IntelliSOLE

Final Report

ECE 445 (Spring 2022)

TA: Hanyin Shao

Team #18

Ritvik Avancha (rra2)

Alkesh Sumant (asumant2)

William Xu (xinyang8)

Abstract

For this project, we have designed and tested a foot pressure-mapping system which informs the user of their plantar foot pressure. This system has the capacity of measuring up to 750 kPa of pressure, and the wearer can track their change in foot pressure via phone or tablet by displaying a pressure-time graph on an android app. The data is transferred wirelessly via Bluetooth Low-Energy (BLE) between the shoe insole and the wireless device.

The foot pressure is measured using five force-sensing resistors (FSRs) attached to a shoe insole, and connected to an enclosure via ribbon cables from underneath the shoe insole. The enclosure is attached to the side of the shoe, which houses the PCB, battery and Bluetooth antenna. The PCB houses the analog circuitry needed to amplify the sensor signals, an analog-to-digital converter and digital signal processing subsystems, the Bluetooth module and the power distribution subsystem necessary for the function of these components.

Table of Contents

1. Introduction

1.1. Description of Problem

Walking and running are activities many of us take for granted. Yet, any disruption to these activities could pose enormous challenges to our daily lives. As such, studying human motion is important in monitoring health and preventing injury. Mapping foot plantar pressure can provide insights into the wearer's foot posture, and identify potential health issues that stem from having abnormal gait, which range from flat feet to feet ulceration problems caused by diabetes [1]. While there are solutions for foot pressure mapping, such as pressure mats and treadmills, these are often high-cost products that are not easily accessible to the average person. Current pressure-mapping shoe insoles on the market also suffer from the issue of high-cost, and face many complaints of being uncomfortable and bulky to wear.

1.2. Solution

The aim is to develop a low-cost, comfortable solution to map foot plantar pressure and provide the average user with information regarding their foot posture without having to see a doctor or spend lots of money on. The plan is to develop a shoe insole sensor that is compatible with any shoe and pair it with a mobile device application that allows the user to see their foot pressure in real time. As seen in Figure 1, the pressure information will be gathered by a network of force-resistive sensors (FSRs) attached to the shoe insole. The data is then transferred via ribbon cable to an assembly hanging on the shoe ankle, which houses the components necessary to process the pressure data. The data is then transferred to a mobile device via Bluetooth, where the software uses the data to generate a heatmap available to the user.

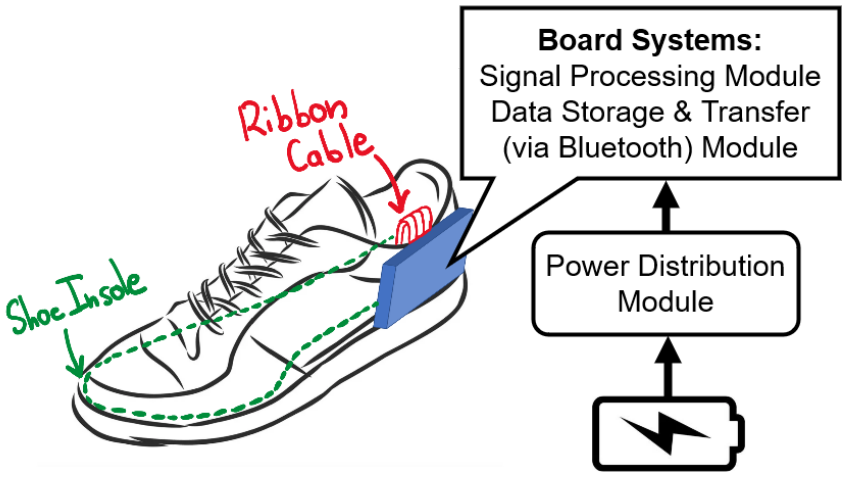


Figure 1: High-level Overview of Product Components (Visual Aid)

1.3. High-level Requirements List

Our success is defined by our ability to meet the criteria below:

1) Ability to sense up to 100 kPa ± 20% of pressure per sensor for accurate plantar pressure mapping.
2) Develop visualization corresponding to this range, indicating 0 kPa ± 20% to 100 kPa ± 20% of pressure.
3) The power source must be composed of small batteries with a voltage output between 3V ± 5% to 6V ± 5% and still efficiently use the power to function for more than 30 minutes.

