Spring 2020

Bioimpedance Sensor

Ryan Byo
rtb39@zips.uakron.edu

Kevin Libertowski
kal125@zips.uakron.edu

BioImpedance Sensor

Senior Project Final Report

Ryan Byo

Kevin Libertowski

Mitchell Sutyak

Steve Weimer

Faculty Advisor – Dr. Robert Veillette

April 24, 2020

i. Table of Contents (SW)

ii. List of Figures (SW, MS, KL)

iii. List of Tables (SW)

Abstract (MS)

Measuring the impedance of the body is a process which can be correlated to several different aspects of the human body, more specifically, a user's hydration level. In order to find a user's impedance, first a signal must be generated and then passed through the user's body. In order to guarantee safety of the user, the signal is passed through various components, so the exact inputs are known. To send this input signal through the user's body, a set of electrodes will be used. Once the signal has passed through the user's body, it is then processed, first through a conversion of analog to digital, followed by an embedded processor, and finally sent to a wireless module via Bluetooth. Once at the wireless module, the information is sent to a server, and stored in a database. From here a user can access all previous readings and keep track of their impedance levels in the form of a mobile application.

1 - Problem Statement

1.1 – Need (SW, RB)

Many people put a great emphasis on personal health, but underestimate the importance of hydration, subjecting themselves to being either under or over-hydrated. While thirst is a strong indicator of a person's hydration level – and is sufficient for most people – consumers have no way of easily monitoring their hydration and tracking it through the day or over a longer period of time. In addition to serious medical conditions which are exacerbated by hydration levels, the symptoms of mild dehydration (e.g. headaches and muscle cramps) and the less-critical effects of overhydration (e.g. inconvenience of frequent drinking and urination) constitute a need for achieving

measured and trackable hydration feedback which many consumers would find useful[11].

1.2 – Objective (SW, KL, MS)

The best solution for the consumer would be a hydration monitor which is compact enough for home or portable use. The user's hydration level can be measured by bioelectric impedance analysis. This is done by sending a signal through the body and analyzing the changes to the output signal; these changes vary by a user's hydration level. Once the signal is measured, the data will automatically be sent to a mobile application where it will be processed and displayed to the user. The data will also be uploaded and saved for each specific user. Similar to diet and fitness applications, the user will then be able to review historic hydration (BIA) levels and better-understand how to adapt their own fluid intake. Since the fluid intake per user is different based on factors – such as limb position, recent exercise, and skin temperature – it will be important to develop a range of hydration levels that can be clearly conveyed to the user, in addition to the user's quantified bioelectric impedance.

1.3 – Background

Basic Theory Behind the Concept (MS)

Bioelectrical impedance analysis (BIA) is a safe, simple, and inexpensive technique for the evaluation of body composition. Bioelectrical impedance analysis involves electrodes being placed on different parts of the body. When placed at the hands or the feet, it is possible to measure the impedance of the body, whereas placing the electrodes at a segmented part of the body allows for a more accurate reading in that segmented area. The impedance instrument then sends a very small and harmless AC

current through a set of electrodes, and then measures how the body impedes the current flow through the set of electrodes[5].

The physical principle behind bioelectrical impedance analysis lies in the cells that are grouped up in the organs and tissues that make up the human body. Cells contain intracellular water and are bathed in extracellular water. Bioelectrical impedance analysis is based on the potential difference or voltage drop when a weak AC current is sent through the body tissues[5]. The intracellular and extracellular waters behave as electrical conductors while the cell membranes behave as imperfect reactive elements. The electrical conduction is then dependent on fluid and electrolyte distribution. The opposition to the flow of a constant alternating AC current is the impedance, which is composed of an electrical resistance that changes based on the amount of fluid in the tissue.

In order to measure this impedance, it is important to understand what variables factor into the impedance. Since impedance is the opposition to the flow of an alternating current, it is dependent on the frequency of the applied current, defined in impedance magnitude (|Z|), and phase angle (). Bioimpedance is a complex quantity which is composed of resistance (R) and reactance (X_c). The resistance is caused by total body water, and the reactance is caused by the capacitance of the cells' membrane[3]. The following equations relate the variables above.

$$Z = R + jX_c \tag{1}$$

$$|Z| = \sqrt{R^2 + {X_c}^2} \tag{2}$$

$$\Phi = \tan^{-1}\left(\frac{X_c}{R}\right) \tag{3}$$

The resistance of an object is determined by a shape defined as length (L) and surface area (A). Material type is defined by resistivity (ρ), and reactance (X_c) of an object is defined as resistance to voltage variation across the object. The reactance is inversely related with signal frequency (f) and capacitance (C) [3]. The following equations relate the variables above.

$$R_{(\Omega)} = \rho_{(\Omega.m)} \left(\frac{L_{(m)}}{A_{(m^2)}} \right) \tag{4}$$

$$X_{c(\Omega)} = \frac{1}{2\pi f_{(Hz)} C_{Farad}} \tag{5}$$

Capacitance then is defined as the ability of a non-conducting object to save electrical charges. That is, the voltage differentiation across an object (dV/dt) and the current that is passed through the object (I(t)). In the following equations, the capacitance is directly proportional to the surface area (A), and inversely proportional to the distance (d) between the charged plates. Capacitance is also dependent on the permittivity of free space (ε_0) and the dielectric permittivity (ε_r), which is defined based on the materials between the plates.

$$C_{Farad} = \varepsilon_0 \varepsilon_r \left(\frac{A_{(m^3)}}{d_{(m)}} \right) \tag{6}$$

$$C_{Farad} = \frac{dV(t)}{dt} / I(t) \tag{7}$$

Body composition estimation using bioimpedance measurements is then based on the body volume (V_b), which can be determined through resistance measurement.

All equations above play a factor in how the impedance is measured, and as a result, they can be combined to give an accurate reading. Looking at the human body, the human body is a volume that is composed fat mass (FM) and fat free mass (FFM). Fat mass is a non-conductor of electric charge and is equal to the difference between body weight (Wt_{Body}) and fat free mass[3]. However, fat free mass is the conducting volume that passes electric current due to the conductivity of electrodes dissolved in body water. The total body water (TBW) is then the major compound of fat free mass and is equal to roughly 73.2% for normal hydration. These variables can be related together, and as a result be used to determine hydration levels.

The theory behind bioelectric impedance analysis is then that the impedance, which opposes the current flow will vary from tissue to tissue. In a statement from the Journal of Medical Engineering and Technology, "tissues containing large amounts of fluid and electrolytes have a high conductivity, and therefore a low impedance. On the contrary, fat and bones have a low conductivity, and therefore a high impedance"[5]. The bioelectric impedance analysis device then measures the impedance to the flow of the current as it passes through the body. As a result, estimates of the body composition parameters, such as hydration, can be obtained.

While the device is not expected to reach a clinical-level of accuracy for body hydration, the device might prove useful for those with chronic health issues which are treated and/or exacerbated by low or over-hydration. Chronic Kidney Disease is one example of a condition which is best-managed by maintaining a consistently-high, but not over-hydrated state. Dehydration results in frequent kidney stones which must be passed (painful) or removed surgically (expensive and risky). Overhydration causes

uncontrolled swelling of the kidneys which impinges on internal organs, nerves, and causes discomfort; for severe cases of kidney failure, overhydration is linked to increased mortality[9][10]. While an "at-home" consumer device is unlikely to be as accurate as clinical devices for measuring hydration, it could still serve as a useful tool for those with hydration-centric health risks due to the convenience of frequent non-invasive measurements. It would be necessary to avoid making any medical claims of the device – due to the possible liability concerns – but this need for a "properly-balanced" hydration level is an example of why drinking when thirsty is not always sufficient and why Users may want a convenient way to measure and track their hydration.(SW, this paragraph)

Current Uses (RB)

There are several different ways in which hydration levels are currently being measured. Shanholtzer and Patterson group these techniques into direct and indirect methods. One direct method is using radioactive isotopes to measure the amount of water in the body[1]. As the name implies, these methods directly measure body water, and therefore are more accurate. However, these techniques are usually more complex. This means they take more time and are usually less convenient, for both the user and the person doing the analysis. A direct method of measurement is not a realistic use-case for a consumer product since they require a large amount of training and human involvement. Indirect methods estimate body water using a measurement of a factor that effects hydration[1]. These methods are less accurate than direct methods, but they are much easier to use and interpret, therefore making them better for a consumer product. One of these methods is urine analysis. As most people know, urine color is an indicator

of hydration status. However, it is not exact and can be influenced by several other factors, such as food consumed and medication[1].

Bioelectrical impedance analysis is another indirect method of measuring hydration. In a statement from the National Institutes of Health, BIA is used in many different fields, including hospitals and clinicians' offices [2]. According to Shanholtzer and Patterson, current technologies measure BIA by sending an electrical current ranging from 5 to 500 kHz through the body via electrodes[1]. It is important that a range of frequencies be used, since different frequency signals give different information about the water in the body. These authors state that a 200 kHz wave is ideal for measuring the total water in the body[1]. This is due to the reaction of the body to various frequency signals. At a low frequency, the signal is unable to pass through the membrane of cells, and therefore can measure only liquid outside of the cell. In contrast, the cell membrane cannot block high frequency signals and cellular water can be taken into account [2]. This technique is ideal for a consumer product since it requires only sensors (electrodes) to be placed on the skin and the analysis is fairly simple compared to direct methods.

Shanholtzer and Patterson also conducted a study to determine the accuracy of BIA on measuring body water and interpreting the validity of using that data to determine hydration level. In this study, impedance was measured using a Multiscan 5000 multifrequency monitor. A current of 800 microamps was send through the user's body across the target frequency range of 5 to 500 kHz. This current amplitude allows the output signal to be read accurately and with little error, but also should not be felt by the user [2]. Data was collected through the use of two electrodes on the right hand and two electrodes on the right foot. The impedance for each of these frequencies was measured,

and the 5 kHz response and the 500 kHz responses were used to determine extracellular water and total body water, respectively. The analysis of these numbers then allowed the researchers to assign the participant a hydration level.

An important aspect of bioelectrical impedance analysis is to make the data-collection and analysis as accurate as possible since there are many variables which affect results, including limb position, disease/medication, gender, skin and core temperature, single-frequency versus multi-frequency, and age.[8] (SW - this paragraph only).

Limitations of the Current Designs or Technology (KL)

Some currently available BIA technologies can measure total body water (TBW) and extracellular body water (ECW). However, they fail to draw any meaningful conclusions from this information and simply output the result as percentages to the user. Some measurements such as body fat or lean mass are useful when displayed as percentages. That is because the user is more familiar with these values and has a better intuition about what they should be. For example, the user most likely knows that their body fat percentage should be around 10-20%. However, when a user is presented with their ECW as a percentage, they probably can't interpret that information meaningfully. Especially with the minor fluctuations that occur with one's ECW.

What the current designs are missing is a way to process and interpret the water data into something more useful, such as a user's current hydration level. The bioelectrical impedance data could be used to measure a user's hydration in a couple of different ways. One method is to input a user's physical stats, such as height, weight, age, sex, and compare the total body, intracellular and extracellular water measurements to a population model. A similar method is what is currently used to measure other body stats(3). The problem with this method is that it does not account for individual variances

and just looks at the population average when making calculations. One of the advantages of measuring hydration is the ability to calibrate for each specific user. The device could have a feature that allows the user to measure once when they are in a hyperhydrated and once when they are in a hypo-hydrated state. This would give a baseline calibration that on top of the population data, could be used to improve results. This is feasible since a user can control their hydration level by drinking a lot or abstaining from drinking water before the calibration. This is different from body composition measurements, since a user cannot easily change their body tissue makeup.

Piccoli et al. proposes another method for detecting fluid changes using BIA. Their method involves breaking down the impedance into the resistance and the reactance components. Then comparing the ratios of the resistance and reactance with the person's height. They call this the RXc graph[4]. This analysis technique could also be useful for measuring hydration when combined with calibration.

Relevant Existing Patents (SW)

Patent No. US006256532B1 is an Apparatus for Analyzing Body Composition Based on Bioelectrical Impedance Analysis and Method Thereof, which is a larger-format version of the consumer-scale apparatus being proposed.[6] This patent has very similar design to a consumer-scale version and discusses the process in great detail.

Patent No. US2015/0148623A1 is a Hydration Monitoring Sensor and Method for Cell Phones, Smart Watches, Occupancy Sensors, and Wearables; this patent is relevant as it seeks to measure the same principle and addresses many of the same challenges.[7] This patent consists of a broadband LED light which is transmitted to a subject user's skin which causes scattering across various wavelengths. By analyzing the reflected light,

the patent holder describes the ability to analyze to determine various biological parameters, including hydration level. The hydration level is not based on a single measurement, but as a comparison between fluid losses, fluid ingested, fluid balance, and rate of fluid loss.

This patent utilizes a system which has some similarities with hydration monitoring performed by BIA, including non-invasiveness, difficulty of achieving a consistent method of measurement, and energy efficiency. By being a method of capturing data externally without gathering any type of respiratory, urinary, or perspiratory sample, both the photosensor and BIA methods aim to make hydration analysis more convenient for users. By being more convenient than current methods, the patent hopes to achieve a wearable scale which can be incorporated into consumer devices (e.g. fitness wearables and smart phones). Much like BIA, the photosensor analysis has the disadvantage of capturing data under uncontrolled conditions; distance from subject, perspiration, and part of the body being monitored are all stated as important factors which need to be controlled for. In the proposed “bathroom scale” design, the User’s own body weight serves to ensure a reliable contact through a consistent signal path. Similar to the low-power signal generation and capture device involved in BIA, the use of a typical 3 mA LED allows for an efficient device capable of being powered by a small battery; to achieve a marketable device which is not reliant on being plugged-in to AC power, it’s critical to achieve an energy efficient design.

This system also has many differences from BIA, including the source signal, the breadth of data being captured, and complicating aspect of blood flow. The photosensor patent deals strictly with generating and capturing light, whereas BIA utilizes an

electrical signal being fed through the subject. While BIA only compares to input and output signals, the photosensor patent is concerned with a wide range of data obtained by the device (e.g. including heart rate, respiratory rate, distance, and motion). Unlike with bioelectrical impedance analysis which takes the whole current path as a single medium, the photosensor patent has to control for the different light-scattering properties of hemoglobin and water. While the differences between the two technologies don't directly impact BIA, these differences inform on potential complicating factors which have to be controlled for or approximated when building a model to convert the impedance level to hydration level.

1.4 – Marketing Requirements (SW, KL, MS, RB)

1. Safe: Voltage and current must be low-enough to be neither felt, nor dangerous. Needs to have some sort of warning label as well.

2. Affordable: Expense must be kept low enough for general consumers.

3. Size: Device must be small enough for convenient use by holding or standing upon.

4. User-Friendly: App must provide user with current and historic data in an intuitive way.

5. Battery Life: Device must have a long life on each charge or with each set of batteries.

6. Accurate: Device must deliver a consistent and accurate measure of the user's Impedance level. Range of Impedance levels will need to be clear and concise, as well as have a responsive display time.

2 – Engineering Analysis

2.1 – Circuits (MS, SW)

Beginning with a generated input voltage signal, the signal will first be sent through a voltage controlled current source (VCCS). The VCCS will force the output current to be proportional to the controlled input voltage signal. An input voltage of 3V will proportionally produce an input current of 1mA. This will be achieved by using an inverting OpAmp. The purpose of the inverting OpAmp is to obtain the exact output current and voltages. By loading the feedback path back to the amplifier and grounding the noninverting terminal with a resistor back to the inverting terminal, the exact output current and voltage is known, and the excess current flows back to the feedback path. Following the inverting OpAmp are two sets of differential amplifiers, which act as buffers, having two inputs, two copies of the same signal that are opposite in phase are sent through the amplifier. Any noise that is induced will be equal and opposite and thus canceling out. This way the generated input signal will be equal to the output, and the voltage gain will be known. Following the differential amplifier will be an inverting amplifier. To keep the circuit safe, the signal must stay below the pain threshold, which is well-below the AC let-go current of 5mA for children; DC let-go current is greater than 60 mA[12][13]. Due to the wide range of human body resistance[14], it is important to limit the signal voltage to keep the signal in a low-voltage range (below 30V).

The system power supply consists of three DC voltage outputs: +9V and ±18V. All three voltages are achieved through the use of standard 9 Volt DC alkaline batteries to reduce cost and make replacements user-friendly. ±18V are each achieved by wiring two batteries in series. Each battery has approximately 500mAh capacity. The limiting

factor for system power is the 9V circuit feeding the Explorer-16/32 board, which has an approximate current load of 200mA while the system is ON. With an expected uptime not to exceed 10seconds per measurement cycle, each measurement draws 0.56mAh making the expected battery life is at least 900measurements. With typical usage expected to be three measurements per day, the real-time battery life is at least 300days. Since the ±18V circuits are powering OpAmps drawing less than 1mA, the series batteries will last for years and likely need replaced only due to degradation from age.

2.2 – Electronics (MS)

An electrode is a conductor through which current enters or leaves an object, in this case the human body. The human body acts as a medium in which the transmitter and receiver make contact between the skin and the electrodes. Two electrodes will be used, one to pass the input current - the other for the input voltage - through the user (transmitter), and then pass the output current - or voltage - to the embedded processor (receiver).

2.3 – Signal Processing (KL)

Measuring the bio-impedance will involve applying a voltage to the body and . To accomplish this, a periodic square wave will be generated and sent through the body. A square wave was chosen because it will be much easier to generate as opposed to a sine wave.

Once the square wave is sent through the body, the current going through the body and the voltage applied across the body will need to be sampled. This will be done by connecting op-amp circuits at the input and output to the AD converter. The sampling

rate will need to be greater than the square wave frequency in order to get an accurate result.

After the points are sampled, the signal will be converted into the frequency domain using the Discrete Fourier Transform. Once in the frequency domain, the magnitude and phase of the first harmonic sine wave will be able to be analyzed, for the voltage and current. From here, the phasor voltage and current can be realized. Finally, the impedance can be calculated by dividing the phasor voltage by the phasor current.

To demonstrate this process, in MATLAB we simulated placing a square voltage wave across an RC circuit and calculated the current at discrete samples. We then wrote code to perform the steps mentioned above to calculate the impedance. The calculated impedance was then compared with the expected impedance of the RC circuit.

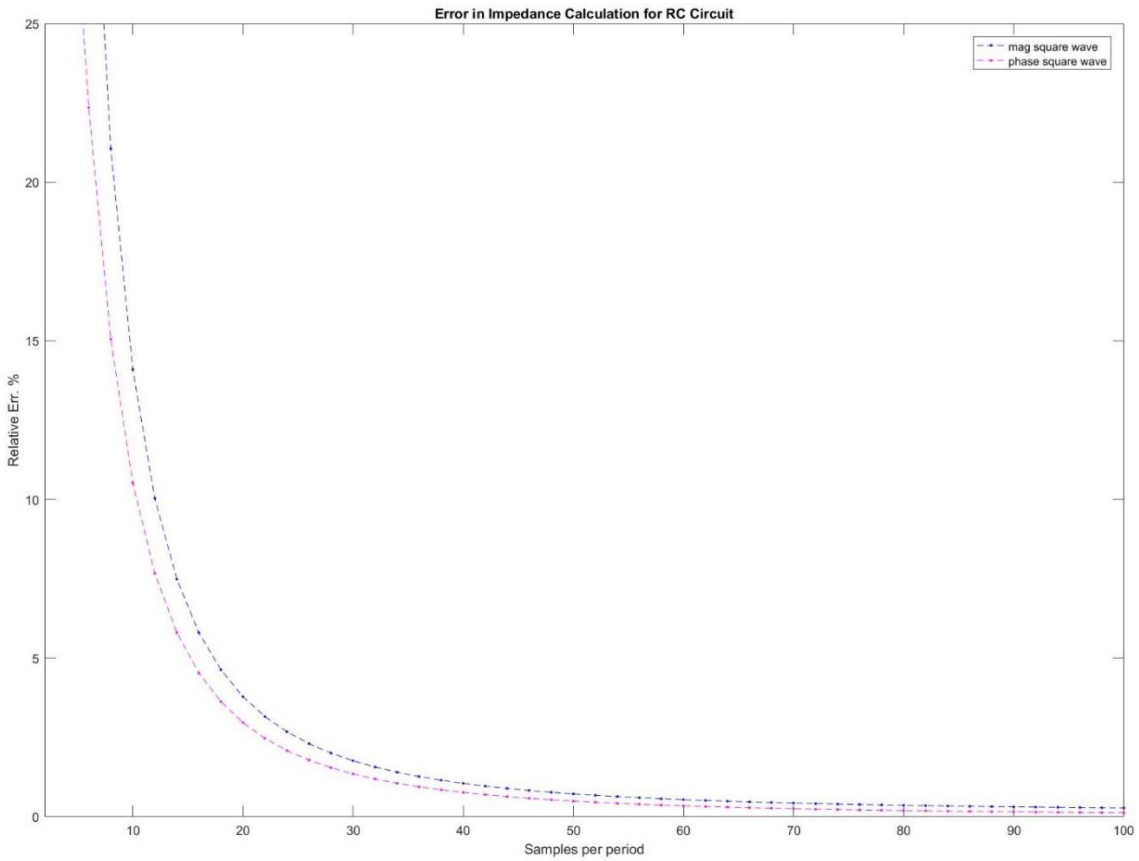

Figure-1: Impedance Calculation Error

Figure 1 shows the error in calculating the impedance using the square wave technique with varying number of sample points per period. The results were an error of around 3% in the magnitude and phase when 20 samples points were used, and less than 1% when 50 samples points were used. This shows that we will be able to calculate the impedance to a reasonable degree of accuracy by using a square wave and the discrete Fourier analysis. This also suggests that our target range for sampling speed should be somewhere between 20 and 100 samples per period in order to get an accurate result.

