, presentation, and report generation and formatting. Excel was also used for data storage.
Google Doc Suite	Documentation, presentation, and report generation and formatting
Google Hangouts	Team meeting when not in person

13.4 Appendix E: User's Manual

DIGITAL TEST:

1. Connect the power supply into a wall outlet
2. Connect the serial cable to the processing laptop
3. Select type of test.
    a) Frequency Sweep
    b) Singular frequency
4. Connect battery into a battery holder
5. Enable test
6. Wait for processing to occur
7. Acquire results and repeat if needed.
8.

NOTE:

- When testing with singular frequencies, please allow for a minimum time of a minute to pass before attempting a test at a different frequency. This is to ensure safety as well as to prevent transient responses that could potentially damage the device.
-

ANALOG TEST:

1. Connect the appropriate power supply to each individual component of the circuit.
    a) For the amplifiers, feed a +10V/-10V into the LF353's power terminals.
    b) For the Function generation module, feed a 12V power supply into the XR2206's power terminals.
    c) For the biasing module, feed a 5V power supply into the power terminal of the biasing module.
2. Using an oscilloscope for reference, adjust the potentiometer until the desired frequency is reached. Rotate the potentiometer clockwise to increase input frequency and vice versa.
3. Once satisfied, connect the biasing module output to the current injection circuit.
4. Place the battery into the battery holder.
5. Measure results
6. Disconnect the battery first then the output of the biasing module.
7. Repeat steps 2-6 as needed.
8.

NOTE:

- When doing multiple tests, please allow for a time delay of thirty seconds before reconnecting new frequency inputs into the current injection circuit. This is done due to safety concerns.
- If doing a step and trying to minimize test times, keep to low step sizes (<10 Hz).
- If an emergency occurs and there are unexpected shorts or other issues, disconnect the battery immediately to cause the emergency shutdown.

13.5 Appendix F: Frequency Lookup Table (DIGITAL)

Desired Frequency	Terminal Value (HEX)	Resistance (Ω)	Capacitance (μF)	Digi pot value (HEX)	Period (s)	Period (minutes)	Collection rate (Period/256)	Delay (ms)
1 Hz	23	10K	100	2	1	0.01666666667	0.00390625	0.004
4 Hz	24	2.5K	100	0	0.25	0.004166666667	0.0009765625	0.001
7 Hz	25	1.42K	0.1	0	0.1428571429	0.002380952381	0.0005580357143	0
10 Hz	26	1M	0.1	FF	0.1	0.001666666667	0.000390625	0
13 Hz	27	769.23K	0.1	C4	0.07692307692	0.001282051282	0.0003004807692	0
16 Hz	28	625K	0.1	9F	0.0625	0.001041666667	0.000244140625	0
19 Hz	29	526.32K	0.1	86	0.05263157895	0.0008771929825	0.0002055921053	0
22 Hz	2A	454.55K	0.1	74	0.04545454545	0.0007575757576	0.0001775568182	0
25 Hz	2B	400K	0.1	66	0.04	0.0006666666667	0.00015625	0
28 Hz	2C	357.14K	0.1	5B	0.03571428571	0.0005952380952	0.0001395089286	0
31 Hz	2D	322.58K	0.1	52	0.03225806452	0.0005376344086	0.0001260080645	0
34 Hz	2E	294.12K	0.1	4B	0.02941176471	0.0004901960784	0.0001148897059	0
37 Hz	2F	270.27K	0.1	45	0.02702702703	0.0004504504505	0.0001055743243	0
40 Hz	30	250K	0.1	3F	0.025	0.0004166666667	0.00009765625	0
43 Hz	31	232.56K	0.1	3B	0.02325581395	0.0003875968992	0.00009084302326	0
46 Hz	32	217.39K	0.1	37	0.02173913043	0.0003623188406	0.00008491847826	0
49 Hz	33	204.09K	0.1	34	0.02040816327	0.0003401360544	0.00007971938776	0
52 Hz	34	192.31K	0.1	31	0.01923076923	0.0003205128205	0.00007512019231	0
55 Hz	35	181.18K	0.1	2E	0.01818181818	0.000303030303	0.00007102272727	0
58 Hz	36	172.41K	0.1	2C	0.01724137931	0.0002873563218	0.00006734913793	0
60 Hz	37	166.67K	0.1	2A	0.01666666667	0.0002777777778	0.00006510416667	0