2. Design

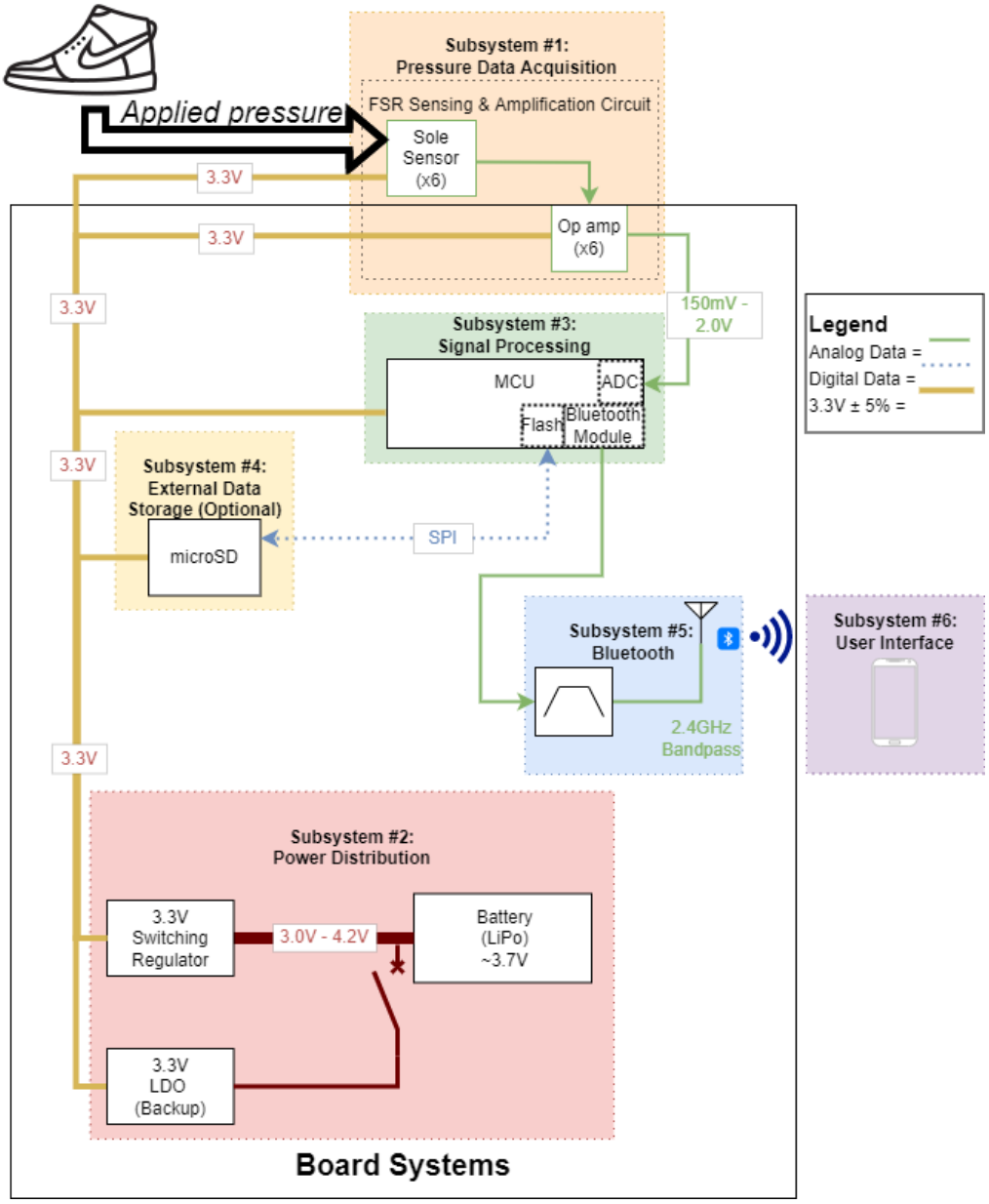


Figure 2: Subsystem Block Diagram

2.1. Design Procedure

As visualized in Figure 2, pressure data is collected from the shoe using the Pressure Data Acquisition subsystem, which uses an array of force-sensing resistors to detect changes in applied pressure. These changes in pressure are characterized by changes in the voltage response of these sensors, which are amplified and fed into the Signal Processing subsystem. The Signal Processing subsystem converts the analog data into digital data and applies digital filters to remove high frequency signal noise. After processing, the pressure data is wirelessly transmitted to the user's phone via the Bluetooth subsystem. The User Interface subsystem converts the received voltage data into a readable pressure data using linear regression model, and then creates a pressure versus time graph upon user's interaction.

Potential design alternatives included using a matrix-sensing array as the shoe insole instead of discrete sensors. The sensor design would be composed of flat copper wires arranged in perpendicular directions separated by a layer of Velostat (a pressure-sensitive conductive sheet). The changes in resistance measured within the array of copper lines would correspond to changes in pressure. The advantage of using this sensing approach is that the entire shoe insole would serve as a sensing array; however, this would require self-manufacturing of the sensor, increasing the difficulty of testing and establishing a range of linearity for pressure sensing. For this reason, it was determined that using existing sensors and incorporating them into the shoe insole would serve better given the time constraints of finishing the project within a semester. As seen in Figure 2, the External Data Storage subsystem was listed as optional. The purpose of this subsystem was to locally store pressure data onto an SD card, which can be transferred to a computer. This subsystem was kept as a backup in case data was not able to be transmitted using Bluetooth. As we had successfully implemented the Bluetooth subsystem, however, there was no need for local storage. While working on the data transmission subsystem, Bluetooth Low energy(BLE) protocol is chosen over regular Bluetooth protocol for energy efficiency purposes. BLE protocol can save up to 99% energy consumption compared to regular Bluetooth[6].

2.2. Design Details: Pressure Data Acquisition Subsystem

The Pressure Data Acquisition subsystem is composed of an array of six Interlink Electronics 400-Series Force Sensing Resistors (FSRs) placed throughout the shoe insole, as seen in Figure 3. These FSRs are attached to the bottom of the shoe insole using masking tape (between the shoe insole and base) with the sensing plane facing upwards. This improved the durability of the FSRs as they are not directly exposed to the shearing forces between the user's foot and the shoe insole, while increasing comfort as the wires connecting to the FSRs can be routed underneath the sole. As seen in Figures 3 and 4, a single FSR is modeled as a variable resistor within an adjustable buffer circuit, with R_{FSR} linearly decreasing based on pressure

applied normal to the sensing plane. When no pressure is applied, R_{FSR} was measured to be greater than 10 MΩ. The minimum activation force for sensor response was measured to be 0.04 N. The sensing area of the FSRs were measured to be 14.33 mm^2, resulting in a total sensing area of 85.98 mm^2.