MATLAB code for error simulation:

```
num_steps = 50;
start = 1;
offset = start-1;
step = start:num_steps;

mag_err = zeros(1, num_steps-offset);
phase_err = zeros(1, num_steps-offset);

mag_err_sin = zeros(1, num_steps-offset);
phase_err_sin = zeros(1, num_steps-offset);

for index = step
    N = 2*index;
    Vs = 1;

    R = 1.5e3;
    C = .1e-9;
    T = 2e-6;
    f = 1/T;
    w = 2*pi*f;

    k = 1;

    Z_actual = R + 1/(1j*k*w*C);

    v_n = zeros(1, N-1);
    i_n = zeros(1, N-1);

    for n = 1:N
        if n < N/2
           v_n(n) = 1;
           i_n(n) = (Vs * exp(-((n/N)*T)/(R*C))) / R;
        else
           v_n(n) = 0;
           i_n(n) = -(Vs * exp(-(((n - N/2)/N)*T)/(R*C))) / R;
        end
```

```
    end

    vn_sin = zeros(1, N-1);
    in_sin = zeros(1, N-1);

    for n = 1:N
        t = (n/N)*T;
        vn_sin(n) = sin(w*t);
        I = 1/Z_actual;
        in_sin(n) = abs(I)*sin(w*t + angle(I));
    end

    V_fft = fft(v_n);

    I_fft = fft(i_n);

    V_k = 0;
    for n = 0:N-1
        V_k = V_k + v_n(n+1)*exp(-1j*2*pi*k*n/N);
    end

    I_k = 0;
    for n = 0:N-1
       I_k = I_k + i_n(n+1)*exp(-1j*2*pi*k*n/N);
    end

    Z_exp = V_k / I_k;

    mag_err(index-offset) = (abs(Z_exp)-abs(Z_actual))/abs(Z_actual);
    phase_err(index-offset) = (angle(Z_exp)-
angle(Z_actual))/angle(Z_actual);

    V_k_sin = 0;
    for n = 0:N-1
        V_k_sin = V_k_sin + vn_sin(n+1)*exp(-1j*2*pi*k*n/N);
    end

    I_k_sin = 0;
    for n = 0:N-1
       I_k_sin = I_k_sin + in_sin(n+1)*exp(-1j*2*pi*k*n/N);
    end

    Z_exp_sin = V_k_sin / I_k_sin;

    mag_err_sin(index-offset) = (abs(Z_exp_sin)-
abs(Z_actual))/abs(Z_actual);
    phase_err_sin(index-offset) = (angle(Z_exp_sin)-
angle(Z_actual))/angle(Z_actual);
end
```

2.4 – **Embedded Systems** (RB)

For our embedded controller, we are looking at using the Explorer 16/32 Development with the dsPIC33FJ256GP710 General Purpose PIM as our processor. The

given PIC24FJ1024GB610 PIM could not simultaneously sample two channels, so we could no longer use it.

One of the biggest roles our embedded system will play is doing the A/D conversion. We will be sampling waveforms of various frequencies, with a maximum of 50 kHz. According to the data sheet listed in the Appendix, the dsPIC33FJ256GP710 can perform at a rate of 1.1 Msps, or 1,100,000 samples per second, which should be fast enough for our needs. This allows for around 20 samples at our highest frequency, which will keep our error under 5%. Our A/D converter will also need to potentially handle multiple channels so we can sample multiple waveforms simultaneously. The data sheet reports that the dsPIC33FJ256GP710 can handle "simultaneous sampling of up to four analog input pins". We will only need two of those. The data sheet covers the A/D converter in detail in section 21.0.

Another key functionality we require of our embedded system is being able to display our readings for the users' purpose. The Explorer 16/32 board makes this simple using its "2-Line by 16-Character LCD Module" as described by the User's Guide shown in the appendix. The dsPIC33FJ256GP710 chip can communicate with the LCD by controlling the signals of the Truly TSB1G7000-E. We will also need the embedded system to be able to wireless communicate. Along with a separate chip, the dsPIC33FJ256GP710 can use UART to communicate with a mobile phone, which will run our application.

All of these features, along with processing to do calculations, can be done with these components and the MPLAB software.

3 – Engineering Requirements Specification (SW, RB, MS)

Table 1 – Engineering Requirements		
Marketing Requirements	**Engineering Requirements**	**Justification**
1	Generate & measure accurate Voltage across Body (1V – 18V) ±10%	Must be safe for all users. Keeping voltage within OpAmp rails will maintain safe power levels.
1	Generate & measure accurate signal Current of 1mA ±10%	Must be safe under all conditions. Current not to exceed 5mA.
6	Impedance Magnitude & Phase measurement & calculation precision within 5% (data points in tight cluster)	To be consistent and repeatable, Impedance data points must have little variance.
6	Impedance Magnitude & Phase measurements & calculation accuracy within 10%	To be useful, Impedance must be found to be close to known value.
6	Frequency Steps of: 50Hz, 500Hz, 5kHz, 50kHz	Low-frequency for extra-cellular resistance, high-frequency for intra-cellular capacitance
4, 6	Wireless range of up to 10m	User will be close to device; low power saves on battery life.
4, 6	Remote Data Storage & Retrieval	Impedance calculation can be done locally, but hydration model and historic data requires database interaction.
3, 5	System will be portable with an energy storage life of 6months	Batteries allow compact, portable device with long life.
6	AC Signal analysis will be used to calculate Impedance Magnitude & Phase	Board limitations and to ease magnitude and phase calculations
4	Impedance Magnitude & Phase Display at device & App	Must show users impedance in a user-friendly manner.

Marketing Requirements:

1. **Safe:** Voltage and current must be low-enough to be neither felt, nor dangerous. Needs to have some sort of warning label as well.
2. **Affordable:** Expense must be kept low enough for general consumers.
3. **Size:** Device must be small enough for convenient use by holding or standing upon.
4. **User-Friendly:** App must provide user with current and historic data in an intuitive way.
5. **Battery Life:** Device must have a long life on each charge or with each set of batteries.
6. **Accurate:** Device must deliver a consistent and accurate measure of the user's Impedance level. Range of Impedance levels will need to be clear and concise, as well as have a responsive display time.

4– Engineering Standards Specifications (MS)

Table 2 – Engineering Standards

Specification	Standard	Use
Safety	NIOSH NEC	Bodies reaction for different currents (~1mA desired).
Communications	USB & Bluetooth	USB to program the board and Bluetooth to send data to the mobile application.
Data Formats	Decimals	Magnitude & Phase.
Design Methods	LTSpice	Circuit design & simulation.
Programming Languages	Embedded – C App – JavaScript, Dart, SQL	C to program the embedded processor, JavaScript (Node.js) for AWS Lambda functions, Dart for mobile app and SQL for Postgres database
Connector Standards	USB	Program and debug board.

5– Accepted Technical Design

Block Diagram Level 0 (SW, RB, KL)

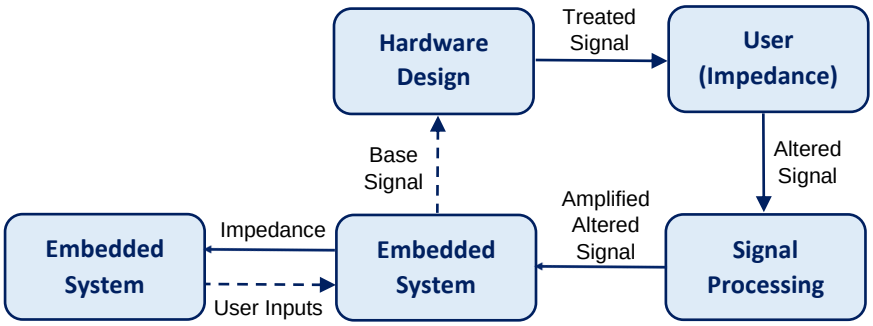

Figure-2: Block Diagram Level 0

5.1 – Hardware Design – Level 1 (SW)

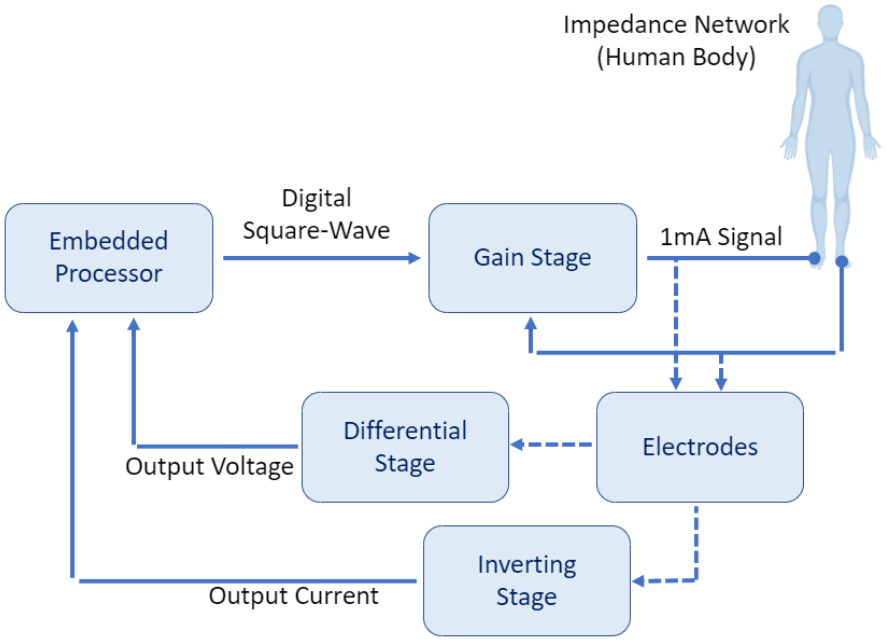

Figure-3: Hardware Block Diagram Level 1

Table 3.1 – Functional Requirements: Voltage-Controlled Current Source	
Module	**Voltage Controlled Current Source**
Designers	Mitchell Sutyak & Steve Weimer
Inputs	Generated input voltage signal
Outputs	Specified input current signal
Description	Current produced in accordance with sign and magnitude of controlled input voltage

Table 3.2 – Functional Requirements: Differential Amplifiers	
Module	**Differential Amplifiers**
Designers	Mitchell Sutyak & Steve Weimer
Inputs	Voltages from Sense Resistor & Across Body
Outputs	Input current and measurement voltage
Description	Differential gain control

Table 3.3 – Functional Requirements: Inverting Operational Amplifier	
Module	**Inverting Operational Amplifier**
Designers	Mitchell Sutyak & Steve Weimer
Inputs	Digital Square-Wave from Embedded Processor
Outputs	Inverting Output to Inverting Terminal, based on Feedback Impedance
Description	Maintains Current at desired level across Feedback Impedance (Body)

Table 3.4 – Functional Requirements: Electrodes	
Module	**Electrodes**
Designers	Mitchell Sutyak
Inputs	Exact input current
Outputs	Output current signal
Description	Connection between the device and the user

Hardware Level 1 Block Diagram (MS)

The circuit design shows the major components needed. Once the input signal has been generated, it will need to pass through a series of components in order to be safely sent through the user via two electrodes. The first component is a voltage controlled current source, followed by a differential amplifier, an inverting amplifier and finally sent through the electrodes. The purpose of these components is to first specify the current being sent through the circuit based off a controlled voltage input. Next, to account for any noise and voltage gain, a differential amplifier is used to buffer or clean the signal. The purpose of the inverting amplifier will be to know the exact current that will be sent into the electrodes.

An input current of 1mA will be injected into the body. The voltage is a product of the input current and the body resistance, which has a wide potential range. When the human body is perfectly dry and calluses have formed on the skin, the resistance of the external body can be as high as 1MΩ, however, this is not a realistic resistance for the average user. Considering all factors that can reduce a user's external resistance, a realistic level of resistance for a user can be as low as 300Ω. Using this as our baseline, the 1V and 1mA values are expected to be safe for each user. The inverting amplifier will then send any excess current back into the feedback path of the amplifier, and finally sent through the electrodes. The electrodes will have two components, the transmitting component and the receiving component. After the signal has traveled through the user's body, it will then be sent to the embedded processor where A/D conversion will take place.

5.2 – Hardware Design – Level 2 (MS)

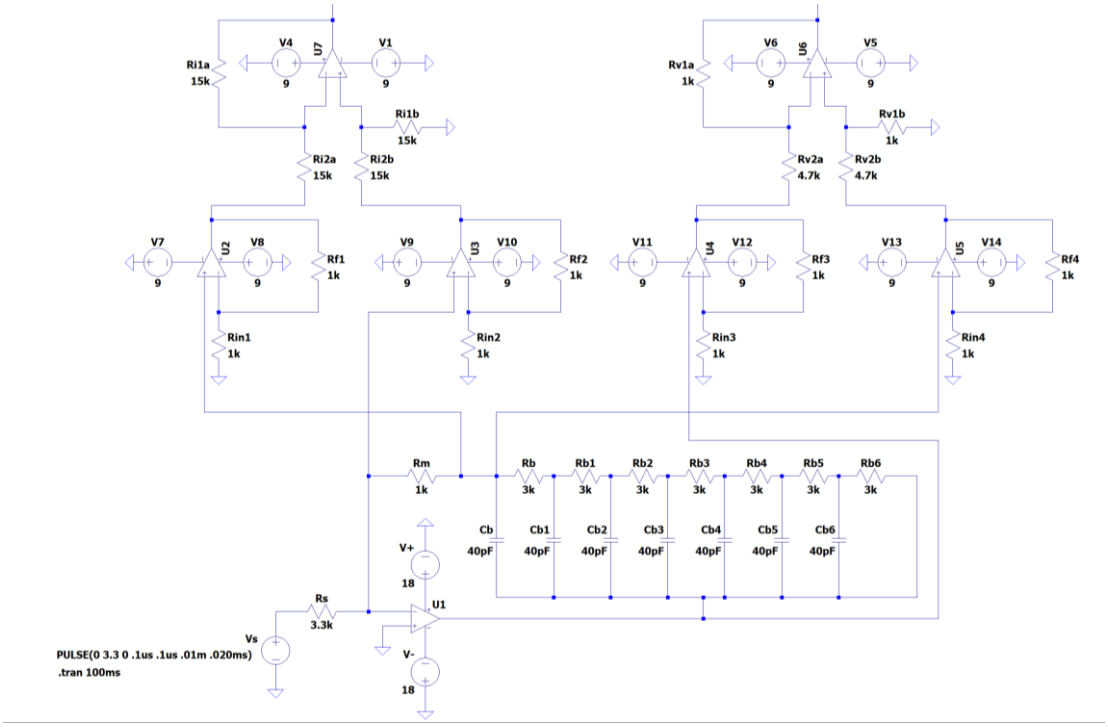

Figure-4: LTSpice Schematic with RC Ladder Network (SW)

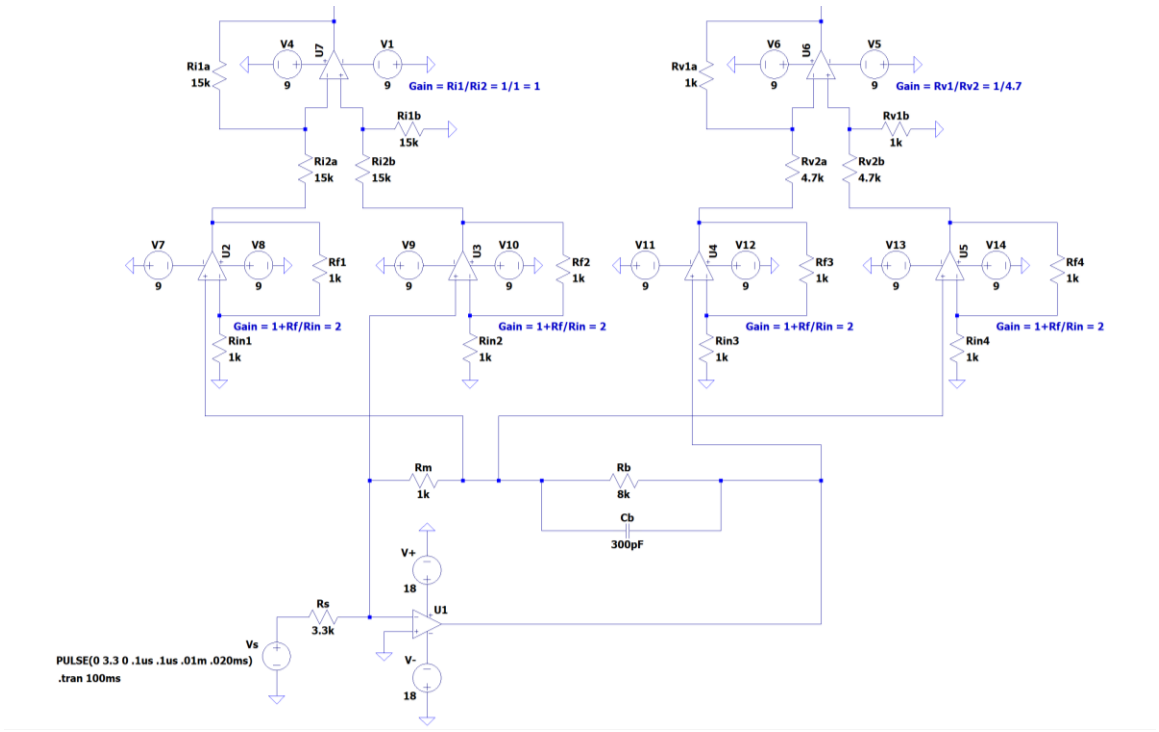

Figure-5: LTSpice Schematic with Simplified RC Network (SW)