Desired Frequency	Terminal Value (HEX)	Resistance (Ω)	Capacitance (μF)	Digi pot value (HEX)	Period (s)	Period (minutes)	Collection rate (Period/256)	Delay (ms)
61 Hz	38	163.93K	0.1	29	0.01639344262	0.0002732240437	0.00006403688525	0
64 Hz	39	156.25K	0.1	27	0.015625	0.0002604166667	0.00006103515625	0
67 Hz	3A	149.25K	0.1	26	0.01492537313	0.0002487562189	0.00005830223881	0
70 Hz	3B	142.86K	0.1	24	0.01428571429	0.0002380952381	0.00005580357143	0
73 Hz	3C	136.99K	0.1	23	0.01369863014	0.0002283105023	0.00005351027397	0
76 Hz	3D	131.58K	0.1	3	0.01315789474	0.0002192982456	0.00005139802632	0
79 Hz	3E	126.58K	0.1	20	0.01265822785	0.0002109704641	0.00004944620253	0
82 Hz	3F	121.95K	0.1	1F	0.01219512195	0.0002032520325	0.00004763719512	0
85 Hz	40	117.65K	0.1	1E	0.01176470588	0.0001960784314	0.00004595588235	0
88 Hz	41	113.64K	0.1	1D	0.01136363636	0.0001893939394	0.00004438920455	0
91 Hz	42	109.89K	0.1	1C	0.01098901099	0.0001831501832	0.00004292582418	0
94 Hz	43	106.38K	0.1	1B	0.01063829787	0.0001773049645	0.00004155585106	0
97 Hz	44	103.09K	0.1	1A	0.01030927835	0.0001718213058	0.00004027061856	0
100 Hz	45	100K	0.1	19	0.01	0.0001666666667	0.0000390625	0
120 Hz	46	83.33K	0.1	15	0.008333333333	0.0001388888889	0.00003255208333	0
130 Hz	47	76.92K	0.1	13	0.007692307692	0.0001282051282	0.00003004807692	0
160 Hz	48	62.5K	0.1	0F	0.00625	0.0001041666667	0.0000244140625	0
190 Hz	49	52.63K	0.1	0D	0.005263157895	0.00008771929825	0.00002055921053	0
220 Hz	4A	45.45K	0.1	0B	0.004545454545	0.00007575757576	0.00001775568182	0
250 Hz	4B	40K	0.1	0A	0.004	0.00006666666667	0.000015625	0
280 Hz	4C	35.71K	0.1	9	0.003571428571	0.00005952380952	0.00001395089286	0
310 Hz	4D	32.26K	0.1	8	0.003225806452	0.00005376344086	0.00001260080645	0
340 Hz	4E	29.41K	0.1	7	0.002941176471	0.00004901960784	0.00001148897059	0

Desired Frequency	Terminal Value (HEX)	Resistance (Ω)	Capacitance (µF)	Digi pot value (HEX)	Period (s)	Period (minutes)	Collection rate (Period/256)	Delay (ms)
370 Hz	4F	27.03K	0.1	6	0.002702702703	0.00004504504505	0.00001055743243	0
400 Hz	50	25K	0.1	6	0.0025	0.00004166666667	0.000009765625	0
430 Hz	51	23.26K	0.1	5	0.002325581395	0.00003875968992	0.000009084302326	0
460 Hz	52	21.74K	0.1	5	0.002173913043	0.00003623188406	0.000008491847826	0
490 Hz	53	20.41K	0.1	5	0.002040816327	0.00003401360544	0.000007971938776	0
520 Hz	54	19.23K	0.1	4	0.001923076923	0.00003205128205	0.000007512019231	0
550 Hz	55	18.18K	0.1	4	0.001818181818	0.0000303030303	0.000007102272727	0
580 Hz	56	17.24K	0.1	4	0.001724137931	0.00002873563218	0.000006734913793	0
610 Hz	57	16.39K	0.1	4	0.001639344262	0.00002732240437	0.000006403688525	0
640 Hz	58	15.63K	0.1	4	0.0015625	0.00002604166667	0.000006103515625	0
670 Hz	59	14.93K	0.1	3	0.001492537313	0.00002487562189	0.000005830223881	0
700 Hz	5A	14.29K	0.1	3	0.001428571429	0.00002380952381	0.000005580357143	0
730 Hz	5B	13.69K	0.1	3	0.001369863014	0.00002283105023	0.000005351027397	0
760 Hz	5C	13.16K	0.1	3	0.001315789474	0.00002192982456	0.000005139802632	0
790 Hz	5D	12.66K	0.1	3	0.001265822785	0.00002109704641	0.000004944620253	0
820 Hz	5E	12.2K	0.1	3	0.001219512195	0.00002032520325	0.000004763719512	0
850 Hz	5F	11.76K	0.1	3	0.001176470588	0.00001960784314	0.000004595588235	0
880 Hz	60	11.36K	0.1	2	0.001136363636	0.00001893939394	0.000004438920455	0
910 Hz	61	10.99K	0.1	2	0.001098901099	0.00001831501832	0.000004292582418	0
940 Hz	62	10.64K	0.1	2	0.001063829787	0.00001773049645	0.000004155585106	0
970 Hz	63	10.31K	0.1	2	0.001030927835	0.00001718213058	0.000004027061856	0
1 KHz	64	10K	0.1	2	0.001	0.00001666666667	0.00000390625	0

13.6 Appendix G: Frequency Lookup Table (ANALOG)

Desired Frequency	Resistance	Capacitance
1 Hz	10K	100µF
4 Hz	2.5K	100µ F
7 Hz	1.42K	0.1µ F
10 Hz	1M	0.1µ F
13 Hz	769.23K	0.1µ F
16 Hz	625K	0.1µ F
19 Hz	526.32K	0.1µ F
22 Hz	454.55K	0.1µ F
25 Hz	400K	0.1µ F
28 Hz	357.14K	0.1µ F
31 Hz	322.58K	0.1µ F
34 Hz	294.12K	0.1µ F
37 Hz	270.27K	0.1µ F
40 Hz	250K	0.1µ F
43 Hz	232.56K	0.1µ F
46 Hz	217.39K	0.1µ F
49 Hz	204.09K	0.1µ F
52 Hz	192.31K	0.1µ F
55 Hz	181.18K	0.1µ F
58 Hz	172.41K	0.1µ F
60 Hz(*)	166.67K	0.1µ F
61 Hz	163.93K	0.1µ F
64 Hz	156.25K	0.1µ F
67 Hz	149.25K	0.1µ F
70 Hz	142.86K	0.1µ F
73 Hz	136.99K	0.1µ F
76 Hz	131.58K	0.1µ F
79 Hz	126.58K	0.1µ F
82 Hz	121.95K	0.1µ F
85 Hz	117.65K	0.1µ F
88 Hz	113.64K	0.1µ F
91 Hz	109.89K	0.1µ F
94 Hz	106.38K	0.1µ F
97 Hz	103.09K	0.1µ F
100 Hz	100K	0.1µ F
120 Hz (*)	83.33K	0.1µ F
130 Hz	76.92K	0.1µ F