An adjustable buffer circuit was an ideal choice for analyzing FSR pressure data, as the gain can be easily adjusted based on the ratio of R2 and R1. In addition, a greater degree of linearity was observed between resistance and output voltage compared to a standard voltage divider circuit. The ideal value of R3 was chosen to be one-twentieth the value of either R1 or R2 (no difference in circuit behavior was observed between these two changes). By running LTspice simulations, it was determined that a lower R2/R1 ratio resulted in higher output gain (VOUT). As such, the ratio was tweaked until the maximum output voltage of ~2.0 V was achieved. Figure 3 denotes the resistance values of R1 and R2.

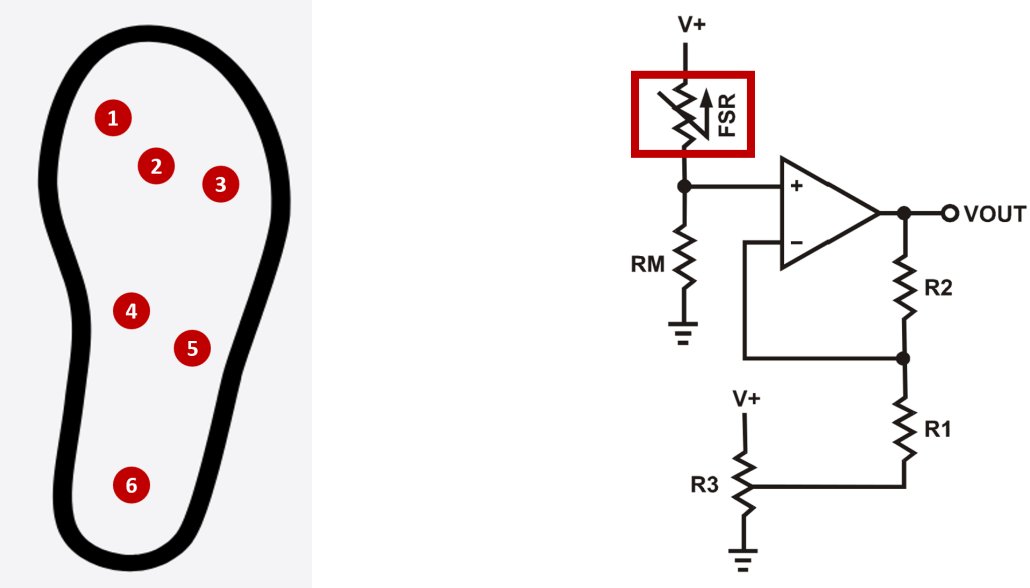


Figure 3: Placement of FSRs within shoe insole numbered 1-6 (left) and adjustable buffer circuit diagram with FSR component highlighted (right).

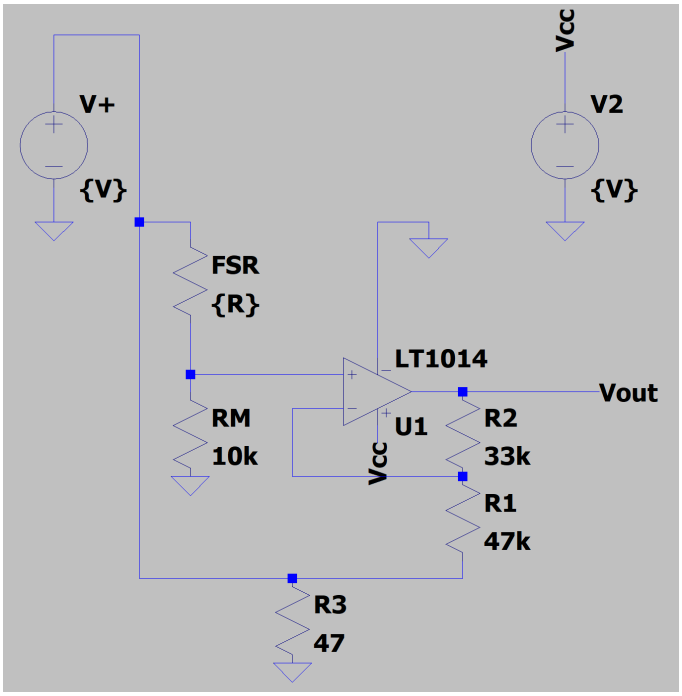


Figure 4: FSR amplifier circuit diagram.

2.3. Design Details: Power Distribution Subsystem

The power distribution subsystem is incharge of powering all the board system components. All components of the system require an operational voltage between 2.8V and 3.3V. The power is supplied by one 1200mAh LiPo cell rated for 3.7V. The cells are chemically powered by a lithium ion reaction with a conductive polymer gel. Since the LiPo cells come fully charged at 4.2V, this solution requires the regulation of the battery voltage down to 3.3V± 5% to power all the on-board subsystems.

The first and simplest option to regulate voltage at 3.3V is the low-dropout regulator (LDO) as illustrated in Figure 5. The efficiency of an LDO is determined by $\frac{V_{out}}{V_{in}} = \frac{3.3V}{V_{in}}$ where V_{in} ranges from 3.6V to 4.2V, the efficiency of the LDO ranges from 78.9% to 91.7%. The second option is to use a boost/buck converter aka a switching regulator illustrated in Figure 6. A switching regulator offers the benefit of stepping the battery voltage down with an efficiency of 95%. The drawback for increased efficiency is the increased circuit complexity compared to an LDO. The increased circuit complexity could pose a critical point of failure for the subsystem. In the interest of chasing efficiency without jeopardizing the success of the project, the PCB was designed for both a switching regulator and an LDO. The system design allows functionality as long as there is only one regulator soldered on board at a time.