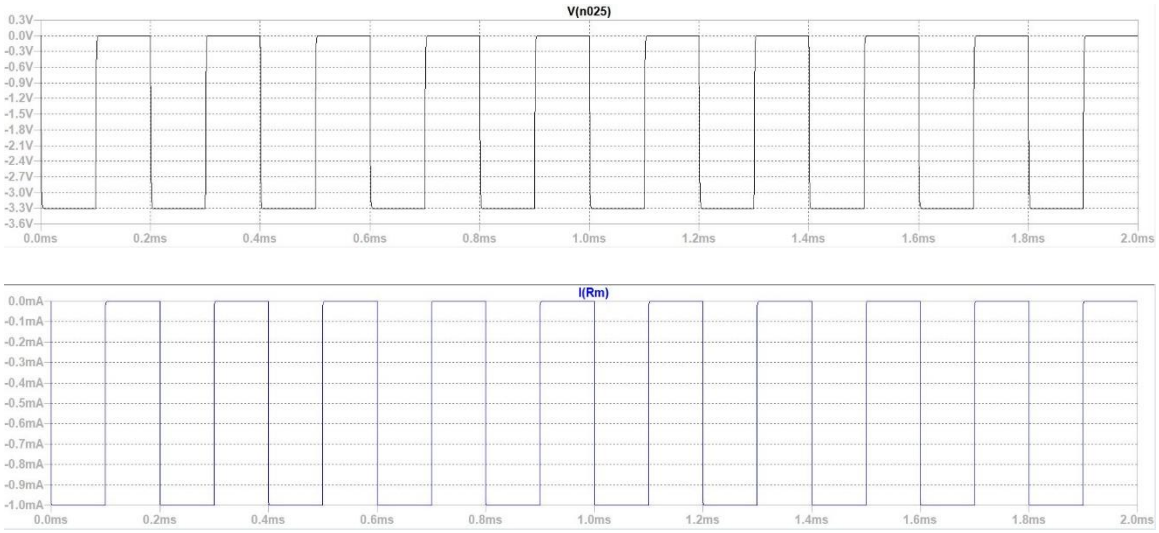

Figure-6: LTSpice Simulated Voltage and Current with Frequency at 50 Hz

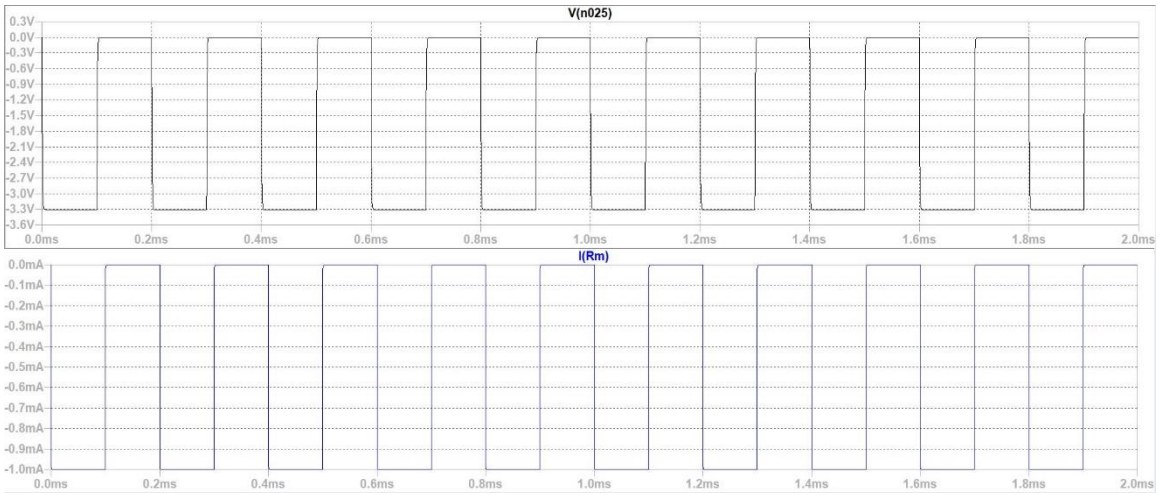

Figure-7: LTSpice Simulated Voltage and Current with Frequency at 500 Hz

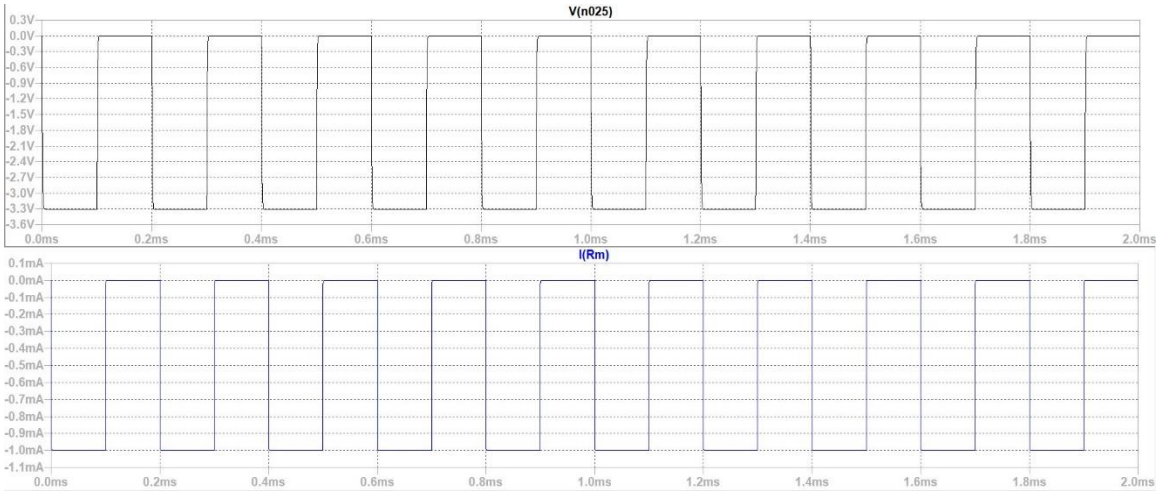

Figure-8: LTSpice Simulated Voltage and Current with Frequency at 5000 Hz

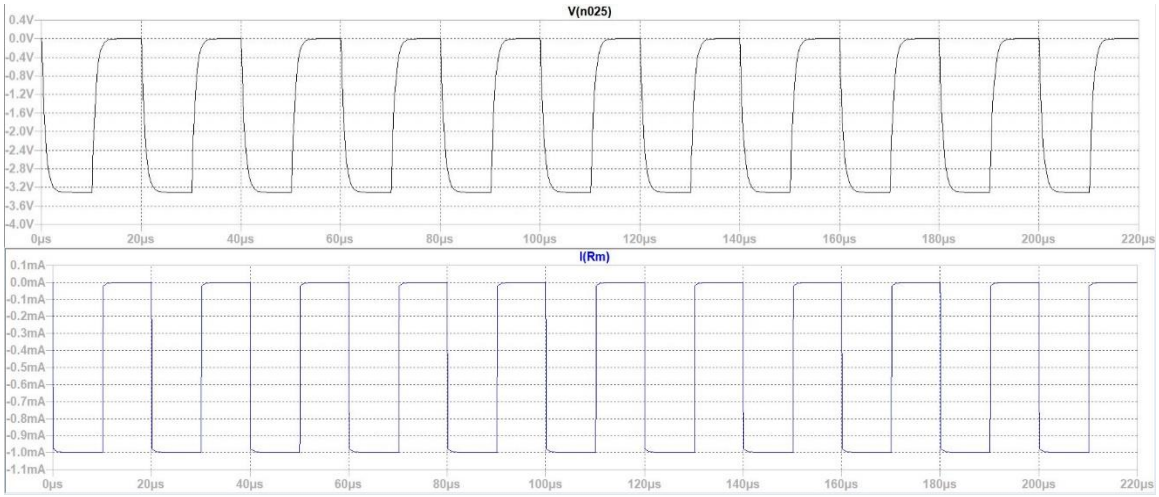

Figure-9: LTSpice Simulated Voltage and Current with Frequency at 50000 Hz

The spice simulations were the building blocks of the hardware design. Once we had a general idea of how the circuit was to be built, we then applied that knowledge to the LTSpice software in order to simulate the circuit. The resulting simulation waveforms above show the voltage and the current at each frequency. At each range of frequency, we were able to maintain a steady current right around 1mA which was the goal of the design as well as not to exceed the rail voltage of 10V. What we learned from these simulations are that the voltage rails are independent of the frequency. Setting an expected resistance of the body at 2-10k, we would push the voltage rails consistently, however, we maintained a steady current of 1mA. Comparing these results with the RC ladder network, we saw results were very similar. A major difference was that the time required to reach Steady-State was large, approximately 30ms or 1500 periods.

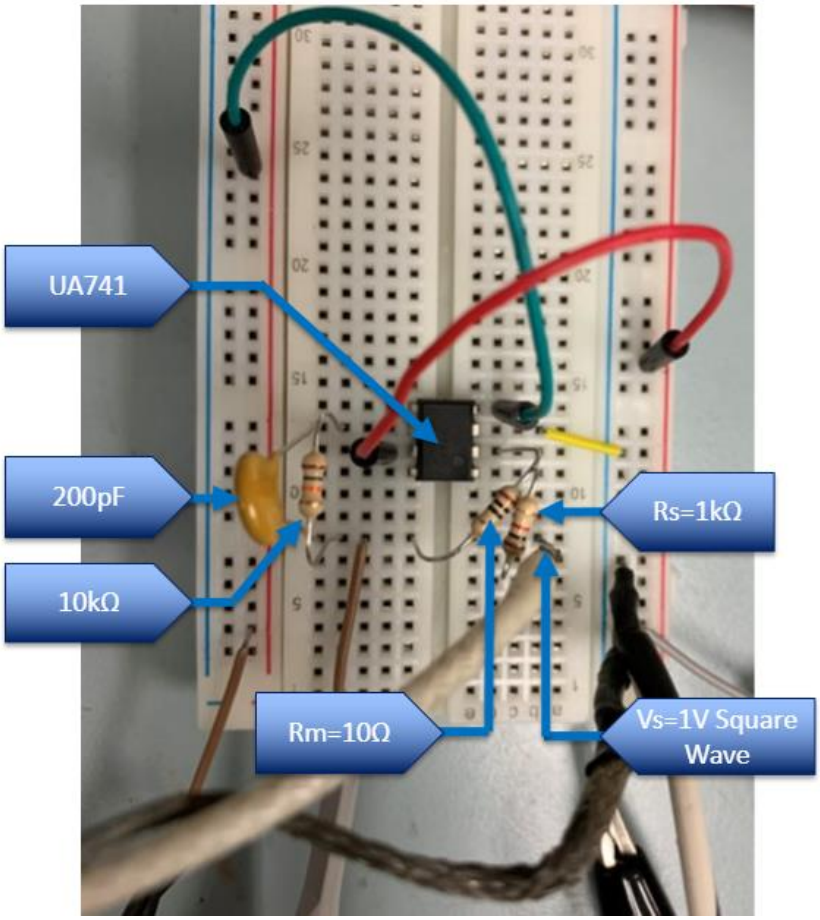

Figure-10: Preliminary Breadboarded Circuit

When designing the circuit on the board, we first started by using a Tektronix AFG3021B Signal/Function Generator, an Agilent Technologies MSO6012A Mixed Signal Oscilloscope and an Agilent #3631A Triple Output DC Power Supply. With the function generator, we were able to set up our 0V – 1V square wave and then vary the frequency from 50Hz – 50000 Hz. With the power supply, we were able to supply each rail of the operational amplifier with +/- 10V. Finally, with the oscilloscope, we were able to measure the voltage and current at different points in the circuit. The end design will include a battery supply consisting of two 9V batteries in series to replace the power supply.

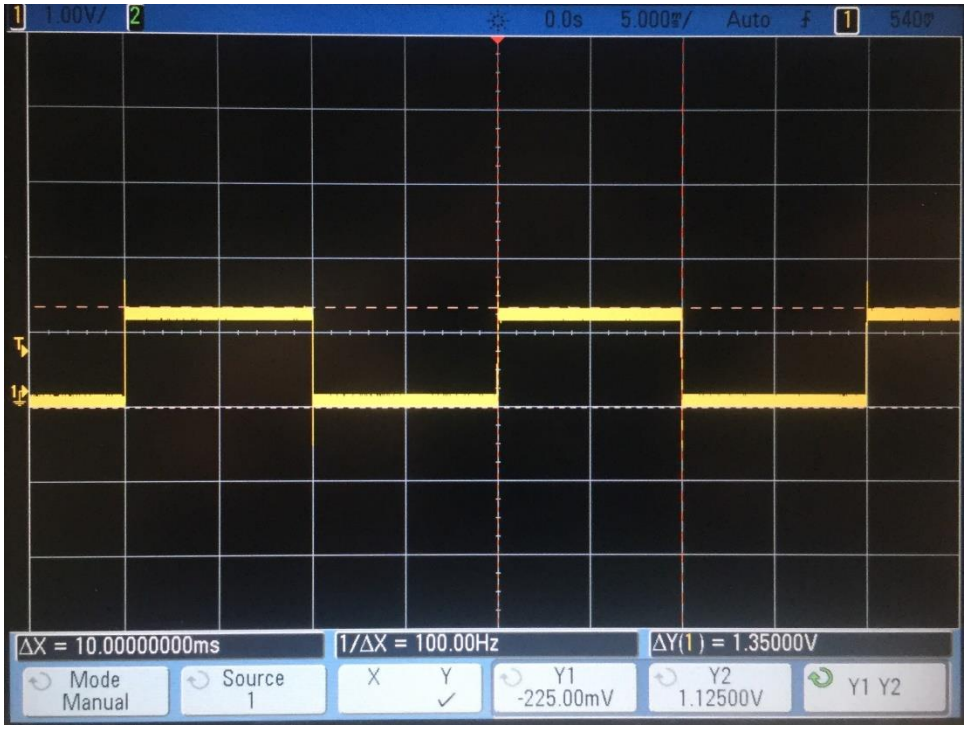

Figure-11: Oscilloscope Measurement at 50 Hz

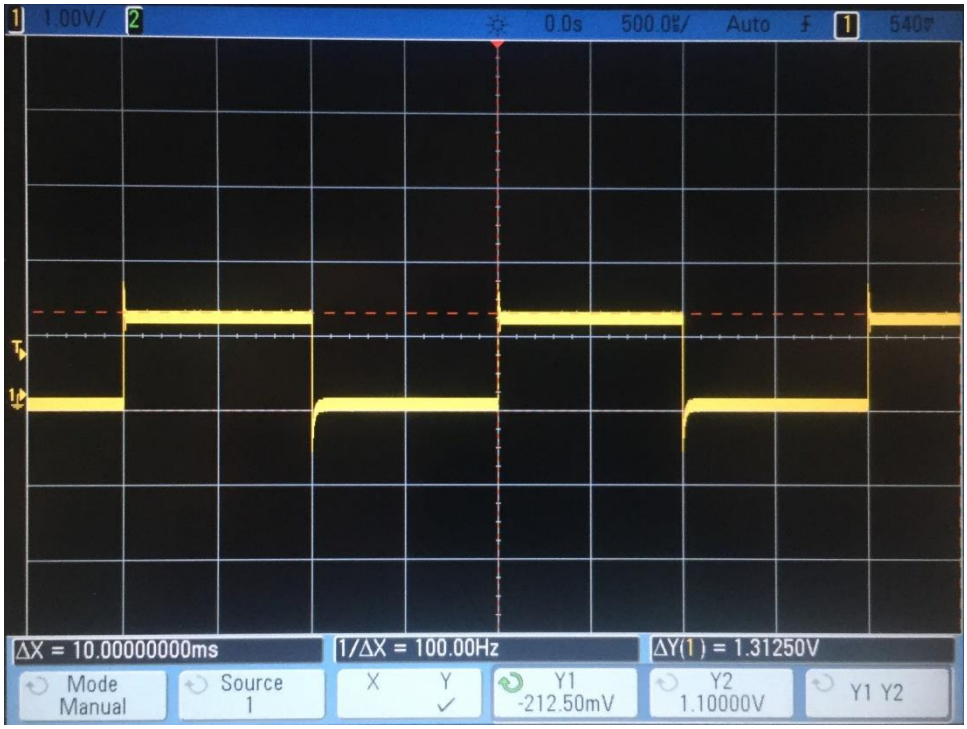

Figure-12: Oscilloscope Measurement at 500 Hz

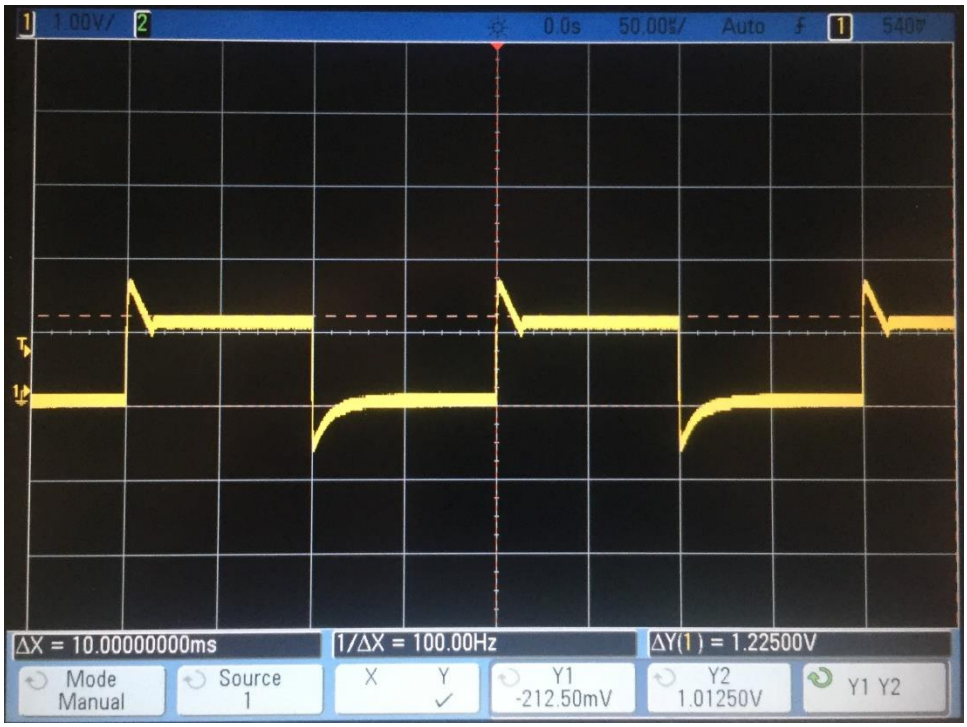

Figure-13: Oscilloscope Measurement at 5000 Hz

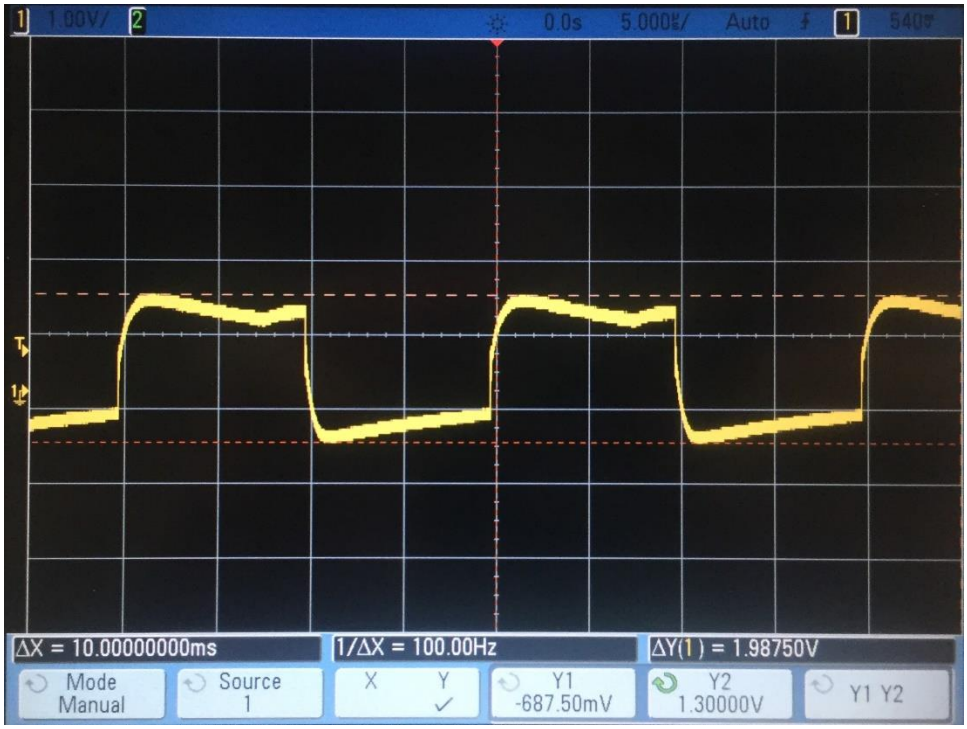

Figure-14: Oscilloscope Measurement at 50000 Hz

The resulting oscilloscope readings show the voltage differentials at each range of frequency. Maintaining a current of just below 1mA over each frequency, the voltage at

each frequency peaked at just under 2V. This differed from the LTSpice simulations as we had reached the rails consistently. Also, in LTSpice at frequencies of 5000Hz and 50000Hz we saw voltage waveforms that only peaked slightly whereas at the same frequencies on the board we saw increased peak durations that more resembled the square wave input. The probable reason for this is due to simulation in LTSpice with a LT1079, and then using a UA741 on the board for the inverting operational amplifier.