Desired Frequency	Resistance	Capacitance
160 Hz	62.5K	0.1µ F
190 Hz	52.63K	0.1µ F
220 Hz	45.45K	0.1µ F
250 Hz	40K	0.1µ F
280 Hz	35.71K	0.1µ F
310 Hz	32.26K	0.1µ F
340 Hz	29.41K	0.1µ F
370 Hz	27.03K	0.1µ F
400 Hz	25K	0.1µ F
430 Hz	23.26K	0.1µ F
460 Hz	21.74K	0.1µ F
490 Hz	20.41K	0.1µ F
520 Hz	19.23K	0.1µ F
550 Hz	18.18K	0.1µ F
580 Hz	17.24K	0.1µ F
610 Hz	16.39K	0.1µ F
640 Hz	15.63K	0.1µ F
670 Hz	14.93K	0.1µ F
700 Hz	14.29K	0.1µ F
730 Hz	13.69K	0.1µ F
760 Hz	13.16K	0.1µ F
790 Hz	12.66K	0.1µ F
820 Hz	12.2K	0.1µ F
850 Hz	11.76K	0.1µ F
880 Hz	11.36K	0.1µ F
910 Hz	10.99K	0.1µ F
940 Hz	10.64K	0.1µ F
970 Hz	10.31K	0.1µ F
1 KHz	10K	0.1µ F

13.7 Appendix H: MATLAB Code/Data formatting code

Frequency Sweep Data Processing Code

```
clear all;
close all;
clc;

%variable decloration
BatteryVoltage = 3.2;
CurrentResistor = 1;
FrequencyCount = 1;
CurrentGain     = 9.2;

PhaseRunner = 0:(8*pi)/255:8*pi;
t = 1:256;

FrequencyRange = [1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49, 52, 55, 58, 60, 61, 64,
67, 70, 73, 76, 79, 82, 85, 88, 91, 94, 97, 100, 120, 130, 160, 190, 220, 250, 280, 310, 340, 370, 400,
430, 460, 490, 520, 550, 580, 610, 640, 670, 700, 730, 760, 790, 820, 850, 880, 910, 940, 970, 1000];
[FrequencyRows,FrequencyColumn] = size(FrequencyRange);

BatteryImpedence = zeros(1,256);

AvgVoltagePull = zeros(1,FrequencyColumn);
AvgCurrentPull = zeros(1,FrequencyColumn);
AvgBatteryImpedence = zeros(1,FrequencyColumn);
AvgComplexBatteryImpedence = zeros(1,FrequencyColumn);

%loop through all test frequencys --> 66 total frequencies
for FrequencyCount =1:FrequencyColumn %FrequencySize

filename = sprintf('%s_%d','TestData',FrequencyRange(FrequencyCount));
filename = strcat(filename, ".csv");

%Pulls columns in from Excel Sheet that the serial terminal will popualte
Voltage1 = xlsread(filename,"A2:A257"); %Gets data for voltage calculations
Voltage2 = xlsread(filename,"A258:A513");
Voltage3 = xlsread(filename,"A514:A769");
Voltage4 = xlsread(filename,"A770:A1025");
Voltage5 = xlsread(filename,"A1026:A1281");

CurrentVoltage1 = xlsread(filename,"A1282:A1537"); %Gets data for current calculations
CurrentVoltage2 = xlsread(filename,"A1538:A1793");
CurrentVoltage3 = xlsread(filename,"A1794:A2049");
CurrentVoltage4 = xlsread(filename,"A2050:A2305");
CurrentVoltage5 = xlsread(filename,"A2306:A2561");
```

```
%Transposes column array to be a row array
Voltage1 = Voltage1';
Voltage2 = Voltage2';
Voltage3 = Voltage3';
Voltage4 = Voltage4';
Voltage5 = Voltage5';

CurrentVoltage1 = CurrentVoltage1';
CurrentVoltage2 = CurrentVoltage2';
CurrentVoltage3 = CurrentVoltage3';
CurrentVoltage4 = CurrentVoltage4';
CurrentVoltage5 = CurrentVoltage5';

%Subtracts the battery voltage out of the reading
Voltage1_mag = Voltage1 - BatteryVoltage;
Voltage2_mag = Voltage2 - BatteryVoltage;
Voltage3_mag = Voltage3 - BatteryVoltage;
Voltage4_mag = Voltage4 - BatteryVoltage;
Voltage5_mag = Voltage5 - BatteryVoltage;

%Divides by the test resistor value to get current
Current1 = CurrentVoltage1 / CurrentResistor;
Current2 = CurrentVoltage2 / CurrentResistor;
Current3 = CurrentVoltage3 / CurrentResistor;
Current4 = CurrentVoltage4 / CurrentResistor;
Current5 = CurrentVoltage5 / CurrentResistor;

%Divide by the current gain to get actual gain
Current1 = Current1 / CurrentGain;
Current2 = Current2 / CurrentGain;
Current3 = Current3 / CurrentGain;
Current4 = Current4 / CurrentGain;
Current5 = Current5 / CurrentGain;

%Take average of the multiple voltage data trials to isolate noise
Voltage_TotalMag = Voltage1_mag+ Voltage2_mag + Voltage3_mag + Voltage4_mag + Volt-
age5_mag;
Voltage_AvgMag = Voltage_TotalMag / 5;

%Takes aveargе of the multiple current data trials to isolate noise
Current_TotalMag = Current1 + Current2 + Current3 + Current4 + Current5;
Current_AvgMag= Current_TotalMag / 5;

%Calculate battery impedence magnituded
for i = 1:256
   if Current_AvgMag(i) ~= 0
      BatteryImpedence(i) = Voltage_AvgMag(i) ./ Current_AvgMag(i);
   end
end
```

```
%Find phase shift
TestSin = sin(FrequencyRange(FrequencyCount).*PhaseRunner);
Average_Battery_Phase    = mean(Voltage_AvgMag);
Amplitude_Battery_Phase  = (max(Voltage_AvgMag)-min(Voltage_AvgMag))/2;
Peak_Battery      = 100 * 2 * pi;
Phaseshift_Battery = 0;

myfit = NonLinearModel.fit(PhaseRunner,Voltage_AvgMag,
                    'y~b0+b1*sin(b2*x1+b3)';
                    [Average_Battery_Phase,
                    Amplitude_Battery_Phase,
                    Peak_Battery,
                    Phaseshift_Battery]);

Z_PhaseShif_radians = acos(dot(myfit.Fitted,TestSin)/(norm(TestSin)*norm(myfit.Fitted)));
Z_PhaseShif_degrees = Z_PhaseShif_radians * 360 / (2*pi);

%Polar to Rectangular
x = BatteryImpedence.*cos(Z_PhaseShif_degrees);
y = BatteryImpedence.*sin(Z_PhaseShif_degrees);
ComplexImpedence = complex(x,y);

%disp(Z_PhaseShif_radians);
AvgVoltagePull(FrequencyCount)             = mean(Voltage_AvgMag);
AvgCurrentPull(FrequencyCount)             = mean(Current_AvgMag);
AvgBatteryImpedence(FrequencyCount)        = mean(BatteryImpedence);
AvgComplexBatteryImpedence(FrequencyCount)  = mean(ComplexImpedence);

end
```

```
%Graphing code
figure(1)
subplot(4, 1, 1)
   hold on
      stem(FrequencyRange,AvgVoltagePull)
      title("Average Voltage")
         xlabel("Frequency")
         ylabel("Magnitude [Volts]")
            grid on
   hold off

subplot(4, 1, 2)
   stem(FrequencyRange,AvgCurrentPull)
      title("Injected Current")
         xlabel("Frequency")
         ylabel("Magnitude [Amps]")
            grid on


subplot(4, 1, 3)
   stem(FrequencyRange, AvgBatteryImpedence)
      title("Average Magnitude of Battery Impedence")
      xlabel("Frequency")
      ylabel("Magnitude [Ohms]")
         grid on

subplot(4, 1, 4)
   scatter(real(ComplexImpedence),imag(ComplexImpedence))
      title("Battery's Impedence (X+jY)")
      xlabel("Real (X) [Ohms]")
      ylabel("Imaginary (jY) [Ohms]")
         grid on
```