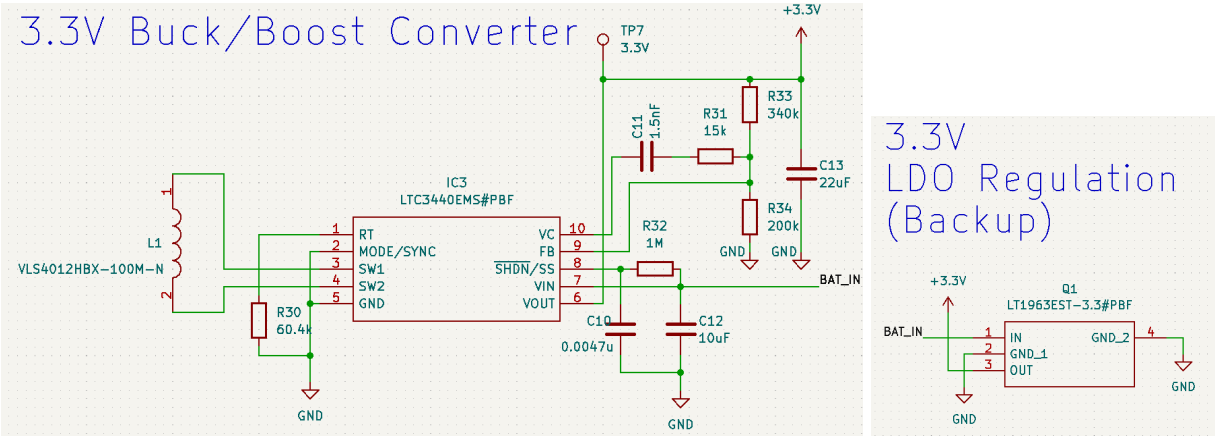


Figure 5: Voltage Regulator Solution Schematics

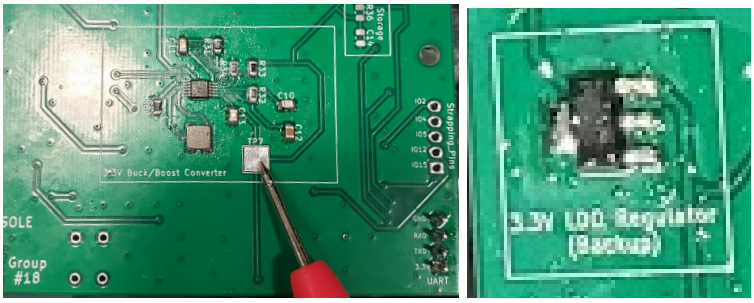


Figure 6: Voltage Regulator Solutions on PCB

2.4. Design Details: Signal Processing Subsystem

The microcontroller samples the analog resistance coming in from the pressure data acquisition subsystem and applies an embedded FIR filter to remove noise. After the filter is applied, the filtered values are stored onboard flash. If bluetooth is not connected, the data is written to the external memory in the form of microSD. ADC convergence timing of 1000ns so the convergence frequency is 1MHz convergence frequency.

2.5. Design Details: Bluetooth Data Transfer Subsystem

The bluetooth module serves as a more convenient medium to communicate data from the microcontroller to the device with the user interface module. The subsystem achieves bluetooth capabilities through the use of ESP32 along with the on-chip antenna. The bluetooth module receives the data from the MCU internally once signal processing is applied to the data. The bluetooth module encodes the data and runs through a bandpass filter before it is transmitted by a 2.4GHz antenna printed on the PCB. The embedded code for the bluetooth module is based on the BLE tutorials[6]. Upon initialization, the microcontroller will create a BLE server(service) that contains 6 characteristics(channels) corresponding to one pressure sensor(Figure 8). Each

time the signal processing module finishes cleaning up data, values in each channel are updated accordingly. The BLE server will then advertise those data in each channel(Figure 8).