5.1.1 – Circuit Construction (SW)

Throughout the project, many different OpAmps were tried in the Inverting, Differential, and Buffer stages, with mixed results. Figure-15 illustrates the difficulties found when using a slow OpAmp like the UA741; the low slew-rate of 500mV/µs was unable to achieve a proper square-wave which drastically affected results at high resistances and low frequencies.

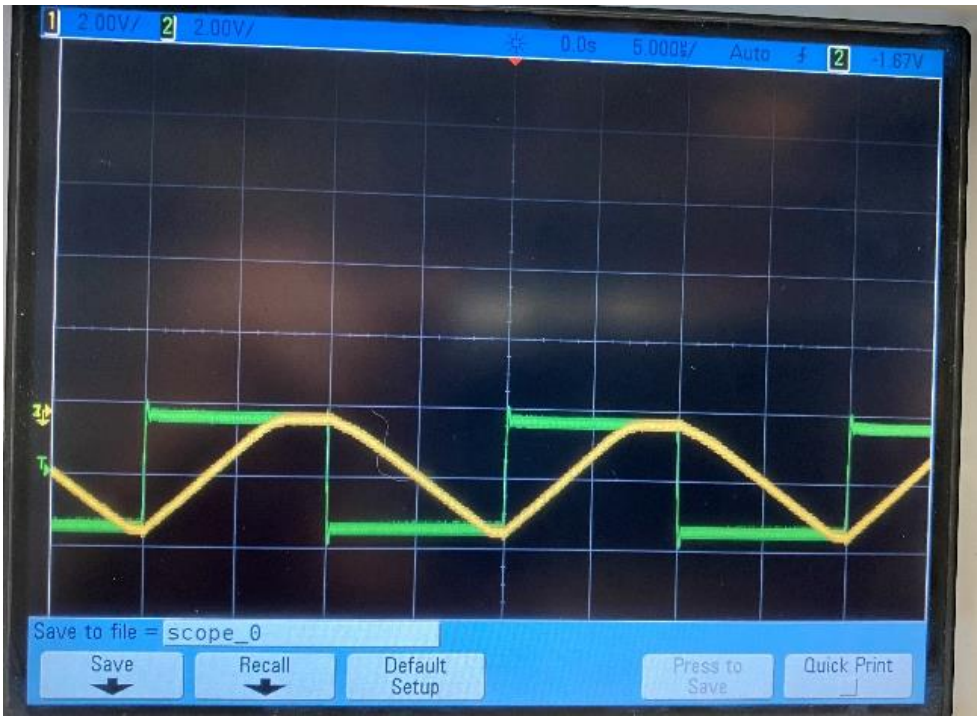

Figure-15: Inverting UA741 at 50Hz

In an effort to remedy the slew-rate issue, the UA741 was replaced with an LM318 which has a slew-rate of 50V/µs. Figures-16,17 show that – while the change fixed the slew rate issue – it introduced a considerable amount of noise, rendering the data useless. Both figures reflect the circuit buffer OpAmps in the differential stage.

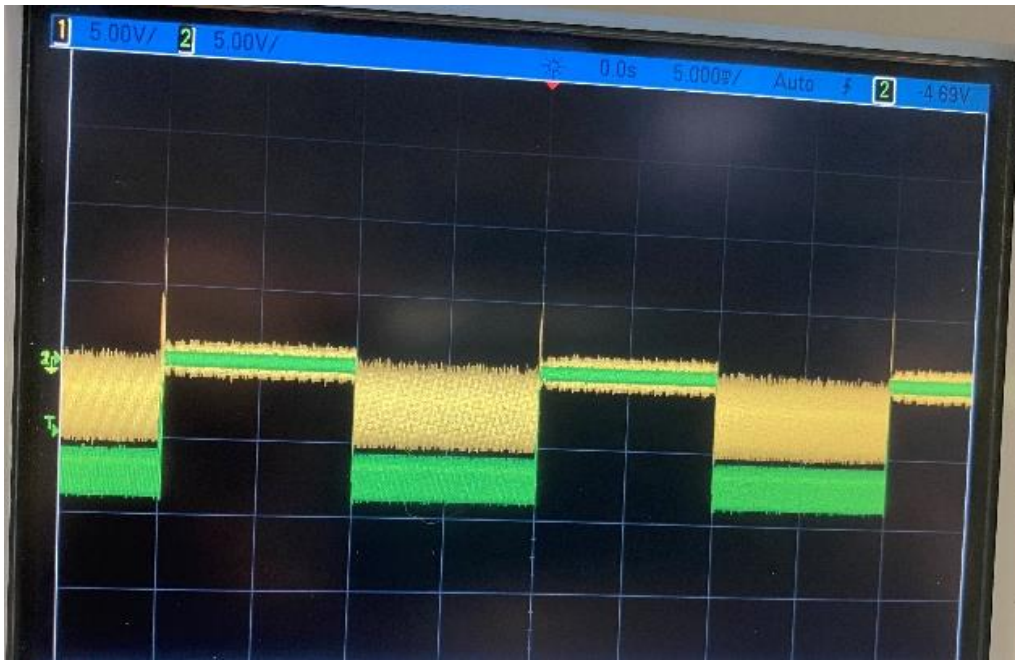

Figure-16: Inverting LM318 at 50Hz (with Buffers)

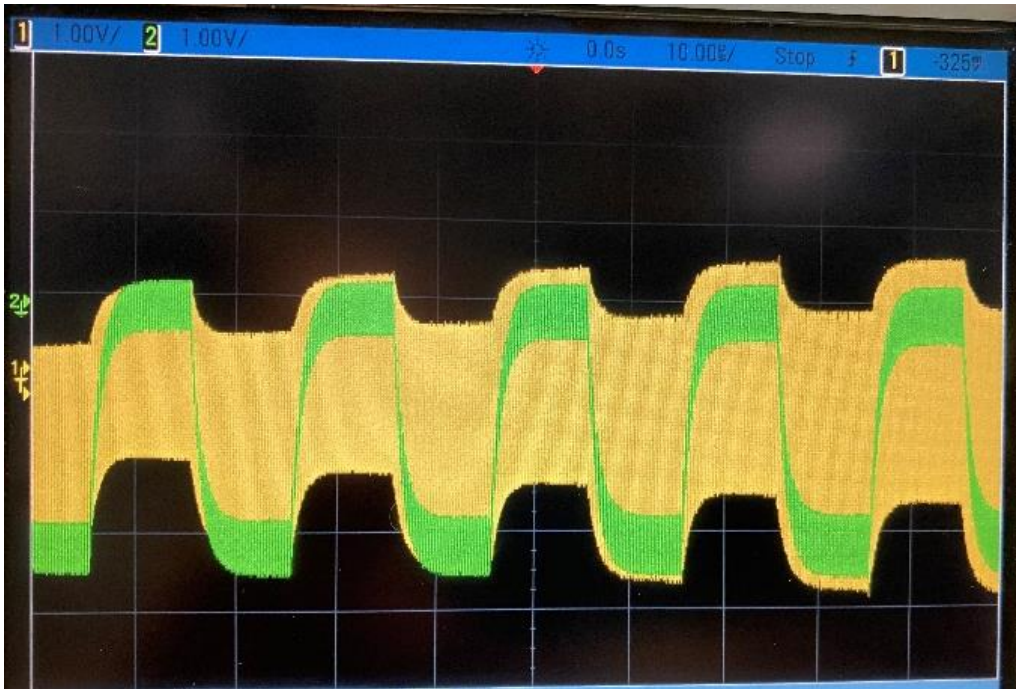

Figure-17: Inverting LM318 at 50kHz (with Buffers)

Figures-18,19 show the same circuit with the buffers removed, demonstrating that the buffers were exacerbating the noise issue and needed to be addressed.

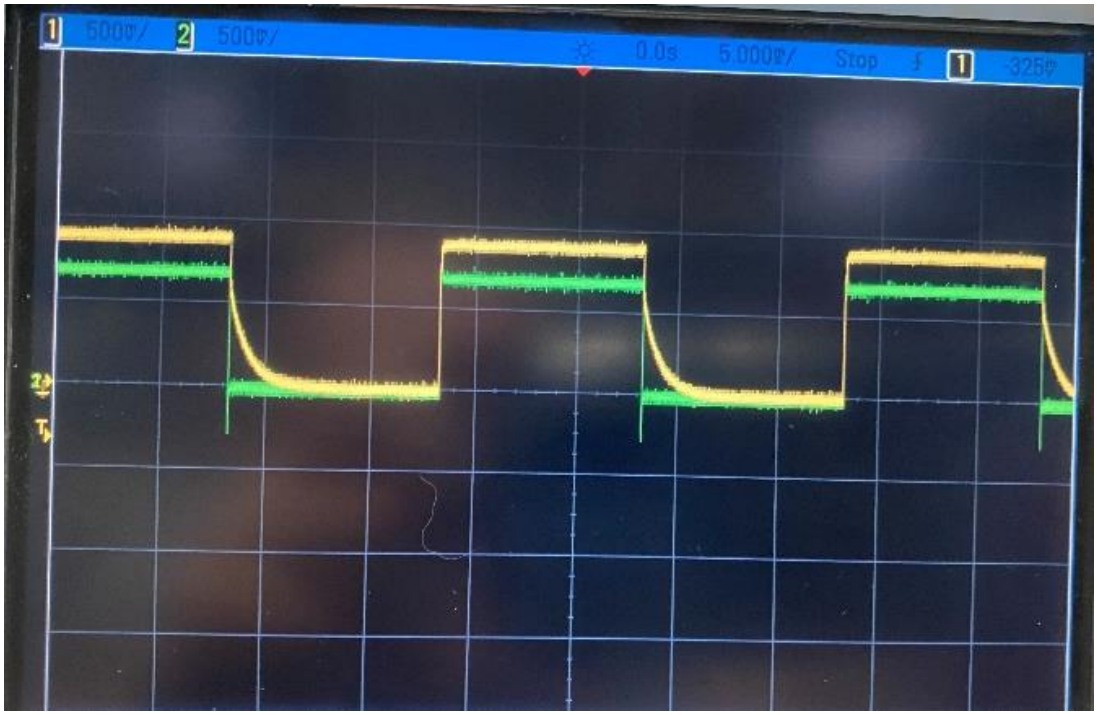

Figure-18: Inverting LM318 at 50Hz (without Buffers)

Figure-19: Inverting LM318 at 50kHz (without Buffers)

After performing the necessary voltage-attenuation to get within the embedded board's voltage range (below 3.3V), the noise – even without buffers – was still too significant to yield useful data. Figures-20,21 show the inverting output signal and the attenuated signal at the differential output.

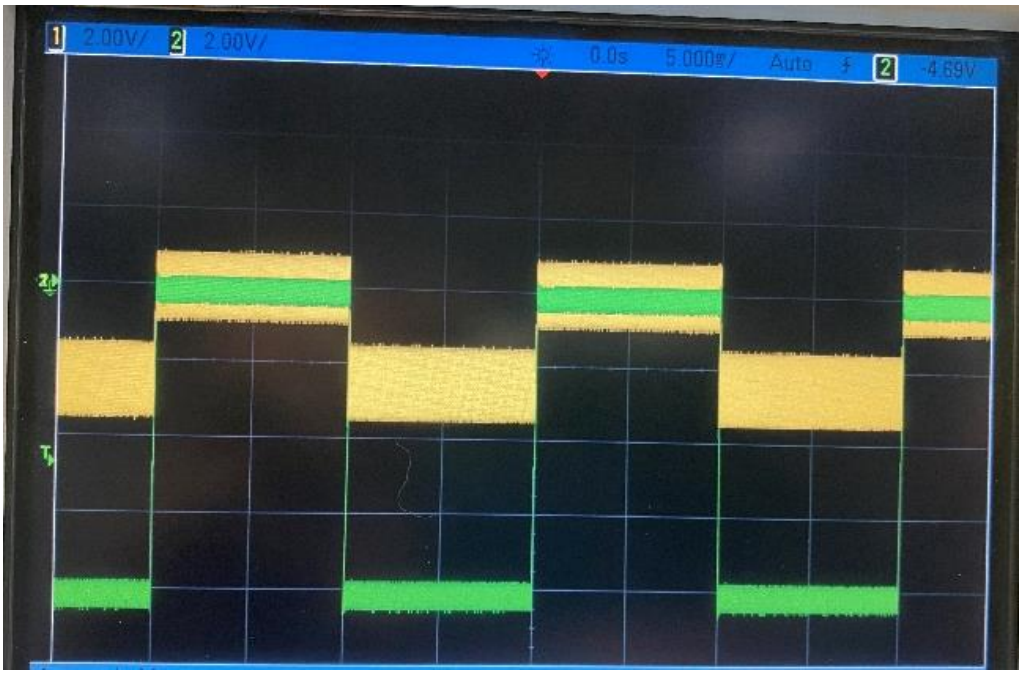

Figure-20: Inverting LM318 at 50Hz (Attenuated)

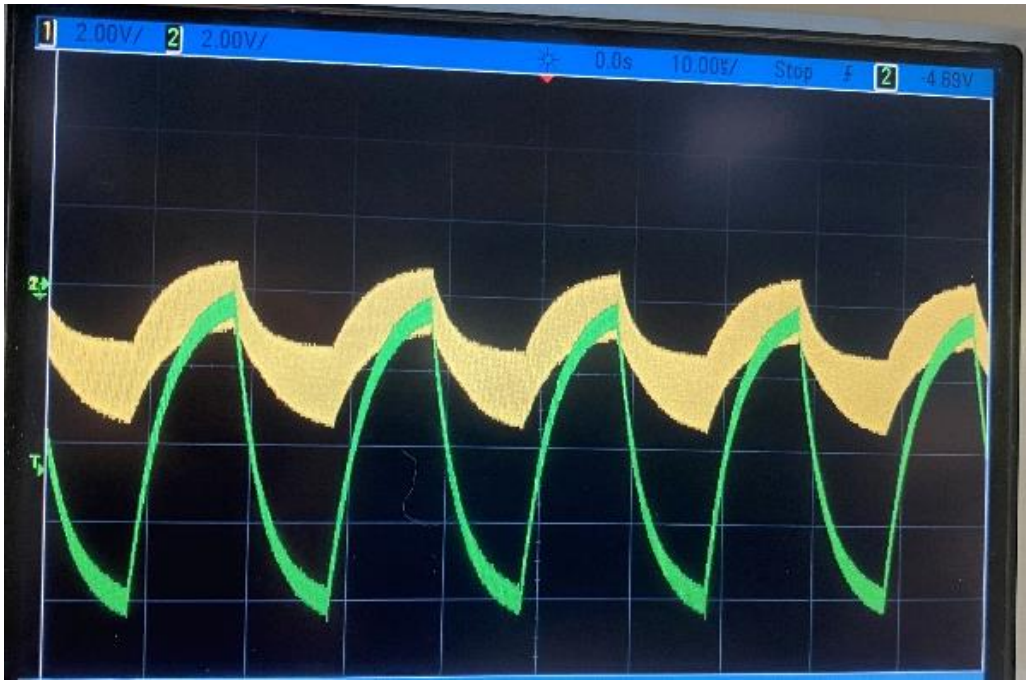

Figure-21: Inverting LM318 at 50kHz (Attenuated)

Figures-22,23 show the same circuit as above, but with a low-noise TLE2142 used in lieu of the previous LM318 Inverting OpAmp. These waveforms show the desired results, an attenuated circuit with neither slew-rate or noise issues. This demonstrates how important it was for the circuit to use high slew-rate, low-noise OpAmps.

Figure-22: Inverting TLE2142 at 50Hz (Attenuated)

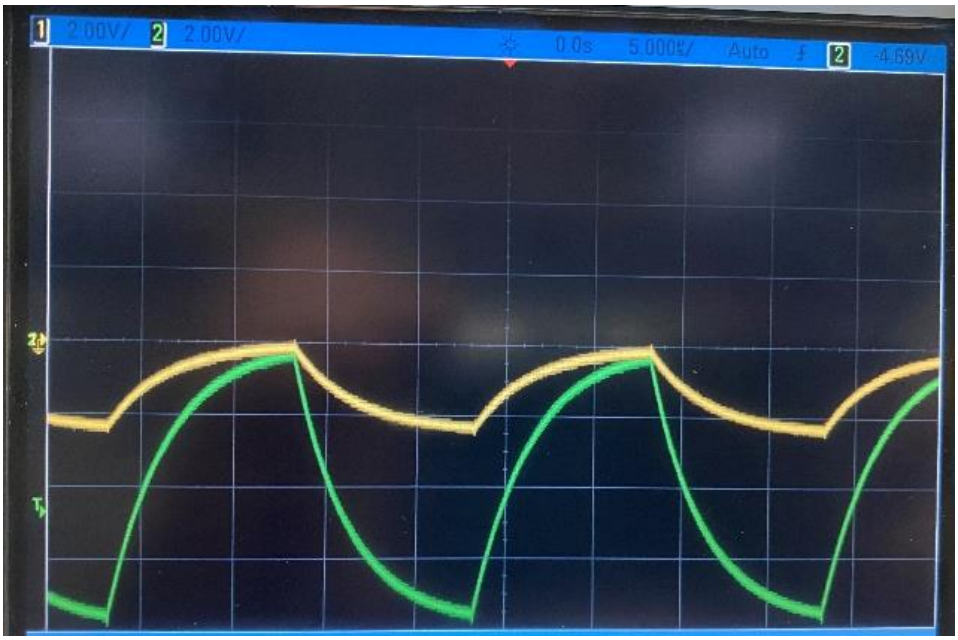

Figure-23: Inverting LM318 at 50kHz (Attenuated)

Since the same signal is passed through the buffer and differential stages, all were replaced with TLE2142 OpAmps to keep the data as useful as possible. Although this solved the Voltage Differential measurement noise, the Current Differential measurement was still noisy due to its low voltage level of 1V. To remedy this, the sense resistor Rm was increased from 10Ω to 1kΩ, which drastically-increased the signal-to-noise-ratio. Preliminary integration between the breadboarded circuit and embedded processor's sampling and calculations validated the circuit; impedance measurements were within 10% of expected values.

5.1.2 – Bench Setup (MS)

The following figures show the preliminary integration setup between the breadboarded circuit and the embedded processor, as well as the hardware setup including the power supplies used to power the TLE2142 op amps.