Single Frequency Data Processing Code

```
clear all;
close all;
clc;

%variable decloration
BatteryVoltage  = 3.5;
CurrentResistor = 1;
Frequency       = 13;
CurrentGain     = 9.2;

PhaseRunner = 0:(8*pi)/255:8*pi;
t = 1:256;

%Generates file name to pull the data from
filename = sprintf('%s_%d','TestData',Frequency);
filename = strcat(filename, ".csv")

%Pulls columns in from Excel Sheet that the serial terminal will popualte
Voltage1 = xlsread(filename,"A2:A257"); %Gets data for voltage calculations
Voltage2 = xlsread(filename,"A258:A513");
Voltage3 = xlsread(filename,"A514:A769");
Voltage4 = xlsread(filename,"A770:A1025");
Voltage5 = xlsread(filename,"A1026:A1281");

CurrentVoltage1 = xlsread(filename,"A1282:A1537"); %Gets data for current calculations
CurrentVoltage2 = xlsread(filename,"A1538:A1793");
CurrentVoltage3 = xlsread(filename,"A1794:A2049");
CurrentVoltage4 = xlsread(filename,"A2050:A2305");
CurrentVoltage5 = xlsread(filename,"A2306:A2561");

%Transposes column array to be a row array
Voltage1 = Voltage1';
Voltage2 = Voltage2';
Voltage3 = Voltage3';
Voltage4 = Voltage4';
Voltage5 = Voltage5';

CurrentVoltage1 = CurrentVoltage1';
CurrentVoltage2 = CurrentVoltage2';
CurrentVoltage3 = CurrentVoltage3';
CurrentVoltage4 = CurrentVoltage4';
CurrentVoltage5 = CurrentVoltage5';
```

```
%Subtracts the battery voltage out of the reading
Voltage1_mag = Voltage1 - BatteryVoltage;
Voltage2_mag = Voltage2 - BatteryVoltage;
Voltage3_mag = Voltage3 - BatteryVoltage;
Voltage4_mag = Voltage4 - BatteryVoltage;
Voltage5_mag = Voltage5 - BatteryVoltage;

%Divides by the test resistor value to get current
Current1 = CurrentVoltage1 / CurrentResistor;
Current2 = CurrentVoltage2 / CurrentResistor;
Current3 = CurrentVoltage3 / CurrentResistor;
Current4 = CurrentVoltage4 / CurrentResistor;
Current5 = CurrentVoltage5 / CurrentResistor;

%Divide by the current gain to get actual gain
Current1 = Current1 / CurrentGain;
Current2 = Current2 / CurrentGain;
Current3 = Current3 / CurrentGain;
Current4 = Current4 / CurrentGain;
Current5 = Current5 / CurrentGain;

%Take average of the multiple voltage data trials to isolate noise
Voltage_TotalMag = Voltage1_mag+ Voltage2_mag + Voltage3_mag + Voltage4_mag + Volt-
age5_mag;
Voltage_AvgMag = Voltage_TotalMag / 5;

%Takes avearge of the multiple current data trials to isolate noise
Current_TotalMag = Current1 + Current2 + Current3 + Current4 + Current5;
Current_AvgMag = Current_TotalMag / 5;

%Calculate battery impedence magnituded
for i = 1:256
   if Current_AvgMag(i) ~= 0
      BatteryImpedence(i) = Voltage_AvgMag(i) ./ Current_AvgMag(i);
   else
      BatteryImpedence(i) = 0;
   end
end

%Find phase shift
TestSin = sin(Frequency.*PhaseRunner);

%Find actualy battery impedence
Average_Battery_Phase    = mean(Voltage_AvgMag);
Amplitude_Battery_Phase  = (max(Voltage_AvgMag)- min(Voltage_AvgMag))/2;
Peak_Battery      = 100 * 2 * pi;
Phaseshift_Battery = 0;
```

```
%myfit = NonLinearModel.fit(PhaseRunner,Voltage_AvgMag,'y~b0+b1*sin(b2*x1+b3)',[Aver-
age_Battery_Phase,Amplitude_Battery_Phase,Peak_Battery,Phaseshift_Battery]);
myfit = NonLinearModel.fit(PhaseRunner,Voltage1_mag,'y~b0+b1*sin(b2*x1+b3)',[Average_Bat-
tery_Phase,Amplitude_Battery_Phase,Peak_Battery,Phaseshift_Battery]);

Z_PhaseShif_radians = acos(dot(myfit.Fitted,TestSin)/(norm(TestSin)*norm(myfit.Fitted)));
Z_PhaseShif_degrees = Z_PhaseShif_radians * 360 / (2*pi);

%Polar to Rectangular
x = BatteryImpedence.*cos(Z_PhaseShif_degrees);
y = BatteryImpedence.*sin(Z_PhaseShif_degrees);
ComplexImpedence = complex(x,y);

disp(Z_PhaseShif_radians)

%Graphing code
figure(1)
subplot(6, 1, 1)
   hold on
      stem(Voltage1_mag)
      stem(Voltage2_mag)
      stem(Voltage3_mag)
      stem(Voltage4_mag)
      stem(Voltage5_mag)
         title("Pulled in Data Array")
         xlabel("Data Point")
         ylabel("Magnitude [Volts]")
         legend('First Data Array', 'Second Data Array', 'Thrid Data Array', 'Fourth data array', 'Fifth
Data Array')
            grid on
   hold off

subplot(6, 1, 2)
   stem(Current_AvgMag)
      title("Average Magnitude of Branch Current")
         xlabel("Data Point")
         ylabel("Magnitude [Amps]")
            grid on

subplot(6, 1, 3)
   stem(Voltage_AvgMag)
      title("Average Magnitude of Battery's Voltage Drop")
      xlabel("Data Point")
      ylabel("Magnitude [Volts]")
         grid on
```

```
subplot(6, 1, 4)
   stem(BatteryImpedence)
      title("Magnitude of the Battery's Impedence")
      xlabel("Data Point")
      ylabel("Magnitude [Ohms]")
         grid on

subplot(6, 1, 5)
   hold on
      plot(PhaseRunner,myfit.Fitted)
      plot(PhaseRunner,Voltage_AvgMag)
         title("Sinusoidal Fit")
         xlabel("Data Points")
         ylabel("Magnitude [Volts]")
         legend('Fitted Equation','Test Points')
            grid on
   hold off

subplot(6, 1, 6)
   scatter(real(ComplexImpedence),imag(ComplexImpedence))
      title("Battery's Impedence")
         xlabel("Real (X) [Ohms]")
         ylabel("Imaginary (jY) [Ohms]")
            grid on
```