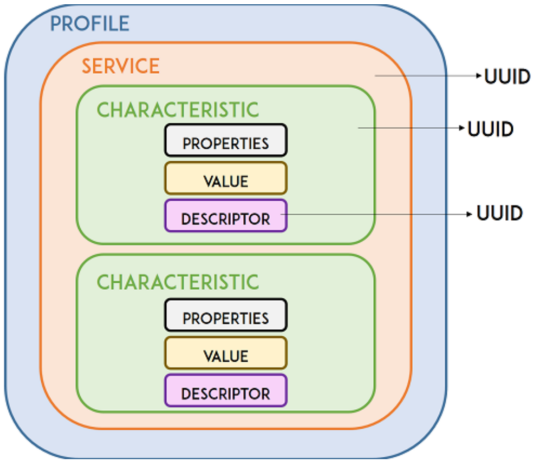


Figure 7: Bluetooth Set-up layers

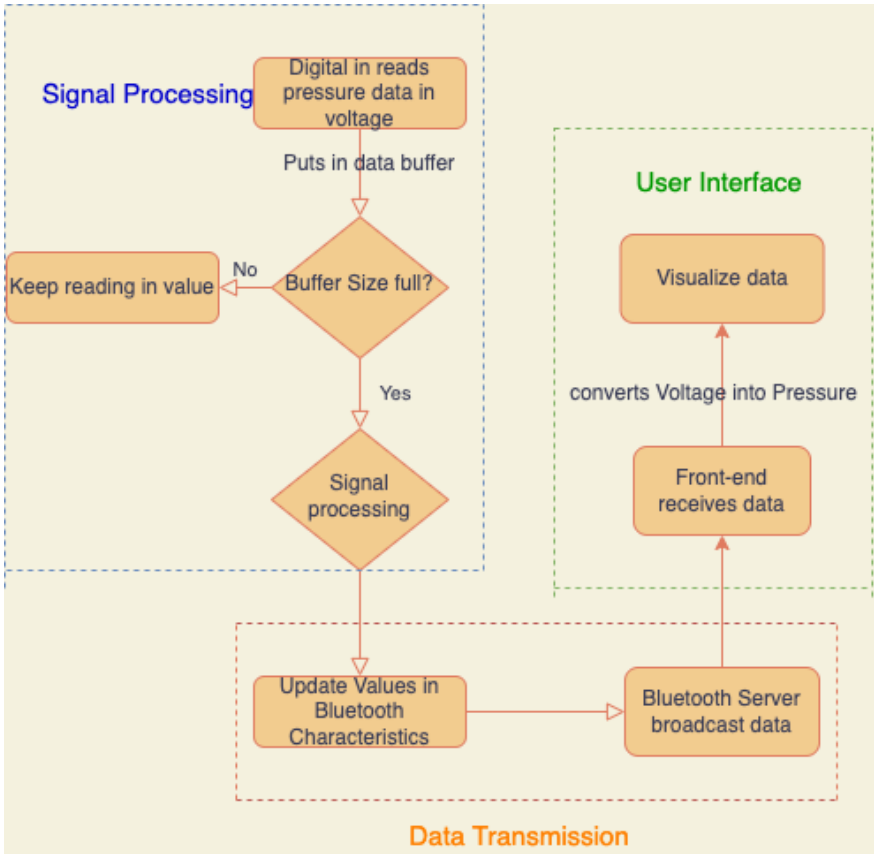


Figure 8: Flow Chart for software Subsystems

2.6. Design Details: User Interface Subsystem

The user interface interprets the sensor data and displays a graph for the user to visualize the foot pressure over time. This is accomplished by establishing connection between frontend and BLE broadcasting server on the microcontroller and then letting the user choose among 6 channels corresponding to each sensor(Figure 8 and 9). After the channel selecting fragment, the user will be taken into the visualization fragment(Figure 9) where the read button will trigger the graph to display the data received. The data being displayed will be preprocessed so as to transform voltage readings into pressure data using a linear regression model.

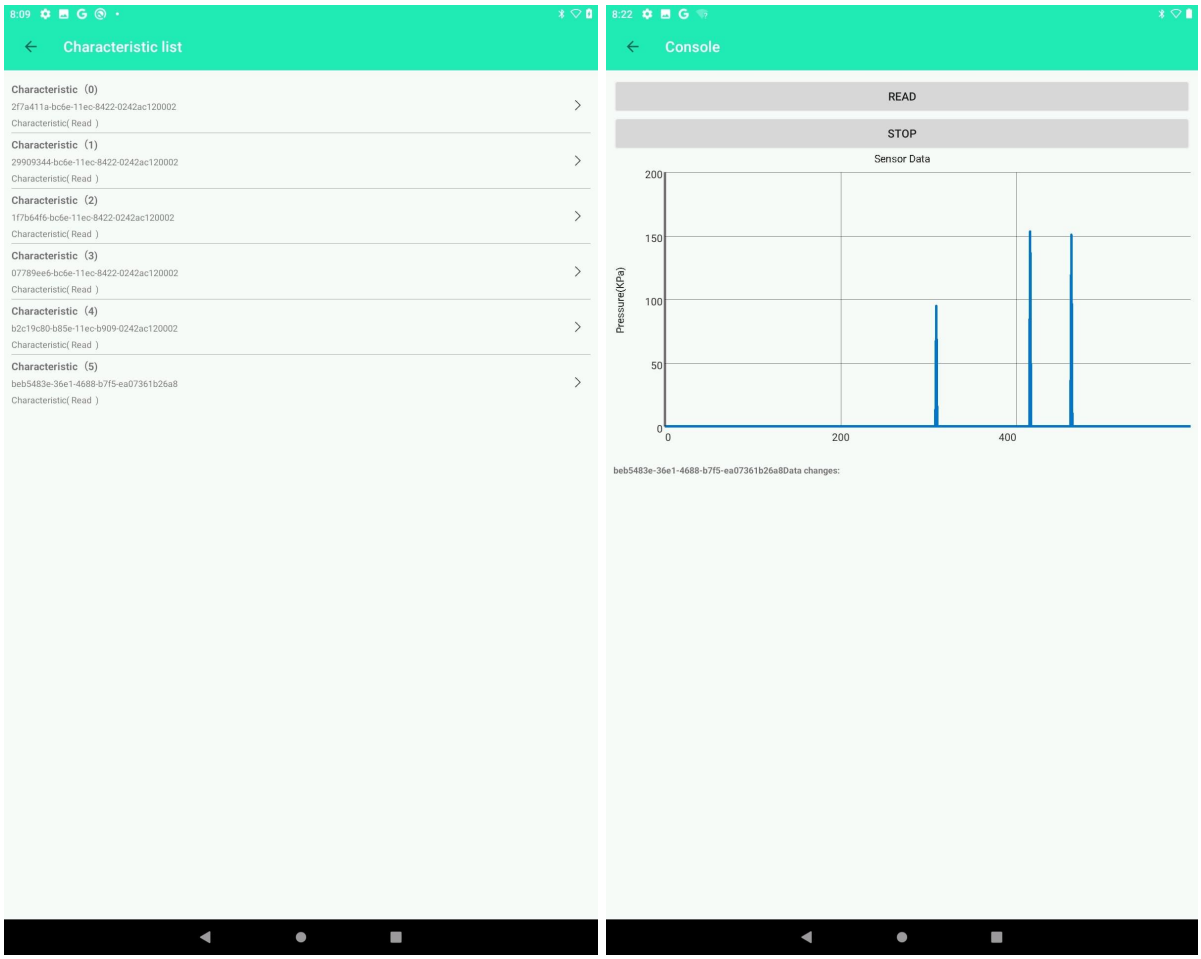


Figure 9: Visualization of buffer setup with rows denoting a sensor and columns denoting specific time instances.