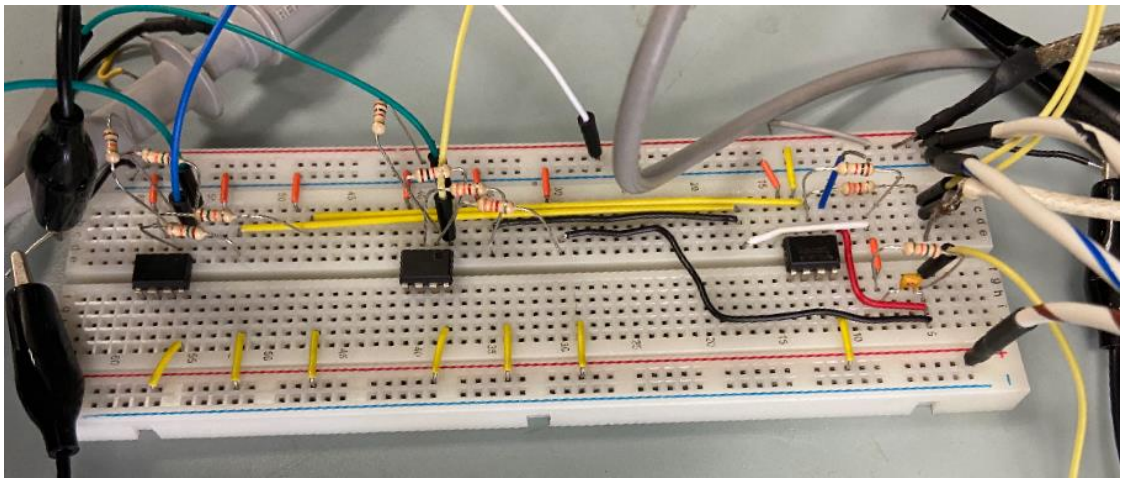

Figure-24: Breadboard Layout Without Buffers

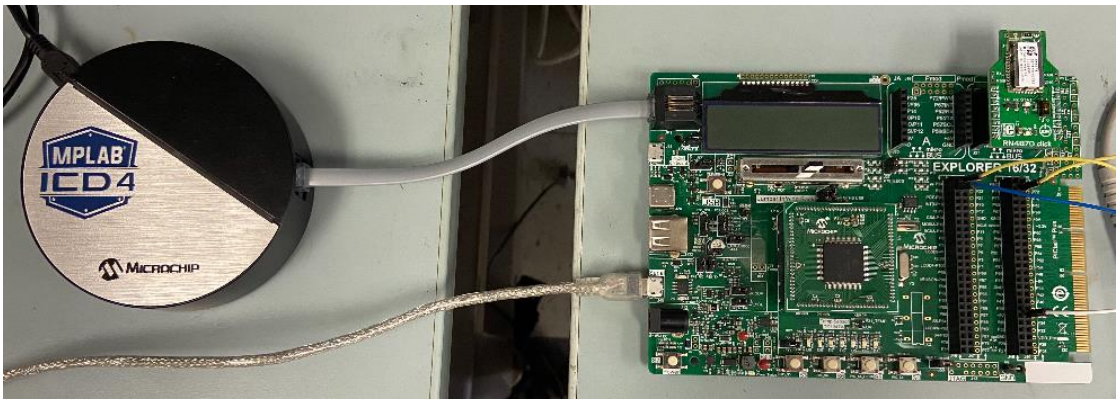

Figure-25: Embedded Processor Layout

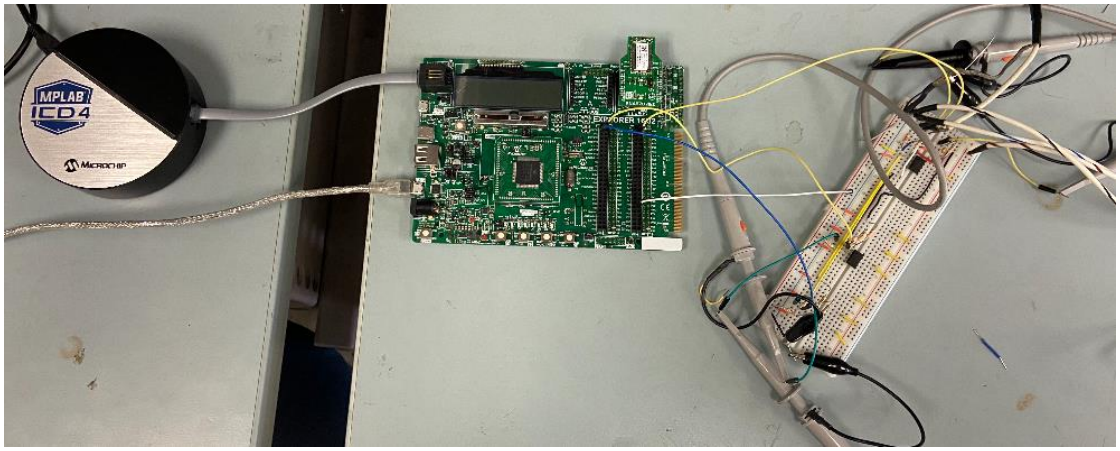

Figure-26: Hardware and Software Integration

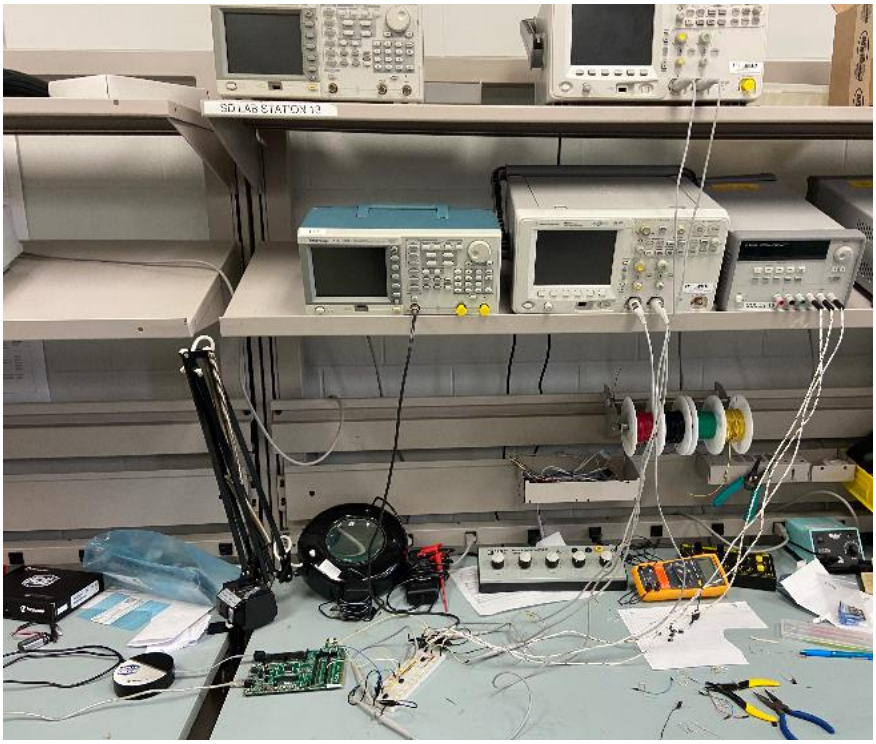

Figure-27: Bench Setup with Hardware and Software Integration

As stated above, removing the buffers was an act directed towards reducing all noise generated throughout the circuit. The theory was that once the components were soldered into the protoboard, most of, if not all noise generated would be removed and the buffers could be added back into the circuit. At this point in time, the hardware side was getting usable data which could be sent to the embedded processor, and thus integration between hardware and software could be introduced. Using a square wave generated from the embedded processor, data was sent to the embedded processor and impedance values were then calculated. The final design would have replaced the power supplies used with 9V batteries connected in series, and with the desired outputs from the circuit, the oscilloscopes would no longer need to be used. Although only early stages of integration were performed, data collected on the software end matched the waveforms obtained through simulation on the hardware end. As a result, impedance calculations were then able to be carried out and analyzed.

5.2 – Software Design (RB)

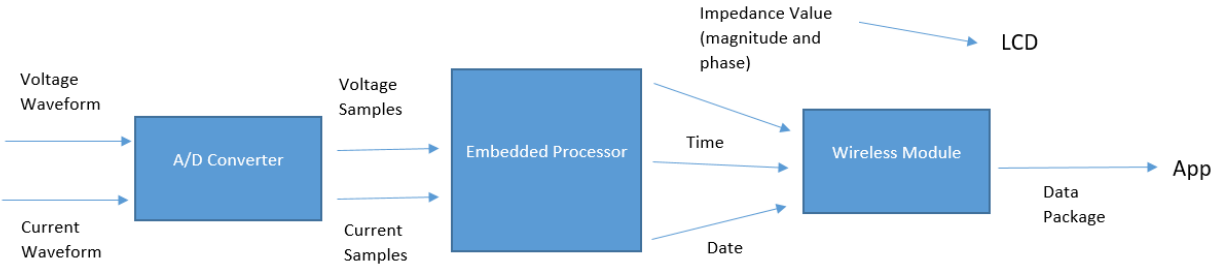

Figure-28: Embedded System Block Diagram Level 1

Table 4.1 – Functional Requirements: Analog/Digital Converter	
Module	**A/D Converter**
Designers	Ryan Byo
Inputs	Measured Vout and Iout
Outputs	Sampled Vout and Iout
Description	Sample the voltage and current waveform to later be processed

Table 4.2 – Functional Requirements: Embedded Processor	
Module	**Embedded Processor**
Designers	Ryan Byo
Inputs	Sampled Vout and Iout
Outputs	Impedance Value
Description	Use the sampled values to calculate the impedance. Also, generate control signals when needed

Table 4.3 – Functional Requirements: Wireless Module	
Module	**Wireless Module**
Designers	Ryan Byo
Inputs	Impedance Value, Time, Data
Outputs	Data package of inputs
Description	Transmit data to the mobile app/server

The embedded controller will be running code to do several different things. The below chart shows the control and data flow. The A/D converter, along with the DMA, direct memory access, unit, will need setup and then readings can begin. After the reading is done, we want to store that value as quickly as possible so the next sample can be taken. We will have to sample many times, so that process will repeat until we have all the data we need. Readings will continue to be taken until we have sufficient samples, and then the DMA buffer will be accessed, and the sampled values are read. Then, processing power will calculate the impedance so it can be displayed on board and sent away to be used by the app.

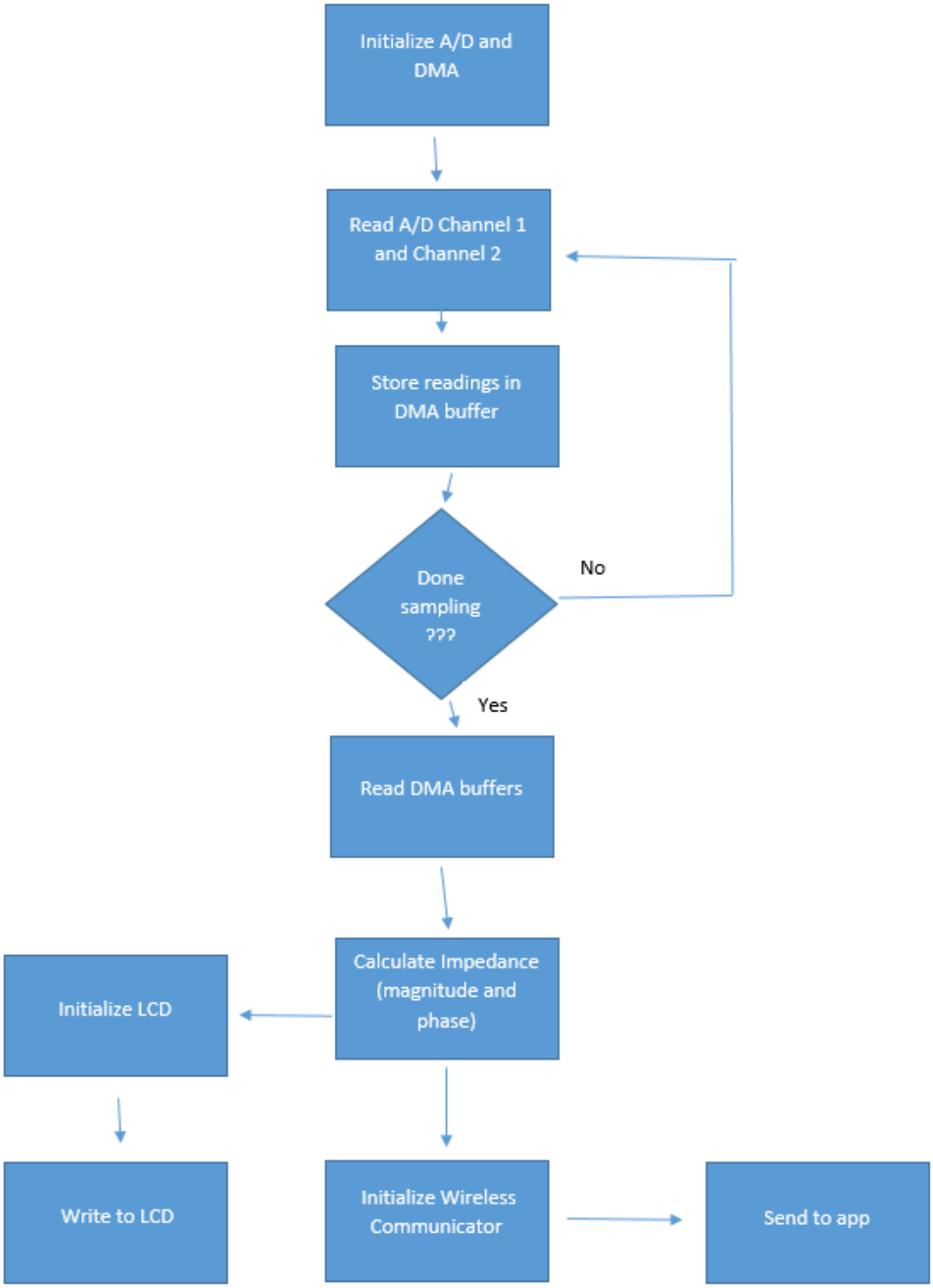

Figure-29: Embedded System Flow Chart (RB)

Subsystem Code (RB, KL)

A/D Converter, DMA, and LCD:

```
// Define Message Buffer Length for ECAN1/ECAN2
#define MAX_CHNUM 3       // Highest Analog input number in Channel Scan
#define SAMP_BUFF_SIZE 128 // Size of the input buffer per analog input
#define NUM_CHS2SCAN 2    // Number of (input pins) channels enabled for channel scan

#define SAMP_FREQUENCY 500000
#define MAX_SAMPLES_PER_PERIOD SAMP_FREQUENCY / 50

// Align the buffer to 512 bytes. This is needed for peripheral indirect mode
int BufferA[MAX_CHNUM + 1][SAMP_BUFF_SIZE] __attribute__((space(dma), aligned(1024)));
int BufferB[MAX_CHNUM + 1][SAMP_BUFF_SIZE] __attribute__((space(dma), aligned(1024)));
```

Figure-30: Useful Constants

We define a couple useful constants at the top of the file that will be used throughout the program.

```
19  struct Impedance
20  {
21    float real;
22    float imag;
23  };
```

Figure-31: Impedance Struct

The impedance struct provides a convenient way of storing the results of the impedance calculation. It consists of two floating point values, one for the real part and one for the imaginary part of the impedance.

```
62  void ms_delay(int N)
63  {
64    unsigned int delay = N; // N ms delay
65    int i;
66    int numLoops = N / 399;
67
68    for (i = 0; i <= numLoops; ++i) {
69      TMR1 = 0;         // reset TMR1
70      int tmr1Delay = delay % 399;
71      if (i < numLoops) {
72        tmr1Delay = 399;
73      }
74
75      tmr1Delay *= 164;
76
77      while (TMR1 < tmr1Delay); // wait for delay time
78    }
79  }
```

Figure-32: msDelay Function

This function is used to implement a millisecond delay in our code. During that time, the code is running NOPs. It is used throughout the code to allow initialization to occur, along with waiting an appropriate amount of time for data to be sent.

```
81  void initClockPLL()
82  {
83    // via the header.h file, the system is set to Primary Oscillator (XT) w/ PLL
84    // the crystal has a speed of 8 MHz and the system has a max speed of 40 MHz
85    // we will run the board at 40 MHz using PLL
86
87    CLKDIVbits.PLLPOST = 0; //Set the postscaler N2 = 2
88    CLKDIVbits.PLLPRE = 0;  //Set the prescaler N1 = 2
89    PLLFBDbits.PLLDIV = 40; //Set the multipler M = 40;
90
91    //This gives us Fosc = Fin*M/(N1*N2) = 8*40/(2*2) = 80 MHz
92    //Then Fcy = Fosc/2 = 40 Mhz
93  }
```

Figure-33: A Function to Setup the Embedded Clock Speed

This function modifies the PLL (phase-lock loop) module. The PPL allows us to modify the base clock speed of our processor to a custom value. In our case, we bumped our 8 MHz clock to run at 40 MHz, which is the fastest it can run, which is ideal for fast sampling.

```
95  void initADC()
96  {
97    AD1CON1bits.ADDMABM = 0; // DMA buffers are built in scatter/gather mode
98    AD1CON1bits.FORM = 0;    // Data Output Format: unsigned integer (0-1023)
99    AD1CON1bits.SSRC = 2;    // Sample Clock Source: GP Timer starts conversion
100   AD1CON1bits.ASAM = 1;    // ADC Sample Control: Sampling begins immediately after conversion
101   AD1CON1bits.AD12B = 0;   // 10-bit ADC operation
102   AD1CON1bits.SIMSAM = 1; // sample simultaneously
103
104   AD1CON2bits.CSCNA = 1; // Scan Input Selections for CH0+ during Sample A bit
105   AD1CON2bits.CHPS = 1; // Converts CH0 and CH1
106   AD1CON2bits.SMPI = 0; // 2 ADC Channel is scanned
107
108   AD1CON3bits.ADRC = 0; // ADC Clock is derived from Systems Clock
109   AD1CON3bits.ADCS = 1; // ADC Conversion Clock Tad=Tcy*(ADCS+1)= (1/40M)*2
110                         // ADC Conversion Time for 10-bit Tc=12*Tab
111   AD1CON3bits.SAMC = 1;
112
113   AD1CON4bits.DMABL = 7; // Each buffer contains 128 words
114
115   //AD1CSSH/AD1CSSL: A/D Input Scan Selection Register
116   AD1CSSH = 0x0000;
117   AD1CSSLbits.CSS0 = 1; // Enable AN0 for channel scan
118   AD1CSSLbits.CSS3 = 1; // Enable AN3 for channel scan
```

Figure-34: Part 1 of Code to Initialize the ADC

```
124     //AD1CHS0: A/D Input Select Register
125     AD1CHS0bits.CH0SA = 0; // MUXA +ve input selection (AIN0) for CH0
126     AD1CHS0bits.CH0NA = 0; // MUXA -ve input selection (VREF-) for CH0
127     //AD1CHS123: A/D Input Select Register
128     AD1CHS123bits.CH123SA = 1; // MUXA +ve input selection (AIN3) for CH1
129     AD1CHS123bits.CH123NA = 0; // MUXA -ve input selection (VREF-) for CH1
130
131     //AD1PCFGH/AD1PCFGL: Port Configuration Register
132     AD1PCFGH = 0xFFFF;
133     AD1PCFGL = 0xFFFF;
134     AD1PCFGLbits.PCFG0 = 0; // AN0 is analog
135     AD1PCFGLbits.PCFG3 = 0; // AN3 is analog
136
137     IFS0bits.AD1IF = 0;   // Clear the A/D interrupt flag bit
138     IEC0bits.AD1IE = 0;   // Do Not Enable A/D interrupt
139     AD1CON1bits.ADON = 1; // Turn on AD
140   }
141
```

Figure-35: Part 2 of Code to Initialize the ADC

This function configures the ADC module of the dsPIC33. Here, we set the configuration registers and setup the channels we will use. The important configuration set is we are using 10-bit mode, using CH0 and CH1, and using the DMA buffer (from the beginning index) to store the data and an interrupt triggers the end of the sampling. We also configure AN0 and AN3 pins as analog so we can use those as our input pins.

```
154   void initDMA()
155   {
156     DMA0CONbits.AMODE = 2; // Configure DMA for Peripheral indirect mode
157     DMA0CONbits.MODE = 2;  // continuous ping pong
158     DMA0PAD = (int)&ADC1BUF0;
159     DMA0CNT = (SAMP_BUFF_SIZE * NUM_CHS2SCAN) - 1;
160     DMA0REQ = 13; // Select ADC1 as DMA Request source
161
162     DMA0STA = __builtin_dmaoffset(BufferA);
163     DMA0STB = __builtin_dmaoffset(BufferB);
164
165     IFS0bits.DMA0IF = 0; //Clear the DMA interrupt flag bit
166     IEC0bits.DMA0IE = 1; //Set the DMA interrupt enable bit
167   }
```