Data Formatting – Python

```
#!/usr/bin/python

"""preprocessing.py: This script takes the terminal output of the Impedance-based BMS device and converts from UTF-8 characters to actual voltage values read by ADC on microcontroller."""

import argparse
import binascii
import csv
import os
import sys
from pathlib import Path

#=====================================================
def formatting(currentFile, outputFile):
	readCapture = open(currentFile, "rb") # Opens & reads contents of given text file as binary
	lines = readCapture.readlines()  # Stores lines into a list with one entry per line from txt file
	readCapture.close()
# Closes object file

	# Initializes a list of list required to write into a CSV file
	# There are as many indicies as there is data points collected
	csvData = [ [] for cell in range(0, round(len( lines[5][:] ) / 2 )) ]

	sublist_index = 0 # Index to traverse sublists
	for c in range(0, len(lines[5])-1):		# Data is specifically found on line 5
		if(c % 2 == 0): # Every two characters are combined together
			adc_value = ((lines[5][c]) << 8) + lines[5][c+1]	# Complete 10-bit value from ADC
			voltage = float(adc_value) * float(5/1023) # Conversion of binary to floating point
			csvData[sublist_index].append(voltage)
			# Appends converted value to list of lists used in making CSV file
			sublist_index += 1
								# Increases index by 1

	csvData.insert(0, ['Voltage Reading'])
	# Appends a title for the column

	del csvData[-1]

	with open(outputFile, 'w', newline='') as csvCaptureFile:			# Opens and prepares csv file for writing to
		writer = csv.writer(csvCaptureFile)
		writer.writerows(csvData)  # Writes each sublist of csvData to newline of capture.csv
	csvCaptureFile.close() # Closes csv file
```

```
#=====================================================

def main():
	parser = argparse.ArgumentParser(description='Takes raw HEX data from impedance testing and converts into Base-10 equivalent.')
	parser.add_argument('-i', '--input', help='Full filepath of folder containing raw data in txt files from testing', required=True)
	parser.add_argument('-o', '--output', help='Full filepath of output file of the newly formatted raw data', required=True)
	args = parser.parse_args()

	workingDirectory = Path(args.input)
		# Automatically converts given input filepath to right format for the current operating system

	if workingDirectory.exists():
		print('Woohoo, directory exists.')
	else:
		print("The given input directory does not exist.")
		sys.exit(2)
						# Exits the script

	workingFiles = os.listdir(workingDirectory)

	for files in workingFiles:
		formatting( Path(args.input + '/' + files + '/' + files + '.txt'), (args.output + '_' + files + '.csv') )

#=====================================================

if __name__ == "__main__":
	main()

#=====================================================
```