3. Design Verification

3.1. Design Verification: Pressure Data Acquisition Subsystem

To verify that the maximum output voltage of the adjustable buffer circuit was within the range of 2V ± 5% when the FSR was under maximum load, the circuit was initially prototyped onto a breadboard connected to a power supply generating 3.3V. When the sensor was fully depressed, an output voltage of 2.005 V was measured using a multimeter probing the voltage output. This value falls within the 2V ± 5% margin of error. The purpose of this verification step was to ensure that the FSR voltage response was amplified enough to be detected by the analog-to-digital converters within the Signal Processing subsystem.

To verify that the FSR linearity was observed between force and resistance while the sensor load is less than 10 kg, weights in ascending order were placed on the FSRs while connected to the breadboard amplifier circuit. When the FSR was fully depressed, the output was observed to be 0.125 V. Figure 10 depicts the increasing voltage response for each corresponding increase in weight, which is shown to be a linear relationship between 4.5 to 254.5 grams of applied weight. Saturation of the FSR response occurred once the weight exceeded 255 g. The range of weight values in which a linear voltage output was observed was converted into pressure values using the formula depicted in Figure 11. It is shown that the maximum pressure value measured before the FSR entered saturation was approximately 170 kPa, higher than what was outlined in the High Level Requirements list (Section 1.3). The data displayed in figures 10 and 11 were fed into a linear regression calculator, which returned a linear model with mean square error of 13.88 and regression p_value of 0.00083 which is considered statistically significant for the model to be considered linear.

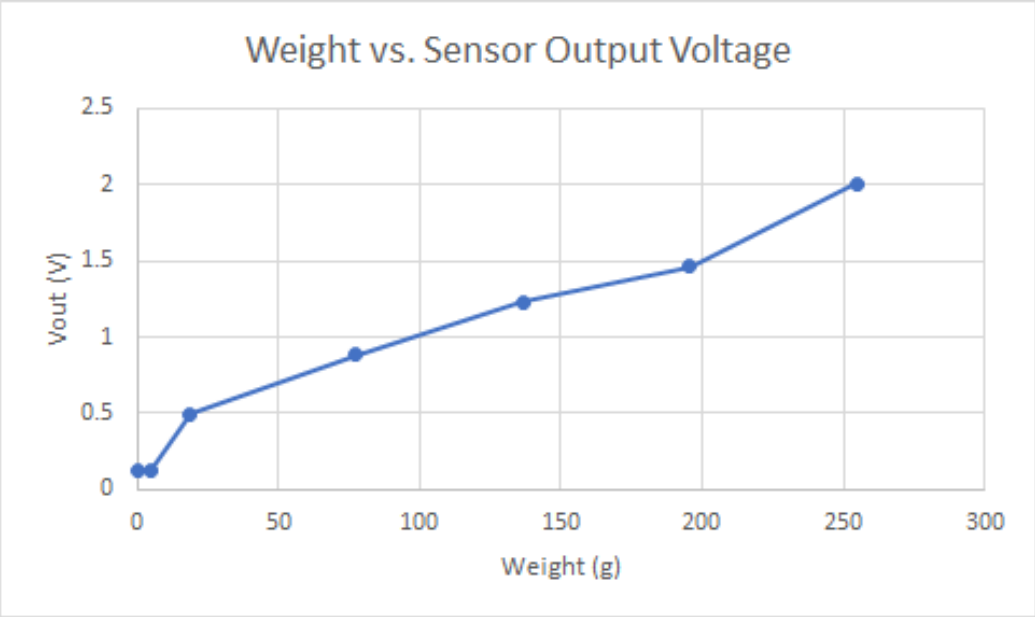


Figure 10. Linear relationship between weight and FSR response.

Weight (g)	Vout (V)
0	0.125
4.5	0.125
18.5	0.496
77.5	0.884
136.5	1.229
195.5	1.462
254.5	2.005

$$9.8 * \frac{(g)}{14.33\ mm^2}$$

Weight	kPa
0	0
4.5	3.004087
18.5	12.35014
77.5	51.73706
136.5	91.12398
195.5	130.5109
254.5	169.8978

Figure 11. Conversion of weight (in grams) to pressure values (in kPa)