Figure-36: Code to Initialize DMA

This function configures the DMA unit of the dsPIC33. We setup the DMA to use Channel 1 in continuous mode, meaning that the DMA will continue to be active the whole time we write to it, and disable ping-pong since we will only be writing to one buffer. We also tell the DMA that its samples will be coming from the ADC. Finally, we enable the DMA interrupt to tell us when sampling is done, and we can calculate a value.

```
142  void startSamp()
143  {
144    AD1CON1bits.SAMP = 1; // start sampling
145    DMA0CONbits.CHEN = 1; // Enable DMA
146  }
147
148  void stopSamp()
149  {
150    AD1CON1bits.SAMP = 0; // stop sampling
151    DMA0CONbits.CHEN = 0; // disable DMA
152  }
```

Figure-37: Start and Stop Sampling

These two functions allow us to start and stop the sampling of our input signals by turning the ADC off and disabling the DMA.

```
169  void initTimer3(double frequency)
170  {
171    TMR3 = 0x0000;
172
173    double scale = 2.0;
174    PR3 = (1.0 / frequency) / (scale * (1.0 / FCY));
175
176    IFS0bits.T3IF = 0; // Clear Timer 3 interrupt
177    IEC0bits.T3IE = 0; // Disable Timer 3 interrupt
178    T3CONbits.TON = 1; // Turn Timer 3 on
179  }
```

Figure-38: Initialization of Timer 3

This function initializes timer 3 to run at the desired frequency. This will ultimately be used to time the ADC and DMA controllers to sample at the desired speed. It also disables the timer interrupts because they will not be needed.

```
233  void __attribute__((interrupt, no_auto_psv)) _DMA0Interrupt(void)
234  {
235    int bufferOffset = dmaInterruptCount * SAMP_BUFF_SIZE;
236    int i;
237    if (pingPongState == 0)
238    {
239      for (i = 0; i < SAMP_BUFF_SIZE; ++i)
240      {
241        vn[bufferOffset + i] = BufferA[0][i];
242        in[bufferOffset + i] = BufferA[3][i];
243      }
244    }
245    else
246    {
247      for (i = 0; i < SAMP_BUFF_SIZE; ++i)
248      {
249        vn[bufferOffset + i] = BufferB[0][i];
250        in[bufferOffset + i] = BufferB[3][i];
251      }
252    }
253
254    pingPongState ^= 1;
255
256    dmaInterruptCount++; // increment DMA interrupt counter
257
258    IFS0bits.DMA0IF = 0; // Clear the DMA0 Interrupt Flag
259  }
```

Figure-39: DMA Interrupt Controller

The DMA interrupt function is where the raw data from the AD conversion gets processed. This function is set to trigger after 64 samples on each channel have been taken. The samples are then converted from 10bit integers to floating point values representing the actual voltage that was read. The converted values are then stored in temporary buffers and passed to the “calculateImpedance” function shown in figure-41. Finally, the last step is to clear the DMF interrupt flag to prevent the interrupt from immediately being trigger again.

```
181  void initPWM(double frequency)
182  {
183    T2CON = 0x0000;
184    PR2 = ((FCY / frequency) - 1) / 2; // divide by 2 since we need 2 toggles for a full period
185
186    OC1CONbits.OCSIDL = 0;
187    OC1CONbits.OCFLT = 0; // PWM fault detection, not used unless OCM=0b111
188    OC1CONbits.OCTSEL = 0; // use timer 2
189    OC1CONbits.OCM = 3; // toggle mode, i.e. square wave
190
191    OC1R = 0;
192
193    T2CONbits.TON = 1; // turn timer 2 on
194  }
195
196  void stopPWM()
197  {
198    T2CONbits.TON = 0; // turn timer 2 off
199  }
```

Figure-40: Initialize the PWM Module

This function sets the OC1 module to create a periodic square wave at the value of the passed in frequency. This will be the input signal that ultimately gets applied over the body.

```
266  struct Impedance calcImpedance(int N)
267  {
268    double vReal = 0.0;
269    double vImag = 0.0;
270    double iReal = 0.0;
271    double iImag = 0.0;
272    double k = 1.0;
273    int n;
274
275    for (n = 0; n < N; ++n)
276    {
277      double t = 2 * M_PI * k * ((double)n / N);
278      double v = convertSample(vn[n]) * 3.3;
279      double i = convertSample(in[n]) / 1000.0;
280
281      vReal += v * cos(t);
282      vImag += v * -1 * sin(t);
283
284      iReal += i * cos(t);
285      iImag += i * -1 * sin(t);
286    }
287
288    struct Impedance imp;
289
290    double mag = sqrt(pow(vReal, 2.0) + pow(vImag, 2.0)) / sqrt(pow(iReal, 2.0) + pow(iImag, 2.0));
291    double phase = atan(vImag / vReal) - atan(iImag / iReal);
292
293    imp.real = mag * cos(phase);
294    imp.imag = mag * sin(phase);
295
296    return imp;
297  }
```

Figure-41: Impedance Calculation

This function calculates the impedance of the voltage and current array that are passed as arguments. The function uses the discrete Fourier transform to find the phasor current and voltage at the sample frequency. The magnitude and phase of the impedance are calculated from the voltage and current samples. The function then converts the magnitude and phase into a real and imaginary form, where it is then returned as a struct.

LCD Control:

```
/*
    For Explorer 16 board, here are the data and control signal definitions
    RS -> RB15
    E  -> RD4
    RW -> RD5
    DATA -> RE0 - RE7
*/

// Control signal data pins
#define  RW  LATDbits.LATD5       // LCD R/W signal
#define  RS  LATBbits.LATB15      // LCD RS signal
#define  E   LATDbits.LATD4       // LCD E signal

// Control signal pin direction
#define  RW_TRIS    TRISDbits.TRISD5
#define  RS_TRIS    TRISBbits.TRISB15
#define  E_TRIS     TRISDbits.TRISD4

// Data signals and pin direction
#define  DATA      LATE           // Port for LCD data
#define  DATAPORT  PORTE
#define  TRISDATA  TRISE          // I/O setup for data Port
```

Figure-42: LCD Control Pins

This code defines our control signals of the LCD. According to the Explorer 16/32 User Guide, the RS bit, which selects the register to either accept data or a command, is tied to RB15. The RW bit, which controls the read/write process, is tied to RD5. The enable bit is tied to RD4. The data pins are on the eight pins of E.

```
void measureImpedance()
{
  unsigned int freqs[3] = {50, 500, 5000, 50000};
  int i;
  struct Impedance impResults[4];

  unsigned char impedanceStr[19];

  for (i = 0; i < sizeof(freqs) / sizeof(freqs[0]); ++i)
  {
    clearDisplay();
    ms_delay(100);
    cursorHome();
    ms_delay(100);
    sprintf(impedanceStr, "Measure %uHz", freqs[i]);
    puts_lcd(impedanceStr, 13 + i);
    ms_delay(3000);

    initPWM(freqs[i]);
    ms_delay(500); // wait 0.5s to reach steady state

    int numSamples = SAMP_FREQUENCY / freqs[i];
    int numInterrupts = numSamples / SAMP_BUFF_SIZE;

    // account for integer division truncation
    if (numInterrupts == 0 || numSamples % SAMP_BUFF_SIZE != 0)
    {
      numInterrupts++;
    }

    startSamp();

    dmaInterruptCount = 0;
    while (dmaInterruptCount < numInterrupts);

    stopSamp();
    stopPWM();

    impResults[i] = calcImpedance(numSamples);

    clearDisplay();
    cursorHome();
    ms_delay(100);
    sprintf(impedanceStr, "Re. %+.2e", impResults[i].real);
    puts_lcd(impedanceStr, 13);
    bottomLine();
    sprintf(impedanceStr, "Im. %+.2e",impResults[i].imag);
    puts_lcd(impedanceStr, 13);
    ms_delay(5000);
  }

  // send final results over bluetooth
}
```

Figure-43: Measure Impedance

This function orchestrates all the different subsystems to measure the impedance of the given load. It first defines the 4 different frequency ranges that will be measured, which are 50, 500, 5k and 50k Hz. For each frequency range, a message is output to the LCD telling the user which range is being currently measured. A square wave at the given frequency is output and AD samples are taken for both the input voltage across the body and the current going through the body. The sample rate is kept constant and is set to the max rate that could be safely achieved using two simultaneous channels. The number of samples used to make the impedance calculation is determined by the current input frequency. After the impedance is calculated, it is displayed on the LCD and sent over Bluetooth to the mobile device.

```
96   void lcd_cmd( char cmd )          // subroutiune for lcd commands
97   {
98       DATA &= 0xFF00;               // prepare RD0 - RD7
99       DATA |= cmd;                  // command byte to lcd
100      RW = 0;                       // ensure RW is 0
101      RS = 0;
102      E = 1;
103      ms_delay(50);                   // toggle E line
104      //NOP
105      E = 0;
106      ms_delay(5);         // 5ms delay
107  }
108
109
110  void lcd_data( char data )        // subroutine for lcd data
111  {
112      RW = 0;                      // ensure RW is 0
113      RS = 1;                       // assert register select to 1
114      DATA &= 0xFF00;               // prepare RD0 - RD7
115      DATA |= data;                 // data byte to lcd
116      E = 1;
117      ms_delay(50);
118      //NOP
119      E = 0;                       // toggle E signal
120      RS = 0;                      // negate register select to 0
121      ms_delay(1000); // 1mS delay
122  }
```

Figure-44: Write Data and Commands to LCD Methods

This code allows us to write both commands and data to the LCD by setting the appropriate RS bit, enabling the line to send the information, and then delaying while the LCD is processing the data.

```
124  void puts_lcd( unsigned char *data, unsigned char count )
125  {
126      while ( count )
127      {
128          lcd_data( *data++ );
129          count --;
130      }
131  }
```

Figure-45: Write String to LCD Methods

This function using the above lcd_data method to write a string to the LCD by repeatedly calling the subroutine for each character.

```
454  int main(void)
455  {
456    initClockPLL();
457    T1CON = 0x8030;
458    T1CONbits.TCKPS = 3; // prescale period of 256
459    T1CONbits.TON = 1; // Turn Timer 1 on
460
461    initADC();
462    initDMA();
463    initTimer3(SAMP_FREQUENCY);
464    Init_LCD();
465
466    measureImpedance();
467
468    while (1)
469    {
470      // cursorHome();
471      // ms_delay(1000);
472      // puts_lcd("Imped = 10", 10);
473      // ms_delay(1000);
474      // clearDisplay();
475      // ms_delay(1000);
476    }
477
478    return 0;
479  }
```

Figure-46: Main Method

This function is what runs when the program starts. It begins by initializing all the needed modules, including PLL, Timers 1 and 3, ADC, DMA, and LCD. It then runs the measureImpedance() procedure to sample, convert, calculate, and display our impedance.

Bluetooth/UART:

```
27  void initU2(){
28  //U2MODEbits.UARTEN = 1; // UART2 is enabled -- moved to end
29  U2MODEbits.USIDL = 1; // stop in idle mode
30  U2MODEbits.IREN = 0; // encoder and decoder disabled
31  U2MODEbits.RTSMD = 1; // Simplex mode
32  U2MODEbits.UEN = 0; // enables TX and RX, rest controlled by latches
33  U2MODEbits.WAKE = 1; // wakeup enabled
34  U2MODEbits.LPBACK = 0; // disable loopback
35  U2MODEbits.ABAUD = 0; // disable auto-baud rate
36  U2MODEbits.URXINV = 0; // U2RX idle state = 1
37  U2MODEbits.BRGH = 1; // high-speed mode
38  U2MODEbits.PDSEL = 0; // 8-bit mode with no parity
39  U2MODEbits.STSEL = 0; // 1 stop bit
40
41  U2STAbits.UTXISEL1 = 0;
42  U2STAbits.UTXISEL0 = 0; // interrupt generated when any character is transferred to the Transmit Shift register
43  U2STAbits.UTXINV = 0; // U2TX idle state = 1
44  U2STAbits.UTXBRK = 0; // Sync Break disabled
45  U2STAbits.URXISEL = 0; // interrupt generated when a character is received
46
47  U2BRG = 90; // sets baud rate = Fcy/4(U2BRG + 1) = 40M/(4*91) ~= 115200
48
49  U2MODEbits.UARTEN = 1; // UART2 is enabled
50  U2STAbits.UTXEN = 1; // transmitter enabled - U2TX and U2RX enabled
51
52  }
```

Figure-47: Initialize the UART

This code initializes the UART module, in this case, U2. This allows us to send a byte of data at a time to the Bluetooth module so that data can be transferred. The most important configuration done here is setting the UART module to use 8-bits with a single stop bit and setting the baud rate to around 115200, which is the value the RN4870 runs at by default

```
54  void putU2(char c){
55      while (U2STAbits.UTXBF) ; //wait if transmission buffer full
56      U2TXREG = c;    // write value to transmit FIFO
57  }
58
59  char getU2(){
60      while (!U2STAbits.URXDA) ; // wait till receive buffer has data
61      return U2RXREG;    // return value of receiver FIFO
62  }
63
64  void __attribute__((__interrupt__)) _U2TXInterrupt(void)
65  {
66      IFS1bits.U2TXIF = 0; // clear TX interrupt flag
67  }
```

Figure-48: UART Send and Receive

This code allows us to listen to the UART line, as well as write to the UART line. This is used to send and receive messages via Bluetooth. These methods simply wait until there is room to send data and sends it or if data has been received, then returns it.

```
69  int main(void) {
70      initClockPLL();
71      IEC1bits.U2TXIE = 1; // Enable UART TX Interrupt
72      initU2();
73      ms_delay(1000); // wait 1 second -- need to wait min of (1/BaudRate)
74
75      char str[20];   // set aside array
76      int i;
77
78      sprintf(str, "TEST");
79      for(i = 0; i < (4); i++){
80          putU2(str[i]);  // write a single char
81          ms_delay(100);   //wait
82      }
83
84      while(1){
85      }
86
87      return 0;
88  }
```

Figure-49: Main Method

This code is to test the functionality of the UART and Bluetooth modules. It simply initializes the U2 module and the sends the string “TEST” character-by-character to the RN4870, which sends it to the connected devices, i.e. a phone, where the message “TEST” can be received and displayed.

Application Software – Level 1 (KL)

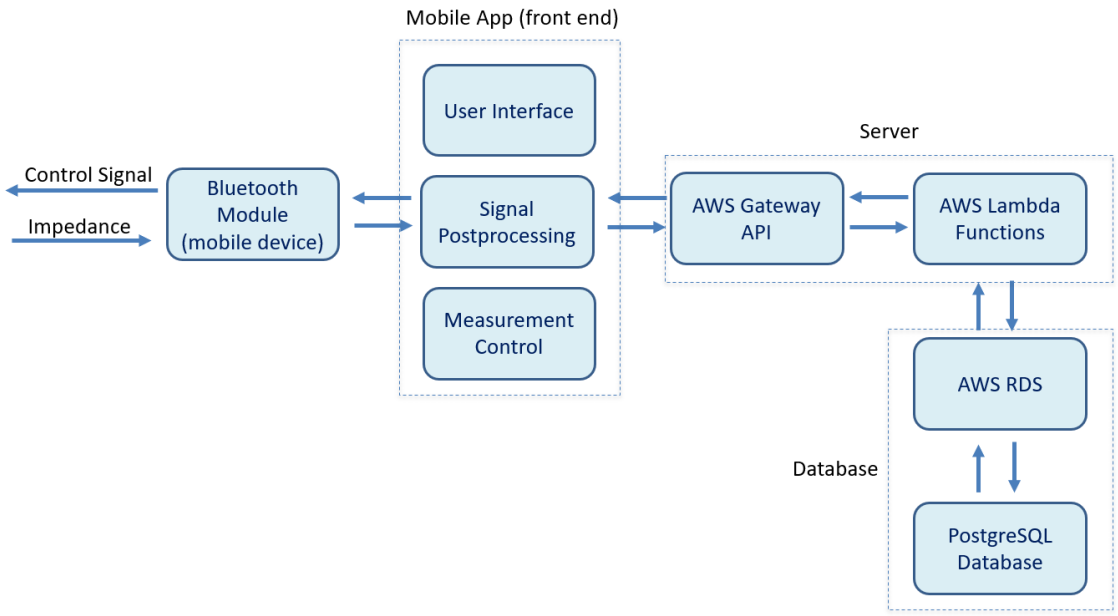

Figure-50: Application Software Block Diagram Level 1

Table 5.1 – Functional Requirements: Bluetooth Module (Phone)	
Module	**Bluetooth module (phone)**
Designers	Kevin Libertowski
Inputs	Impedance measurements
Outputs	Control signal
Description	This will allow the phone application to communicate with the embedded processor. It will be receiving impedance measurements from the embedded processor and sending signals to start the measurement.

Table 5.2 – Functional Requirements: Mobile App (Front End)	
Module	**Mobile App (front end)**
Designers	Kevin Libertowski
Inputs	Impedance measurements
Outputs	User and measurement data, control signal
Description	The mobile app front end will be running on the user's phone or tablet. It will display the measurements and provide an interface for saving the users info and measurements to the cloud. It will provide both current and past measurements displayed in text and graph form. This is also where the post processing will be done for calculating the user's hydration level.

Table 5.3 – Functional Requirements: Server	
Module	**Server**
Designers	Kevin Libertowski
Inputs	HTTP requests containing user/measurement data
Outputs	HTTP responses containing requested data
Description	The purpose of the server is to provide an interface for our mobile app to save and retrieve data from the database. The server will be setup using AWS Lambda using the Node.JS runtime. An AWS gateway API will be configured to bridge the mobile application with the Lambda functions.

Table 5.4 – Functional Requirements: Database	
Module	**Database**
Designers	Kevin Libertowski
Inputs	SQL queries from server
Outputs	Stored measurement and user data
Description	The database will be setup using Amazon RDS for PostgreSQL. The database will save measurement and account data for each individual user. It will allow all previous readings to be retrieved anytime from any mobile device with the user login credentials.

The application software block diagram shows the different components and data flow for the app. The application software will serve the main purpose of displaying the user's measurements in a user-friendly manner. It will also allow the user to easily view past results and see them plotted over a time period of their choice. The mobile app will also save data to the cloud where it will be able to be accessed across multiple devices.

To accomplish this, the application software will be distributed across several components. To start things off, the phone app will need access to the measurement data. This will happen by sending each measurement to the phone over Bluetooth. For this to work, there will need to be an initial pairing process between the embedded processor and the phone. Once this is done, the phone and embedded processor will be able to send and receive data when in range.

The mobile app will consist of all the code running on the phone or tablet itself. It will be built using Flutter, which is a modern cross platform mobile framework developed by Google. This will allow the source code to be compiled to run on both iOS and Android devices.

The mobile app will play a crucial role in the overall user experience. One of the primary functions will be displaying the measurement data to the user. This measurement will include the impedance value and the user's estimated hydration level. The measurement will be displayed automatically when a new measurement is recorded and received by the phone. The user will also be able to view a plot of, past measurements over a time period of their choice.

The server will provide an interface for the mobile app to communicate with the database. It will also handle authentication to ensure that a user can only access data they have permission to use. The server will be implemented using AWS Lambda, with a Node.js runtime. The API end points will be configured using the Amazon API Gateway. Each API endpoint will contain corresponding code and interacts with the database, whether it is adding or retrieving data.

Table 6.1 – User Database SQL Table

ID	Username	Password (encrypted)	DOB	Height	Weight	Sex

Table 6.2 – Measurement Database SQL Table

ID	Impedance (Re)	Impedance (Im)	Frequency	Date	ICW	ECW	User

The database will allow user data to be stored over time. The tables above describe the database scheme. The database will consist of two tables. The user table will

store data specific to each user such as username password and body statistics. The body statistics may be useful for analyzing the impedance. The other table is the measurement table. This has a many-to-one relationship with the user table, as a single user may have many measurements. This table will contain a row for every individual measurement taken. It will store the impedance, date taken, user it was associated with and eventually the approximated intra and extracellular water contents of the user.