13.8 Appendix I: Microcontroller Code

Main Function loop

```
#include <avr/io.h>
#include <avr/eeprom.h>
#include "ADC.c"
#include "twimaster.c"
#include "frequency_table.c"
#include "usart.c"

#define TRUE 1                                                                  //
Defines TRUE as always being 1, used for control purposes
#define FALSE 0                                                                 //
Defines FALSE as always being 0, used for control purposes
#define NUM_DATA_POINTS (5*256)                                                 // Number of data
points to collect from ADC during acquisition
#define LENGTH(x) ( sizeof(x) / sizeof((x)[0]) )// Macro to determine number of elements in a given
array

// Flags used for state machine transitions, all initialized to FALSE (0)
volatile unsigned char ADC_ACQUISITION_COMPLETE                 = FALSE;
volatile unsigned char BUTTON_INPUT                                     = FALSE;
volatile unsigned char USART_TRANSMISSION_COMPLETE = FALSE;
volatile unsigned char USART_RX_RECEIVED_FLAG           = FALSE;
volatile unsigned char I2C_TRANSMIT_SUCESS_FLAG                 = FALSE;

// Global variables
volatile unsigned char USART_RECEIVED_DATA = 0x00;
volatile unsigned char voltages [(NUM_DATA_POINTS+NUM_DATA_POINTS)];
volatile unsigned char current [(NUM_DATA_POINTS+NUM_DATA_POINTS)];

// Strings sent to terminal to help user know what state they are in
char initialize_statement[]             = {"Initializing. . ."};
char wait_statement[]                   = {"Begin test?"};
char input_statement[]                  = {"Choose frequency: "};
char set_statement[]                    = {"Setting resistor. . ."};
char acquire_statement[]                = {"Acquiring Data . . ."};
char transmit_statement[]               = {"Sending Data. . ."};
```

```
//============================== State Machine ==============================
enum States {INITIALIZE, WAIT, INPUT, SET, ACQUIRE, TRANSMIT} state;

void StateManager() {

	unsigned short voltage_reading = 0;

	// Switch statement for state transitions
	switch(state) {
		case INITIALIZE:
			break;
		case WAIT:
			if( BUTTON_INPUT == TRUE ) {
				BUTTON_INPUT = FALSE;
				state = INPUT;
			}
			else {
				state = WAIT;
			}
			break;
		case INPUT:
			if( USART_RX_RECEIVED_FLAG == TRUE ) {
				USART_RX_RECEIVED_FLAG = FALSE;
				state = SET;
			}
			else {
				state = INPUT;
			}
			break;
		case SET:
			if( I2C_TRANSMIT_SUCESS_FLAG == TRUE ) {
				I2C_TRANSMIT_SUCESS_FLAG = FALSE;
				state = ACQUIRE;
			} else {
				state = SET;
			}
			break;
		case ACQUIRE:
			if( ADC_ACQUISITION_COMPLETE == TRUE ) {
				ADC_ACQUISITION_COMPLETE = FALSE;
				state = TRANSMIT;
			}
			else {
				state = TRANSMIT;
			}
			break;
		case TRANSMIT:
			if( USART_TRANSMISSION_COMPLETE == TRUE ) {
				USART_TRANSMISSION_COMPLETE = FALSE;
```

```
                        state = WAIT;
                }
                else {
                        state = TRANSMIT;
                }
                break;
        default:
                break;
}
//Switch statement for state actions
switch(state) {
        case INITIALIZE:
                USART_init();
                USART_transmit_string(initialize_statement);
                ADC_init();
                init_frequency_table();
                state = WAIT;
                break;
        case WAIT:
                USART_transmit_string(wait_statement);
                while((PINC & 0x01) == 0);
                BUTTON_INPUT = TRUE;
                break;
        case INPUT:
                USART_transmit_string(input_statement);
                while ( !(UCSR0A & (1 << RXC0)) );
                USART_RECEIVED_DATA = UDR0;
                USART_RX_RECEIVED_FLAG = TRUE;
                break;
        case SET:
                USART_transmit_string(set_statement);
                while((PINC & 0x01) == 0);
                I2C_TRANSMIT_SUCESS_FLAG = TRUE;
                break;
        case ACQUIRE:
                USART_transmit_string(acquire_statement);

                for(unsigned int i = 0; i < LENGTH(voltages); i = i + 2) {
                        voltage_reading = ADC;
                        voltages[i]     = (char)((voltage_reading & 0x0300) >> 8);
                        voltages[i+1]   = (char)(voltage_reading & 0x00FF);
                }
                switch_ADC(1);
                for(unsigned int i = 0; i < LENGTH(current); i = i + 2) {
                        voltage_reading = ADC;
                        current[i]      = (char)((voltage_reading & 0x0300) >> 8);
                        current[i+1]    = (char)(voltage_reading & 0x00FF);
    }
                switch_ADC(0);
```

```
                        ADC_ACQUISITION_COMPLETE = TRUE;
                        break;
                case TRANSMIT:
                        USART_transmit_string(transmit_statement);
                        for(unsigned int j = 0; j < LENGTH(voltages); j++) {
                                USART_transmit(voltages[j]);
                        }
                        for(unsigned int j = 0; j < LENGTH(current); j++) {
                                USART_transmit(current[j]);
                        }
                        USART_transmit_newline();
                        USART_TRANSMISSION_COMPLETE = TRUE;
                        break;
                default:
                        state = INITIALIZE;
                        break;
        }
}
int main(void) {
        DDRC = 0xFE; PORTC = 0x00;              // Configure PORTC PIN0 as input, initialize
to 0s

        state = INITIALIZE;                     // Makes the first state Initialize on startup
   while (1) {
                StateManager();
   }
}
```