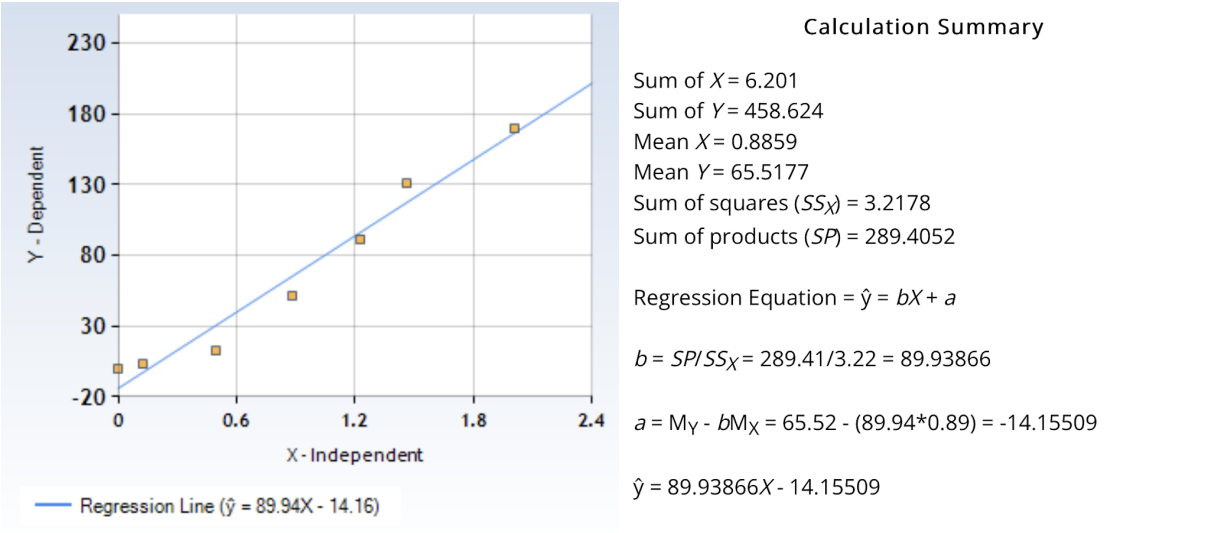


Figure 12: Linear Regression Model Calculation Between Voltage and Pressure

3.2. Design Verification: Power Distribution Subsystem

The power distribution subsystem is primarily required to regulate the battery voltage to 3.3V ±5% (see Appendix B, Table 3). To verify this requirement, only the components required to regulate power on the board were soldered and the battery terminals were connected to a power supply. The power supply voltage was swept from 3.6V to 4.2V and the output of the regulated terminals were probed.

The boost-buck converter solution was verified with the method described above. During verification, the regulated output voltage was measured at 1.85V ±0.15V thus the boost-buck solution does not meet the requirements of regulating output voltage at 3.3V ±5%. The root cause of the failure stems from the wrong variant of the boost-buck IC being shipped due to vendor error. The IC variant shipped to us was intended to regulate voltage at 1.8V as opposed to 3.3V.

Similarly, the low-dropout regulator solution was verified using the same method. The output terminals measured 3.3V ±0.15V thus this solution met the requirement of regulating output voltage at 3.3V ±5%.

Another requirement of the subsystem was to limit the battery current to 600mA to prevent over current scenarios. This was verified by soldering the relevant components to the PCB and connecting the battery terminals to a power supply. The power supply was set to output 700mA at 4.2V. The power solution managed to limit the power supply current to 600mA thus the requirements were successfully met.

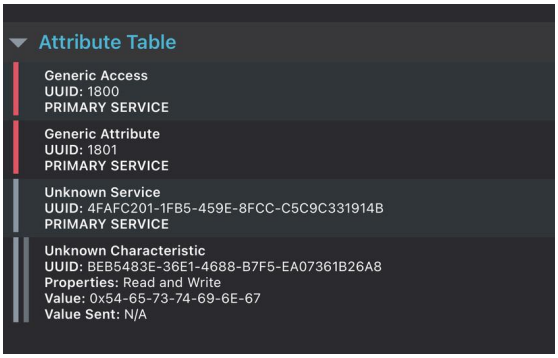
3.3. Design Verification: Signal Processing Subsystem

The signal processing subsystem is primarily required to convert analog data to digital values. This was verified using a waveform generator across the sensor terminals as described in Table 4 (see Appendix B). The waveform generator produced a sine wave with a frequency of 60Hz with an amplitude of 1.5V and an offset of 1.5V. The success of the analog to digital conversion was verified by utilizing a JTAG debugger connected to the microcontroller and visualized using the serial plotter. The shape of the converted waveform was very similar to the analog sine waveform produced without clipping or saturation indicating initial success of data conversion. The result is verified as a scaled digital waveform has a peak to peak amplitude of 3V.

Another requirement of the signal processing subsystem is the ability to digitally filter out high frequency noise. To verify the success of the digital filter, small signal noise was created using a waveform generator with an offset as described in Table 4 (see Appendix B) and the relevant filtering capacitors were desoldered. While the analog data was noisy as expected, the digital waveform illustrated a near linear line around 2V verifying the success of the digital filter.

3.4. Design Verification: Bluetooth Data Transfer Subsystem

The Bluetooth subsystem is required to set up a BLE server and establish stable broadcast channels to send out data received from the signal processing module. The verification process is that Bluetooth connection between our device and the microcontroller using nRF application be established and a string value (“testing”) hard-coded (Figure 13) to see if the data received is correct (Appendix B, Table 5). After a positive response, values are updated to match the data received from FSRs and by comparing to values displayed on oscilloscope and values received on client side.




```
// BLECharacteristic *pCharacteristic2 = pService->createCharacteristic(
//     CHARACTERISTIC2_UUID,
//     BLECharacteristic::PROPERTY_READ |
//         BLECharacteristic::PROPERTY_WRITE);
// pCharacteristic2->setValue("Tesing");
pService->start();
```

Figure 13: Bluetooth Data Transmission Tests

3.5. User Interface Subsystem

User Interface Subsystem is required to establish stable connection and receive data correctly from the microcontroller and while converting the voltage data into pressure data and visualize the data in a graph(Appendix B, Table 7).

As shown in Figure 9, a stable connection and correct reception of data from 6 characteristics are verified. While constantly receiving voltage data, linear transformation takes place and voltage is transformed into Pressure data. This is verified using Figure 10's linearity table.

A final verification is tested by having one of our teammates wearing the insole and reading the data on the frontend(Figure 14). As shown in the figure, a pike with up and down trendings is displayed on the graph with maximum pressure and minimum pressure the same with our projections: 0 Kpa when foot is in the air, a slow increase on the pressure when foot is being pressed onto the ground and 150 Kpa maximum pressure when the foot is fully pressed.