Subsystem Code (RB, KL)

Mobile App (RB)

```
15  void main() {
16    runApp(MaterialApp(
17      title: 'App',
18      home: MyApp(),
19    ));
20  }
21
22  class MyApp extends StatefulWidget{
23    @override
24    _MyAppState createState() => _MyAppState();
25  }
```

Figure-51: Initiate the App

These two methods determine what happens when the app is launched. The main() methods is what is first run and it tells the phone to run MyApp(), which is what contains the layout and controls for everything in our app.

```
class _MyAppState extends State<MyApp> {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'BioImpedance Tracker',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: Scaffold(
        appBar: AppBar(
          title: Text('BioImpedance Tracker'),
        ),
        body: Column(children: <Widget>[
          Container(
            alignment: Alignment.center,
            padding: EdgeInsets.all(20.0),
            child: Text(
              'Bioimpedance Tracker App',
              textAlign: TextAlign.center,
              style: TextStyle(
                fontSize: 25.0,
              ),
            ),
          ),
          Container(
            child: Image.asset('assets/body.jpg'),
          ),
```

Figure-52: Part 1 of Login Page

```
            Container(
              padding: EdgeInsets.all(20.0),
              child: TextFormField(
                decoration: InputDecoration(
                  labelText: 'Username',
                ),
              ),
            ),
            Container(
                margin: EdgeInsets.all(50),
                child: RaisedButton(
                  onPressed: () {
                    Navigator.push(
                      context,
                      MaterialPageRoute(builder: (context) => NavigationPage()),
                    );
                  },
                  child: Text(
                    'Sign In',
                    style: TextStyle(
                      fontSize: 20.0,
                    ),
                  ),
                )),
          ]),
        ),
      );
  }
}
```

Figure-53: Part 2 of Login Page

The above two figures make up the first page of the app the users' will see. It has four main parts, each listed as a "container." The first is just a title, giving the name of the app. The second is a picture to describe the app. The third contains a textbox, labeled "username" for a user to access their account. Lastly is a raised button, which would send the information above to the server to verify the user and gather his/her information.

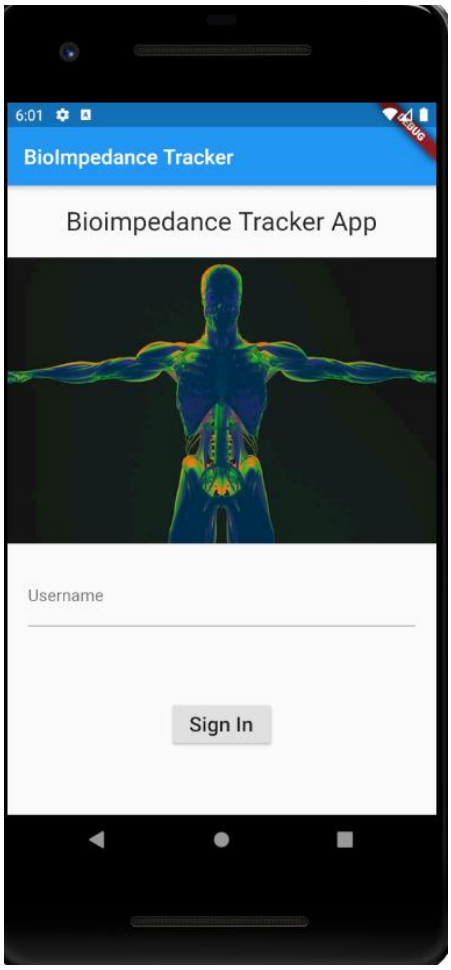

Figure-54: Login Page

The login page would look something like this.

```
class NavigationPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Navigation Home'),
        //leading: new Container(),
      ),
      body: Column(
        children: <Widget>[
          Container(
            padding: EdgeInsets.all(20.0),
            child: Row(
              children: <Widget>[
                Expanded(
                  flex: 3,
                  child: Text(
                    'View Readings',
                    style: TextStyle(
                      fontSize: 20.0,
                    ),
                  ),
                ),
                Expanded(
                    flex: 1,
                    child: RaisedButton(
                        onPressed: () {
                          Navigator.push(
                            context,
                            MaterialPageRoute(builder: (context) => Readings()),
                          );
                        },
```

Figure-55: Part 1 of Navigation Page

```
                    shape: StadiumBorder(),
                    child: Icon(
                      Icons.arrow_right,
                      size: 50.0,
                    ))),
              ],
            ),
          ),
          Container(
            padding: EdgeInsets.all(20.0),
            child: Row(
              children: <Widget>[
                Expanded(
                  flex: 3,
                  child: Text(
                    'View Graphs',
                    style: TextStyle(
                      fontSize: 20.0,
                    ),
                  ),
                ),
                Expanded(
                    flex: 1,
                    child: RaisedButton(
                        onPressed: () {
                          Navigator.push(
                            context,
                            MaterialPageRoute(builder: (context) => Graphs()),
                          );
                        },
```

Figure-56: Part 2 of Navigation Page

```
                    shape: StadiumBorder(),
                    child: Icon(
                      Icons.arrow_right,
                      size: 50.0,
                    ))),
              ],
            ),
          ),
          Container(
            padding: EdgeInsets.all(20.0),
            child: Row(
              children: <Widget>[
                Expanded(
                  flex: 3,
                  child: Text(
                    'Bluetooth Settings',
                    style: TextStyle(
                      fontSize: 20.0,
                    ),
                  ),
                ),
                Expanded(
                    flex: 1,
                    child: RaisedButton(
                        onPressed: () {
                          Navigator.push(
                            context,
                            MaterialPageRoute(builder: (context) => BluetoothSettings()),
                          );
                        },
```

Figure-57: Part 3 of Navigation Page

```
                        shape: StadiumBorder(),
                        child: Icon(
                          Icons.arrow_right,
                          size: 50.0,
                        ))),
                  ],
                ),
              ),
            ],
          ),
        );
      }
    }
```

Figure-58: Part 4 of Navigation Page

The above for figures make up the navigation page. This creates three titles and a corresponding button for each. Each button leads to a corresponding page, most of which are not final. All three of them have a similar appearance, so I will only show the one with the most progress.

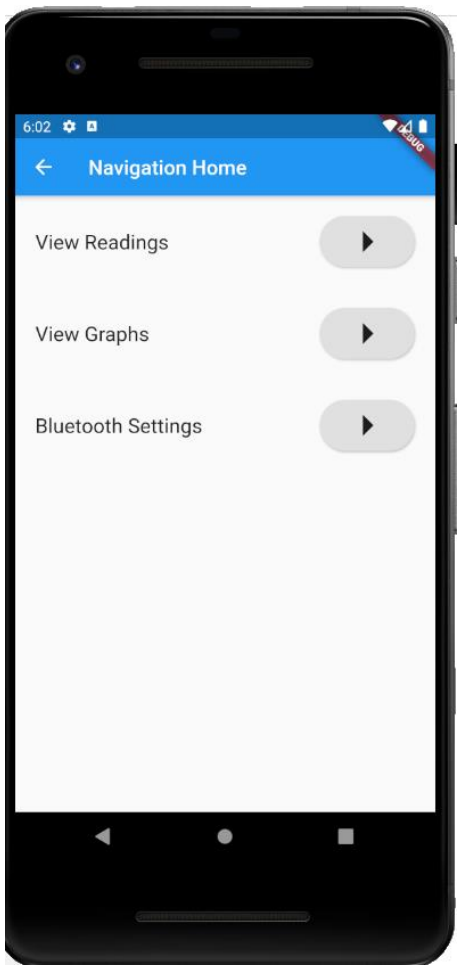

Figure-59: Navigation Page

The overall appearance of the navigation page. More buttons would be added in the future.

```
  static var series = [
    charts.Series<ImpedanceReading,int>(
      domainFn: (ImpedanceReading zData, _) => zData.day,
      measureFn: (ImpedanceReading zData, _) => zData.z,
      colorFn: (_, __) => charts.MaterialPalette.blue.shadeDefault,
      id: 'Impedance',
      data: data,
    ),
  ];

  static var chart = charts.LineChart(
    series,
    animate: true,
    behaviors: [
      new charts.ChartTitle('Day',
        behaviorPosition: charts.BehaviorPosition.bottom,
        titleOutsideJustification: charts.OutsideJustification.middleDrawArea),
    new charts.ChartTitle('Impedance',
        behaviorPosition: charts.BehaviorPosition.start,
        titleOutsideJustification: charts.OutsideJustification.middleDrawArea)
    ],
  );

  var chartWidget = Padding(
    padding: EdgeInsets.all(32.0),
    child: SizedBox(
      height: 200.0,
      child: chart,
    ),
  );
```

Figure-60: Part 1 of Graph Page

```
    @override
    Widget build(BuildContext context) {
      return Scaffold(
          appBar: AppBar(
            title: Text('Impedance Graphs'),
          ),
          body: Column(children: <Widget>[
            Text("Put graph here"),
            chartWidget,
          ],
          ));
    }
  }
```

Figure-61: Part 2 of Graph Page

The above two figures show the code for making the graph page. It consists of first making the graph widget (currently using test data). It them makes the user page, which has a title and the graph, and would eventually have some further information.

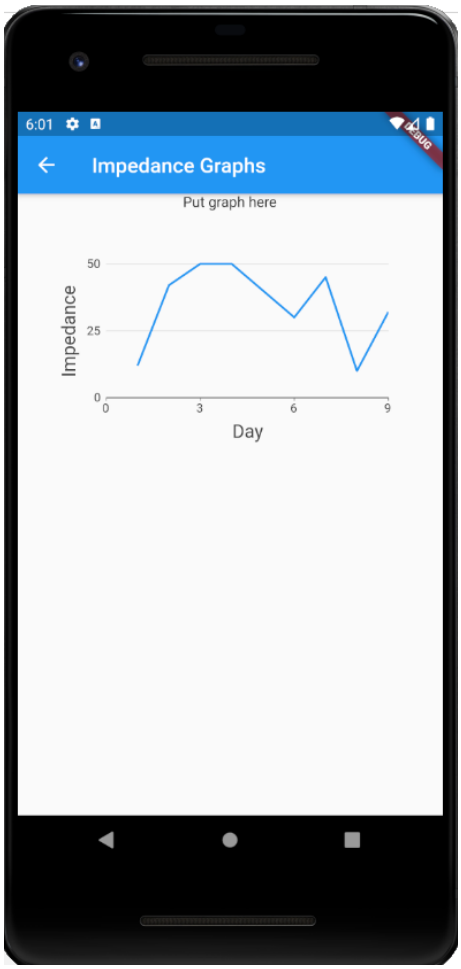

Figure-62: Graph Page

The above figure shows that a sample graph would look like. We would eventually add more graphs and the ability to pick a time scale.

Finally, I had implemented some code to connect a phone to the Bluetooth module on our Explorer Board, which was the RN4870 Click. This was code in progress and by no means finalized.

```
void connectBluetooth() async{

  //make instance
  FlutterBlue bluetooth = FlutterBlue.instance;
  StreamSubscription<ScanResult> scanSub;

  scanSub = bluetooth.scan().listen((scanResult){
    if(scanResult.device.name == TARGET_DEVICE_NAME){
      targetDevice = scanResult.device;
      connectToDevice(targetDevice);
      bluetooth.stopScan();
      scanSub?.cancel();
      scanSub = null;
    }
  });

  List<BluetoothService> services = await targetDevice.discoverServices();
  services.forEach((service){
    if(service.uuid.toString() == SERVICE_UUID){
      service.characteristics.forEach((characteristic){
        if(characteristic.uuid.toString() == CHAR_UUID){
          targetCharacteristic = characteristic;
          readStream = targetCharacteristic.value;
          targetCharacteristic.setNotifyValue(!targetCharacteristic.isNotifying);
        }
      });
    }
  });
}

connectToDevice(BluetoothDevice targetDevice) async{
  await targetDevice.connect();
}
```

Figure-63: Bluetooth App Code

The above figure shows how we would connect and read data from a phone via Bluetooth. This process would start of on the click of a button. The connection would start by scanning for all Bluetooth devices, and then connect to the target device, in this case, the RN4870. It would then scan that device's services and characteristics, looking for the target message. This message would cause an update to the app and display the data on the "Bluetooth Settings" page shown in the navigation page.

App Server (KL)

The app server would be implemented using AWS Lambda. This service allows API endpoints to be registered to handler functions call Lambda functions. This is called a "serverless" approach since the management of the hardware is handled by AWS and

the code is deployed to a virtual server when needed. We would create and implement various Lambda functions to complete the needs of our application. The pseudocode for those functions can be found below.

Register User Route:
Get username, password and user's body stats from request body
If missing info, return a http status 400
If all info there, hash password and insert row into database users table

Login Route:
Get username and password from http request body
Hash password and verify that it matches what is stored in the database
If password is incorrect, return a http status code of 403
If password is correct, create JWT that contains user id that will be used for authentication purposes

Get Measurements Route:
Check request for "Authorization" http header
If header not found, return an http status of 401
If header found, check that JWT is valid and extract user id from JWT
Query database for impedance measurements that belong to this user
Return found measurements in http response body

Add Measurement:
Check for authorization (same process as get measurements route)
Insert measurement into database table

6 - Mechanical Sketch of System (SW)

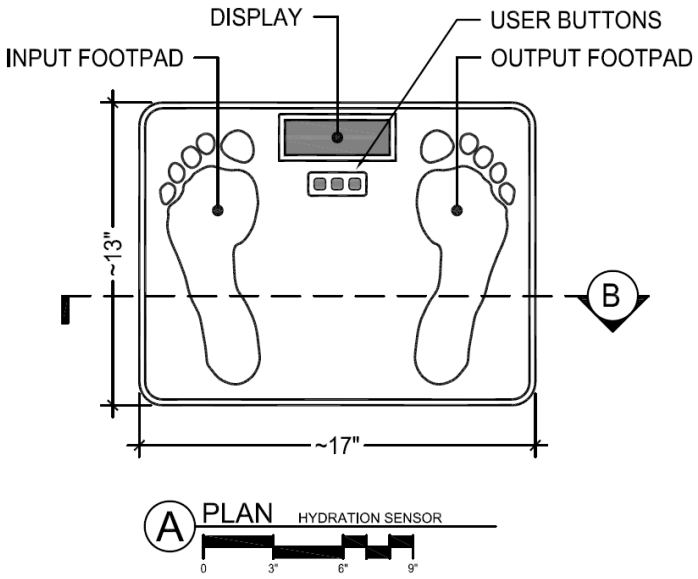

Figure-64: Mechanical Sketch (Plan View)

While additional configurations are being considered, Figure-64 shows the current design which is of a "bathroom scale" type of device. This is preferred as it's the most user-friendly and durable option.

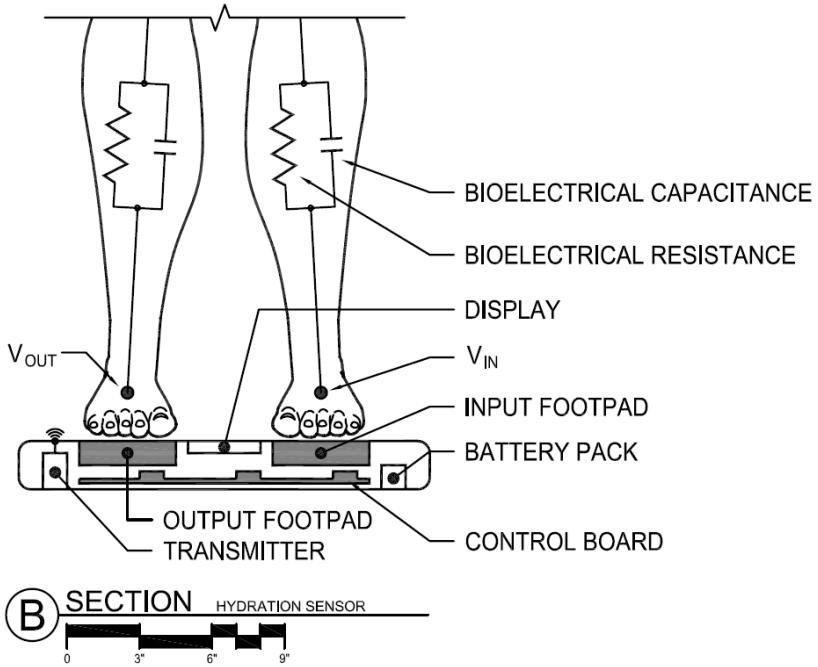

Figure-65: Mechanical Sketch (Section View)

Figure-65 shows a cross-section view of the Impedance Monitor device which describes the various systems in the Block Diagrams. Some segments of the system may be combined or separated into more-discrete modules as the design is taken to greater detail.

Figure-66 shows an overall section of the device and the User. This figure shows the approximate scale of the device in comparison to a person, and the approximate signal path through which the bioelectric impedance will be calculated.

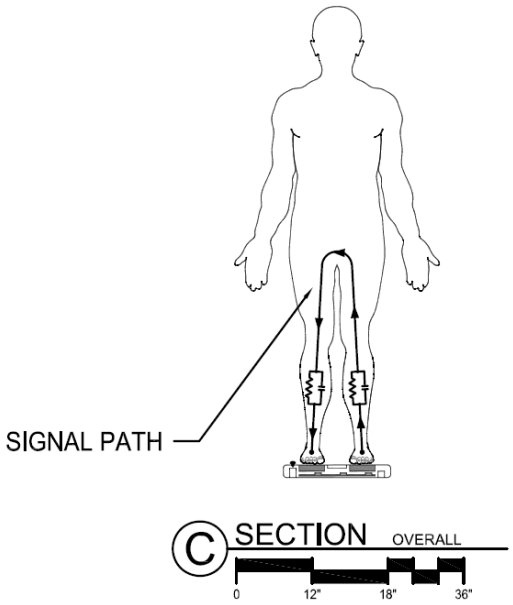

Figure-66: Mechanical Sketch (Overall View)

7 – Design Team Information (RB)

- Ryan Byo, Computer Engineering
- Kevin Libertowski, Computer Engineering
- Mitchell Sutyak, Electrical Engineering, No ESI
- Steve Weimer, Electrical Engineering, No ESI

8 – Parts Lists

8.1 – Parts List (SW)

Table 7.1 – Parts List			
Quantity	**Reference**	**Part Number**	**Description**
1		Explorer 16/32	Embedded Processor Board
1		dsPIC33FJ256GP710A	High-Frequency Sampling Processor
1			Bluetooth Module
1	Rs		Supply Resistor (1kΩ)
1	U1	TLE2142	Inverting OpAmp
2	U2, U3	TLE2142	Differential OpAmps
4	U4-U7	TLE2142	Buffer OpAmps
1	Rm		Measurement Resistor (1kΩ)
1	Rb	PTV09A-4020U-B104	Body Potentiometer (1k-100kΩ)
1	Cb	1266PH-ND	Body Capacitor (200pF)
1	Vs	9VDC Battery	Power Source for Embedded Processor
1	V+	(2) 9VDC Batteries	Positive Rail for OpAmps
1	V-	(2) 9VDC Batteries	Negative Rail for OpAmps
8	R9-R16		Buffer-Gain Resistors (vary)
8	R1-R8		Differential-Stage Resistors (vary)

8.2 – Materials Budget List (SW)

Table 7.2 – Materials Budget List				
Qty	**Part Number**	**Description**	**Unit Cost**	**Total Cost**
1	dsPIC33FJ256GP710A	High-Frequency Sampling Processor	9.67	9.67
1	TLE2142CP	Inverting OpAmp	4.67	4.67
1	RN4870 (Click Board)	Bluetooth Module	33.00	33.00
1	PTV09A-4020U-B104	Body Potentiometer (1k-100kΩ)	0.83	0.83
1	1266PH-ND	Body Capacitor (200pF)	0.5	0.5
5	BA9VPC	Battery Holder	2.29	11.45
2	Prototype Board	Prototype Board (Stock)	n/a	n/a
2	ST SIP	Single In-Line Solder Sockets	n/a	n/a
6	TLE2142CP	Low-Noise Diff OpAmps	4.67	28.02
			Total Cost	$88.14

The materials budget differed from the original materials list and the final materials list. The chief differences from the original materials list were the High-Frequency Sampling Processor and the OpAmps. When implementation was being done, it was determined the embedded processor was incapable of the sampling rate necessary to obtain useful data at high frequency (over 10kHz). The OpAmps were changed multiple times throughout implementation due to issues with low slew-rates and high-noise. The final materials list would have also included items for the device enclosure, sensor pads, etc; due to the unplanned Campus closure, none of the final assembly occurred.