ADC.C Header

```
void ADC_init( void ) {
	ADMUX = 0x00; //Default value
	ADCSRA |= (1 << ADEN) | (1 << ADSC) | (1 << ADATE);
	// ADCSRA = 0xC0
	// Current configuration is Single Conversion mode.
	// ADEN: setting this bit enables analog-to-digital conversion.
	// ADSC: setting this bit starts the conversion.
	// Current configuration has Vin connected to PA0.
	// The AREF (Vref) pin is connected directly to the +5 Volt power supply
	// AREF is the pin located between PA7 and the ground pin.
}

// Right now, only switches ADC to channel 1
void switch_ADC( unsigned char whichADC ) {
	if(whichADC == 1) ADMUX = 0x01;
	else {
		ADMUX = 0x00;
	}
	ADCSRA |= (1 << ADEN) | (1 << ADSC) | (1 << ADATE);
	// ADCSRA = 0xC0
	// Current configuration is Single Conversion mode.
	// ADEN: setting this bit enables analog-to-digital conversion.
	// ADSC: setting this bit starts the conversion.
	// Current configuration has Vin connected to PA0.
	// The AREF (Vref) pin is connected directly to the +5 Volt power supply
	// AREF is the pin located between PA7 and the ground pin.
}
```

Twimaster.c Header file

```
/*************************************************************************
* Title:    I2C master library using hardware TWI interface
* Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
* File:     $Id: twimaster.c,v 1.4 2015/01/17 12:16:05 peter Exp $
* Software: AVR-GCC 3.4.3 / avr-libc 1.2.3
* Target:   any AVR device with hardware TWI
* Usage:    API compatible with I2C Software Library i2cmaster.h
*************************************************************************/
#include <inttypes.h>
#include <compat/twi.h>
#include <i2cmaster.h>
/* define CPU frequency in hz here if not defined in Makefile */
#ifndef F_CPU
#define F_CPU 8000000UL
#endif
/* I2C clock in Hz */
#define SCL_CLOCK  100000L
/*************************************************************************
 Initialization of the I2C bus interface. Need to be called only once
*************************************************************************/
void i2c_init(void)
{
  /* initialize TWI clock: 100 kHz clock, TWPS = 0 => prescaler = 1 */
  TWSR = 0;                         /* no prescaler */
  TWBR = ((F_CPU/SCL_CLOCK)-16)/2;  /* must be > 10 for stable operation */
}/* i2c_init */
/*************************************************************************
  Issues a start condition and sends address and transfer direction.
  return 0 = device accessible, 1= failed to access device
*************************************************************************/
unsigned char i2c_start(unsigned char address)
{
    uint8_t   twst;
        // send START condition
        TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
        // wait until transmission completed
        while(!(TWCR & (1<<TWINT)));
        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_START) && (twst != TW_REP_START)) return 1;
        // send device address
        TWDR = address;
        TWCR = (1<<TWINT) | (1<<TWEN);
        // wail until transmission completed and ACK/NACK has been received
        while(!(TWCR & (1<<TWINT)));
        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;
        return 0;
}/* i2c_start */
```

```
/*************************************************************************
 Issues a start condition and sends address and transfer direction.
 If device is busy, use ack polling to wait until device is ready

 Input:   address and transfer direction of I2C device
*************************************************************************/
void i2c_start_wait(unsigned char address)
{
   uint8_t   twst;


   while ( 1 )
   {
            // send START condition
            TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

         // wait until transmission completed
         while(!(TWCR & (1<<TWINT)));

         // check value of TWI Status Register. Mask prescaler bits.
         twst = TW_STATUS & 0xF8;
         if ( (twst != TW_START) && (twst != TW_REP_START)) continue;

         // send device address
         TWDR = address;
         TWCR = (1<<TWINT) | (1<<TWEN);

         // wail until transmission completed
         while(!(TWCR & (1<<TWINT)));

         // check value of TWI Status Register. Mask prescaler bits.
         twst = TW_STATUS & 0xF8;
         if ( (twst == TW_MT_SLA_NACK )||(twst ==TW_MR_DATA_NACK) )
         {
            /* device busy, send stop condition to terminate write operation */
               TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

               // wait until stop condition is executed and bus released
               while(TWCR & (1<<TWSTO));

            continue;
         }
         //if( twst != TW_MT_SLA_ACK) return 1;
         break;
    }

}/* i2c_start_wait */
```

```
/*****************************************************************************
 Issues a repeated start condition and sends address and transfer direction

 Input:   address and transfer direction of I2C device

 Return:  0 device accessible
          1 failed to access device
*****************************************************************************/
unsigned char i2c_rep_start(unsigned char address)
{
    return i2c_start( address );

}/* i2c_rep_start */


/*****************************************************************************
 Terminates the data transfer and releases the I2C bus
*****************************************************************************/
void i2c_stop(void)
{
    /* send stop condition */
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

        // wait until stop condition is executed and bus released
        while(TWCR & (1<<TWSTO));

}/* i2c_stop */


/*****************************************************************************
  Send one byte to I2C device

  Input:    byte to be transfered
  Return:   0 write successful
            1 write failed
*****************************************************************************/
unsigned char i2c_write( unsigned char data )
{
    uint8_t   twst;

        // send data to the previously addressed device
        TWDR = data;
        TWCR = (1<<TWINT) | (1<<TWEN);

        // wait until transmission completed
        while(!(TWCR & (1<<TWINT)));

        // check value of TWI Status Register. Mask prescaler bits
        twst = TW_STATUS & 0xF8;
        if( twst != TW_MT_DATA_ACK) return 1;
        return 0;

}/* i2c_write */
```

```
/*****************************************************************************
 Read one byte from the I2C device, request more data from device

 Return:  byte read from I2C device
*****************************************************************************/
unsigned char i2c_readAck(void)
{
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
        while(!(TWCR & (1<<TWINT)));
   return TWDR;

}/* i2c_readAck */


/*****************************************************************************
 Read one byte from the I2C device, read is followed by a stop condition

 Return:  byte read from I2C device
*****************************************************************************/
unsigned char i2c_readNak(void)
{
        TWCR = (1<<TWINT) | (1<<TWEN);
        while(!(TWCR & (1<<TWINT)));
   return TWDR;

}/* i2c_readNak */
```