9 – Project Schedule – Proposed Implementation Gantt Chart (MS)

	Task Mode	Task Name	Duration	Start	Finish	Resource Names
1		SDP I 2019				
2		**Project Design**	**64 days**	**Fri 8/30/19**	**Wed 11/27/19**	**Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer**
3		**Midterm Report**	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
4		Cover Page	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
5		T of C, L of T, L of F	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
6		**Problem Statement**	**29 days**	**Fri 8/30/19**	**Wed 10/9/19**	**Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer**
7		Need	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
8		Objective	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
9		Background	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
10		Marketing Requirements	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
11		Engineering Requirements Specification	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
12		**Engineering Analysis**	**29 days**	**Fri 8/30/19**	**Wed 10/9/19**	**Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer**
13		**Circuits**	**29 days**	**Fri 8/30/19**	**Wed 10/9/19**	**Steve Weimer,Mitchell Sutyak**
14		Differential Amplifier	29 days	Fri 8/30/19	Wed 10/9/19	Steve Weimer,Mitchell Sutyak
15		Voltage Controlled Current Source	29 days	Fri 8/30/19	Wed 10/9/19	Steve Weimer,Mitchell Sutyak
16		Inverting Amplifier	29 days	Fri 8/30/19	Wed 10/9/19	Steve Weimer,Mitchell Sutyak
17		**Electronics**	**29 days**	**Fri 8/30/19**	**Wed 10/9/19**	**Mitchell Sutyak,Steve Weimer**
18		Electrode Wiring	29 days	Fri 8/30/19	Wed 10/9/19	Mitchell Sutyak,Steve Weimer
19		**Signal Processing**	**29 days**	**Fri 8/30/19**	**Wed 10/9/19**	**Kevin Libertowski,Ryan Byo**
20		Topology of Oscillator	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
21		Frequency Sweep	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
22		Square Wave Input	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
23		**Computer Networks**	**29 days**	**Fri 8/30/19**	**Wed 10/9/19**	**Kevin Libertowski,Ryan Byo**
24		Wireless Module	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
25		Bluetooth	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
26		Server	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
27		**Psuedo Code**	**29 days**	**Fri 8/30/19**	**Wed 10/9/19**	**Kevin Libertowski,Ryan Byo**
28		Mobile Application	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
29		Hydration Model	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
30		FFT	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
31		Signal Generation	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
32		BIA Calculation	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
33		**Embedded Systems**	**29 days**	**Fri 8/30/19**	**Wed 10/9/19**	**Kevin Libertowski,Ryan Byo**
34		Embedded Processor	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
35		Analog to Digital Converter	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
36		Signal Analysis	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
37		Transmitter/Receiver	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
38		User Feedback (LCDs)	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
39		**Accepted Technical Design**	**29 days**	**Fri 8/30/19**	**Wed 10/9/19**	**Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer**
40		**Hardware Design: Phase 1**	**29 days**	**Fri 8/30/19**	**Wed 10/9/19**	**Mitchell Sutyak,Steve Weimer**
41		Hardward Block Diagrams Levels 0-N	29 days	Fri 8/30/19	Wed 10/9/19	Mitchell Sutyak,Steve Weimer
42		**Software Design: Phase 1**	**29 days**	**Fri 8/30/19**	**Wed 10/9/19**	**Kevin Libertowski,Ryan Byo**
43		Software Behavioral Models Levels 0-N	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Ryan Byo
44		**Mechanical Sketch**	29 days	Fri 8/30/19	Wed 10/9/19	Steve Weimer
45		**Team Information**	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
46		**Project Schedules**	**29 days**	**Fri 8/30/19**	**Wed 10/9/19**	**Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer**
47		Midterm Design Gantt Chart	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
48		**References**	29 days	Fri 8/30/19	Wed 10/9/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
49		Midterm Parts Request Form	32 days	Fri 8/30/19	Mon 10/14/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
50		**Preliminary Design Presentations**	0 days	Thu 9/19/19	Thu 9/19/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
51		Project Poster	10 days	Thu 10/10/19	Wed 10/23/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
52		Final Design Report	35 days	Thu 10/10/19	Wed 11/27/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
53		**Absract**	35 days	Thu 10/10/19	Wed 11/27/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
54		**Hardware Design: Phase 2**	**35 days**	**Thu 10/10/19**	**Wed 11/27/19**	**Mitchell Sutyak,Steve Weimer**
55		**Modules 1...n**	**35 days**	**Thu 10/10/19**	**Wed 11/27/19**	**Mitchell Sutyak,Steve Weimer**
56		Simulations	35 days	Thu 10/10/19	Wed 11/27/19	Mitchell Sutyak,Steve Weimer
57		Schematics	35 days	Thu 10/10/19	Wed 11/27/19	Mitchell Sutyak,Steve Weimer
58		**Software Design: Phase 2**	**35 days**	**Thu 10/10/19**	**Wed 11/27/19**	**Kevin Libertowski,Ryan Byo**
59		**Modules 1...n**	**35 days**	**Thu 10/10/19**	**Wed 11/27/19**	**Kevin Libertowski,Ryan Byo**
60		Code	35 days	Thu 10/10/19	Wed 11/27/19	Kevin Libertowski,Ryan Byo
61		System Integration Behavior Models	35 days	Thu 10/10/19	Wed 11/27/19	Kevin Libertowski,Ryan Byo
62		**Parts Lists**	**35 days**	**Thu 10/10/19**	**Wed 11/27/19**	**Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer**
63		Parts List(s) for Schematics	35 days	Thu 10/10/19	Wed 11/27/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
64		Materials Budget List	35 days	Thu 10/10/19	Wed 11/27/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
65		**Proposed Implementation Gantt Chart**	35 days	Thu 10/10/19	Wed 11/27/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
66		**Conclusions and Recommendations**	35 days	Thu 10/10/19	Wed 11/27/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
67		Final Parts Request Form	13 days	Thu 10/10/19	Mon 10/28/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
68		Final Design Presentations Part 1	0 days	Thu 11/14/19	Thu 11/14/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
69		Final Design Presentations Part 2	0 days	Thu 11/21/19	Thu 11/21/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
70		Parts Request Form for Spring Semester	9 days	Tue 11/26/19	Fri 12/6/19	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer

Figure-67: Original Gantt Chart

	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1		**SDPII Implementation 2020**	**103 days**	**Mon 1/13/20**	**Fri 4/24/20**		
2		Revise Gantt Chart	14 days	**Mon 1/13/20**	Sun 1/26/20		Mitchell Sutyak
3		**Implement Project Design**	**96 days**	**Mon 1/13/20**	**Fri 4/17/20**		**Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer**
4		**Hardware Implementation**	**49 days**	**Mon 1/13/20**	**Sun 3/1/20**		**Mitchell Sutyak,Steve Weimer**
5		**Validate Circuit**	**7 days**	**Thu 1/30/20**	**Wed 2/5/20**		**Mitchell Sutyak,Steve Weimer**
6		Add in LT1079	7 days	Thu 1/30/20	Wed 2/5/20		Mitchell Sutyak,Steve Weimer
7		Check performance band	7 days	Thu 1/30/20	Wed 2/5/20		Mitchell Sutyak,Steve Weimer
8		**Get Power Supply Working**	**7 days**	**Thu 2/6/20**	**Wed 2/12/20**		**Mitchell Sutyak,Steve Weimer**
9		Differential/single end inputs	7 days	Thu 2/6/20	Wed 2/12/20		Mitchell Sutyak,Steve Weimer
10		Battery regulator	7 days	Thu 2/6/20	Wed 2/12/20		Mitchell Sutyak,Steve Weimer
11		Build Gain Stage for Amplification and Attenuation	7 days	Thu 2/13/20	Wed 2/19/20		Mitchell Sutyak,Steve Weimer
12		**Midterm Demonstration and Troubleshooting**	**7 days**	**Thu 2/20/20**	**Wed 2/26/20**		**Mitchell Sutyak,Steve Weimer**
13		Functioning BIA with circuit/embedded integration	7 days	Thu 2/20/20	Wed 2/26/20		Mitchell Sutyak,Steve Weimer
14		Solder Board & Socket Strips for DIP's	7 days	Thu 2/27/20	Wed 3/4/20		Mitchell Sutyak,Steve Weimer
15		Finsh Soldering and Validate Circuit	7 days	Thu 3/5/20	Wed 3/11/20		Mitchell Sutyak,Steve Weimer
16		EagleCAD & PCB Schematic	7 days	Thu 3/12/20	Wed 3/18/20		Mitchell Sutyak,Steve Weimer
17		Finish PCB & Order	7 days	Thu 3/19/20	Wed 3/25/20		Mitchell Sutyak,Steve Weimer
18		Validate PCB and Integrate with Embedded	7 days	Thu 3/26/20	Wed 4/1/20		Mitchell Sutyak,Steve Weimer
19		Mount Hardware onto Acrylic Platform	7 days	Thu 4/2/20	Wed 4/8/20		Mitchell Sutyak,Steve Weimer
20		Final Validation of all Hardware	7 days	Thu 4/9/20	Wed 4/15/20		Mitchell Sutyak,Steve Weimer
21		Prep for Demo	7 days	Thu 4/16/20	Wed 4/22/20		Mitchell Sutyak,Steve Weimer
22		Final Report	7 days	Thu 4/23/20	Wed 4/29/20		Mitchell Sutyak,Steve Weimer
23		**Software Implementation**	**49 days**	**Mon 1/13/20**	**Sun 3/1/20**		**Kevin Libertowski,Ryan Byo**
24		**Verify AD/DMA and LCD Code**	**7 days**	**Thu 1/30/20**	**Wed 2/5/20**		**Ryan Byo**
25		Clean up code, add comments, break down	7 days	Thu 1/30/20	Wed 2/5/20		Ryan Byo
26		Start/stop measurements with button	7 days	Thu 1/30/20	Wed 2/5/20		Ryan Byo
27		Variably control sample rate	7 days	Thu 1/30/20	Wed 2/5/20		Kevin Libertowski
28		Test with RC circuit	7 days	Thu 1/30/20	Wed 2/5/20		Kevin Libertowski
29		**Generate Square Wave Voltage**	**7 days**	**Thu 2/6/20**	**Wed 2/12/20**		**Kevin Libertowski**
30		Variably change generated sqaure wave frequency	7 days	Thu 2/6/20	Wed 2/12/20		Kevin Libertowski
31		Communicate to bluetooth module with UART	7 days	Thu 2/6/20	Wed 2/12/20		Ryan Byo
32		**Tie Square Wave and AD Frequency Together**	**7 days**	**Thu 2/13/20**	**Wed 2/19/20**		**Kevin Libertowski**
33		Create routine for stepping through each frequency	7 days	Thu 2/13/20	Wed 2/19/20		Kevin Libertowski
34		Display frequency while measuring	7 days	Thu 2/13/20	Wed 2/19/20		Kevin Libertowski
35		Display impedance after all measured	7 days	Thu 2/13/20	Wed 2/19/20		Kevin Libertowski
36		Connect Bluetooth to mobile device	7 days	Thu 2/13/20	Wed 2/19/20		Ryan Byo
37		Send data package to phone	7 days	Thu 2/13/20	Wed 2/19/20		Ryan Byo
38		**Integrate and Test Embedded System with Hardware**	**7 days**	**Thu 2/20/20**	**Wed 2/26/20**		**Kevin Libertowski,Ryan Byo**
39		Prep for demo	7 days	Thu 2/20/20	Wed 2/26/20		Kevin Libertowski,Ryan Byo
40		**Setup Mobile Application Front End**	**7 days**	**Thu 2/27/20**	**Wed 3/4/20**		**Ryan Byo**
41		Setup back end (Server and DB)	7 days	Thu 2/27/20	Wed 3/4/20		Kevin Libertowski
42		**Send Impedance Measurements to Phone**	**7 days**	**Thu 3/5/20**	**Wed 3/11/20**		**Kevin Libertowski**
43		Receive Bluetooth message	7 days	Thu 3/5/20	Wed 3/11/20		Ryan Byo
44		Display impedance to user	7 days	Thu 3/5/20	Wed 3/11/20		Ryan Byo
45		**Save Measurements to DB**	**7 days**	**Thu 3/12/20**	**Wed 3/18/20**		**Kevin Libertowski**
46		Retreive measurements from DB	7 days	Thu 3/12/20	Wed 3/18/20		Kevin Libertowski
47		Display history of measurements to user with graphs	7 days	Thu 3/12/20	Wed 3/18/20		Ryan Byo
48		**User Account Info and Sign In Forms**	**7 days**	**Thu 3/19/20**	**Wed 3/25/20**		**Ryan Byo**
49		Save and retrieve user information from DB	7 days	Thu 3/19/20	Wed 3/25/20		Kevin Libertowski
50		Finish Mobile App	7 days	Thu 3/26/20	Wed 4/1/20		Kevin Libertowski,Ryan Byo
51		Post Processing - Hydration Calculations	7 days	Thu 4/2/20	Wed 4/8/20		Kevin Libertowski,Ryan Byo
52		Final System Integration and Debugging	7 days	Thu 4/9/20	Wed 4/15/20		Kevin Libertowski,Ryan Byo
53		Preparation for Demo	7 days	Thu 4/16/20	Wed 4/22/20		Kevin Libertowski,Ryan Byo
54		Final Report	7 days	Thu 4/23/20	Wed 4/29/20		Kevin Libertowski,Ryan Byo
55		**Develop Final Report**	**103 days**	**Mon 1/13/20**	**Fri 4/24/20**		**Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer**
56		Write Final Report	103 days	Mon 1/13/20	Fri 4/24/20		Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
57		Submit Final Report	0 days	Fri 4/24/20	Fri 4/24/20	56	Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
58		Spring Recess	7 days	Mon 3/23/20	Sun 3/29/20		Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer
59		***Project Demonstration and Presentation***	0 days	Mon 4/20/20	Mon 4/20/20		Kevin Libertowski,Mitchell Sutyak,Ryan Byo,Steve Weimer

Figure-68: Final Gantt Chart

The changes to the original and final Gantt Charts differ based on what was originally planned to be accomplished compared to what needed to be accomplished. At one point there was a plan to use a printed circuit board through EagleCAD, however, with a lack of time and availability of PCB's to be shipped, the final design was going to be soldered onto a protoboard. With regards to the hardware side of the project, figuring out which OpAmp to use ended up being the key to making the design work. Like the original Gantt Chart, in the final semester the same blocks of hardware were used -

inverting, VCCS, differentials - in order to obtain useable data. With regards to the electrodes, a simpler approach was going to be used by using two plates that the user would step onto in order to make the connection. This differed from the original idea of using electrodes that would be built into a scale like device.

10 – Conclusions and Recommendations (KL & SW)

These past semesters we have demonstrated the theory behind this project and have implemented many of the subsystems it consists of. We have designed and simulated the circuits required to send and analyze a signal through the body. We have implemented algorithms to calculate the impedance based on discrete samples of the input and output voltage and current waveforms. We have constructed and validated a breadboard circuit which closely matches the simulated Bioimpedance circuit. And we have demonstrated the ability to be able to sample analog signals using the embedded processor and output to the LCD of the Explorer board.

The next steps in this project – if the Spring Semester had not been interrupted due to the Campus Shutdown – would have been to finish integrating all of the subsystems, validate the overall combined system, and constructed an enclosure package for the user to interface with the device. While most subsystems were merely a matter of proper implementation, we recommend special attention be given to OpAmp selection; to construct a circuit of this type it was necessary to use fast (high slew-rate), low-noise OpAmps to achieve useful data.

11 - References

[1] Shanholtzer, B. A., & Patterson, S. M. (2003). Use of bioelectrical impedance in hydration status assessment: reliability of a new tool in psychophysiology research. International Journal of Psychophysiology, 49(3), 217-226.

[2] Yanovski, S. Z., Hubbard, V. S., Lukaski, H. C., & Heymsfield, S. B. (1996). Bioelectrical impedance analysis in body composition measurement-Proceedings of a National Institutes of Health Technology Assessment Conference held in Bethesda, MD, December 12-14, 1994-Introduction.

[3] Khalil, S. F., Mohktar, M. S., & Ibrahim, F. (2014). The theory and fundamentals of bioimpedance analysis in clinical status monitoring and diagnosis of diseases. *Sensors (Basel, Switzerland), 14*(6), 10895-928. doi:10.3390/s140610895

[4] Piccoli, A., Rossi, B., Pillon, L., & Bucciante, G. (1994). A new method for monitoring body fluid variation by bioimpedance analysis: The RXc graph. *Kidney International, 46*(2), 534-539. doi:10.1038/ki.1994.305

[5] Steven Brantlov, Lars Jødal, Aksel Lange, Søren Rittig, Leigh C. Ward. (2017) Standardisation of bioelectrical impedance analysis for the estimation of body composition in healthy paediatric populations: a systematic review. Journal of Medical Engineering & Technology 41:6, pages 460-479

[6] Ki Chul Cha, "Apparatus for Analyzing Body Composition Based on Bioelectrical Impedance Analysis and Method Thereof" U.S. Patent 6256532B1, July 3, 2001., Lars Jødal, Aksel Lange, Søren Rittig, Leigh C. Ward. (2017) Standardisation of bioelectrical impedance analysis for the estimation of body composition in healthy paediatric populations: a systematic review. Journal of Medical Engineering & Technology 41:6, pages 460-479. (SW)

[7] David Alan Benaron, "Hydration Monitoring Sensor and Method for Cell Phones, Smart Watches, Occupancy Sensors, and Wearables" U.S. Patent 2015/0148623A1, May 28, 2015. (SW)

[8] C H González-Correa and J C Caicedo-Eraso, "Bioelectrical Impedance Analysis (BIA): A Proposal For Standardization of the Classical Method in Adults," in *Journal of Physics: Conference Series 407012018, January 1, 2012* (SW)

[9] Vega, Almudena et al. "Any grade of relative overhydration is associated with long-term mortality in patients with Stages 4 and 5 non-dialysis chronic kidney disease" Clinical kidney journal vol. 11,3 (2018): 372-376. (SW)

[10] Tabinor, Matthew et al. "Bioimpedance-defined overhydration predicts survival in end stage kidney failure (ESKF): systematic review and subgroup meta-analysis" Scientific reports vol. 8,1 4441. 13 Mar. 2018, doi:10.1038/s41598-018-21226-y (SW)

[11] Kenefick RW, Cheuvront SN, Leon LR, O'brien KK. Dehydration and rehydration. In: Auerbach PS, Cushing TA, Harris NS, eds. Auerbach's Wilderness Medicine. 7th ed. Philadelphia, PA: Elsevier; 2017:chap 89. (SW)

[12] W.B. Kouwenhoven. "Human Safety and Electric Shock" Electrical Safety Practices, Monograph, 112, Instrument Society of America, P. 93. November 1968; referenced by the US Department of Labor Occupational Safety & Health Administration agency. (SW)

[13] H, Spencer Turner, M.D. "Human Responses to Electricity: A Literature Review" , prepared under contract NSR 36-008-108, The Aviation Medicine Research Laboratory Department of Preventative Medicine, The Ohio State University, 1972. (SW)

[14] "Worker Deaths by Electrocution: A summary of NIOSH Surveillance and Investigative Findings", US Department of Health and Human Services, May 1998. (SW)

12 – Appendices

1. PIC24FJ1024GB610 Data Sheet
   http://ww1.microchip.com/downloads/en/DeviceDoc/PIC24FJ1024GA610-GB610-Family-Data-Sheet-DS30010074F.pdf

2. PIC24FJ1024GB610 Data Sheet
   http://ww1.microchip.com/downloads/en/DeviceDoc/70286C.pdf

3. Explorer 16/32 Development Board User's Guide
   https://microchipdeveloper.com/boards:explorer1632

4. UA741 Data Sheet
   http://www.ti.com/lit/ds/slos094g/slos094g.pdf

5. LM318 Data Sheet
   http://www.ti.com/lit/ds/symlink/lm134.pdf

6. TLE2142 Data Sheet
   http://www.ti.com/lit/ds/slos628/slos628.pdf