Frequency.c Header file

```
#define NUM_FREQUENCIES 100       // Number of frequencies available to set for testing

// Defining a variable type that has two fields
struct Frequencies {
        unsigned char terminal_value;
        unsigned char set_value;
        double adc_delay;
};

// Initializes array of struct Frequencies
struct Frequencies frequency_table[NUM_FREQUENCIES];

// Order is important
unsigned char resistance_hex[NUM_FREQUENCIES] = {
        0xFF, 0x3F, 0x24, 0x19, 0x13, 0x0F, 0x0D, 0x0B, 0x0A, 0x09,
        0x08, 0x07, 0x06, 0x06, 0x05, 0x05, 0x05, 0x05, 0x04, 0x04,
        0x04, 0x04, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
        0x02, 0x02, 0x02, 0x02, 0x02, 0x00, 0x00, 0xFF, 0xC4, 0x9F,
        0x86, 0x74, 0x66, 0x5B, 0x52, 0x4B, 0x45, 0x3F, 0x3B, 0x37,
        0x34, 0x31, 0x2E, 0x2C, 0x2A, 0x29, 0x27, 0x26, 0x24, 0x23,
        0x03, 0x20, 0x1F, 0x1E, 0x1D, 0x1C, 0x1B, 0x1A, 0x19, 0x15,
        0x13, 0x0F, 0x0D, 0x0B, 0x0A, 0x09, 0x08, 0x07, 0x06, 0x06,
        0x05, 0x05, 0x05, 0x04, 0x04, 0x04, 0x04, 0x04, 0x03, 0x03,
        0x03, 0x03, 0x03, 0x03, 0x03, 0x02, 0x02, 0x02, 0x02, 0x02
};

// Order is important
double delay[NUM_FREQUENCIES] = {
        0.390, 0.098, 0.056, 0.039, 0.030, 0.024, 0.020, 0.018, 0.016,
        0.014, 0.013, 0.011, 0.010, 0.090, 0.009, 0.008, 0.008, 0.008,
        0.007, 0.007, 0.007, 0.006, 0.006, 0.006, 0.006, 0.005, 0.005,
        0.005, 0.005, 0.004, 0.004, 0.004, 0.004, 0.004, 0.004, 0.001,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0
};

// Assigns array values to specific structs
void init_frequency_table(void) {
        for(unsigned char i = 0; i <= (NUM_FREQUENCIES-1); i++) {
                frequency_table[i].terminal_value       = (i+1);
                frequency_table[i].set_value            = resistance_hex[i];
                frequency_table[i].adc_delay            = delay[i];
        }
}
```

USART.c Header

```
#define BAUDRATE 51//ATMEGA1284P data sheet suggests this value to produce ~9600 bps baud rate

void USART_init( void ) {
	// USART initialization
	// Communication parameters: 8 data bits, 1 stop bit, no parity checking
	// USART Transmitter: ON
	// USART Receiver: ON
	// USART Mode: Asynchronous
	// USART Baud Rate: 9600 bps
	UBRR0 = BAUDRATE;		// Sets baud rate
	UCSR0B = (1 << RXEN0) | (1 << TXEN0);	// Enable transmitter and receiver
	UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);  // Enables 1 stop bit mode
}

void USART_transmit( char data ) {
	// Wait for empty transmit buffer
	while( !( UCSR0A & (1 << UDRE0)) );
	// Put data into buffer, sends the data
	UDR0 = data;
}

void USART_transmit_newline( void ) {
	USART_transmit(0x0D);
	USART_transmit(0x0A);
}

void USART_transmit_string( char string[] ) {
	unsigned char i = 0;
	while(string[i] != 0x00) {
		USART_transmit(string[i]);
		i++;
	}
	USART_transmit_newline();
}

unsigned char USART_receive( void ) {
	// Wait for data to be received
	while ( !(UCSR0A & (1 << RXC0)) );
	// Get and return received data from buffer
	return UDR0;
}
```