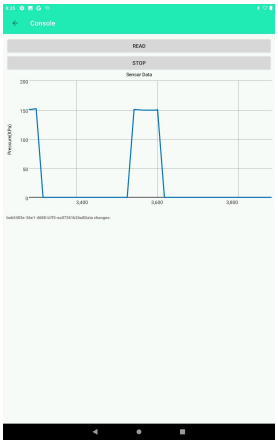


Figure 14: Data Visualization on a Step Taken by Use

4. Costs

Table 1: Cost breakdown of parts

Component	Subsystem	Total Item Cost (Quantity)
LM324PWRG3 Operational Amplifier	Pressure Data Acquisition	$4.04 (x10)
FSR 402 Force Sensing Resistor	Pressure Data Acquisition	$51.84 (x6)
BAT-HLD-003-SMT Battery Retainer	Power Distribution	$1.84 (x4)
36-1058-ND Battery Holder	Power Distribution	$4.92 (x4)
CR2032 Battery	Power Distribution	$4.66 (x12)
TPS62203 Voltage Step-Down	Power Distribution	$5.22 (x3)
ESP32-DEVKITC-32D	Signal Processing	$19.94 (x2)
ESP32-U4WDH	Signal Processing	$6.21 (x3)
HR1964TR-ND Micro-SD Connector	Data Storage	$2.50 (x1)
AP-MSD256ISI-1T Micro-SD	Data Storage	$12.92 (x2)
		TOTAL: $116.65

Labor Cost:

$Cost = \$43.75/hr * 3\,people * 2.5 * 10\,weeks * 10\,hrs/week = \$32,812.50$

$\frac{Cost}{person} = \$43.75/hr * 2.5 * 10\,weeks * 10\,hrs/week = \$10,937.50$

Total Cost: $\$32,812.50 + \$116.65 = \$32,929.15$

5. Conclusion

5.1. Accomplishments

Overall, we were able to demonstrate that IntelliSOLE was able to detect changes in pressure, convert the analog voltage response of the FSRs into digital pressure values, wirelessly transmit and visualize pressure values into a pressure versus time graph. The Data Acquisition subsystem displayed high accuracy and sensitivity in amplifying the voltage response of the FSRs when the pressure applied was within the linear range (3-170 kPa). The signal process subsystem successfully applies real time digital filtering and removes high frequency noise. The Bluetooth subsystem is able to broadcast the received data and establish a stable connection with the Frontend Subsystem while the Frontend subsystem is able to receive the data, transform the data into pressure data and display it on a graph.

5.2. Uncertainties

Pressure exceeding 170 kPa resulted in the FSRs reaching saturation and unable to change their voltage response for any additional change in pressure. As such, pressure cannot be accurately measured past 170 kPs, which poses an upper limit to how much pressure can be sensed by the shoe insole. This was a tradeoff that was considered, as flexible pressure sensing devices with thin profiles are known to have a limited sensing range - however, sensors with such profiles were needed in order to be successfully integrated into the space of a shoe.

Bluetooth connections are disabled each time the Frontend application is restarted and restart of the microcontroller is required to reestablish stable connections. The frontend application is also unfinished in that it cannot receive data from all 6 channels simultaneously.

5.3. Ethical Considerations

The primary aim of this project is to map the user's foot plantar pressure in order to provide information on the wearer's foot posture. This correlates with IEEE's Code of Ethics Section I.2, which is to "improve the understanding by individuals of the capabilities of conventional and emerging technologies, including intelligent systems [4]," as information is being provided regarding the wearer's health.

The mapping of the user's feet plantar pressure will be done by obtaining a finite set of sensor values from the shoe insole and using software to generate a continuous heatmap visualizing high and low areas of pressure. As such, it must be ensured that the visuals seen by the user are as accurate as possible to the data collected. However, in accordance with IEEE's Code of Ethics Section I.5 [4], it must be acknowledged that there will be limitations of this accuracy based on the sensor resolution and sensitivity, the placement of sensors on the shoe insole, and the algorithms used in software to generate the pressure heatmaps.

As this is a wearable product meant for regular use, extra steps were taken to ensure the safety of the user and developers in order to prevent injury, in accordance with IEEE's Code of Ethics Section I.1 [4]. The device power is supplied by CR2032 batteries, which were an ideal choice due to their low-profile and robust construction. These batteries utilize a Li/MnO_2 chemistry, which have the potential to explode or cause serious burns caused by the lithium component. As such, ensuring good handling of power distribution is essential for safe operations, and fail-safe mechanisms designed to cut-off battery power are implemented to ensure user safety. For example, Schottky diodes are connected in parallel with the battery to prevent current flow in the opposite direction if battery polarity is reversed. In addition, design considerations such as choosing slim FSRs and keeping the microcontroller/battery assembly as light as possible are considered to improve user comfort. This is particularly important as this product is aimed to be worn for extended periods of time.

In addition to ensuring user safety, considerations to minimize risk are taken within the design process. As this device requires PCBs, the use of soldering is required to attach the components to the board. Soldering produces dust and fumes which are considered hazardous, and so much of this work was conducted under a fume hood to minimize exposure.

5.4. Future Work

Future iterations of the IntelliSOLE can drastically reduce the footprint of the PCB along with the enclosure. Without using smaller components or redesigning the schematic, the PCB size can be reduced by optimizing the space on the rear of the PCB and omitting the redundant boost-buck power distribution solution in favor of the compact LDO.

Additionally, the shoe insole can be redesigned to allow for use of a matrix pressure-sensing array (as described in Section 2.1). Instead of retrofitting an existing shoe insole with force-resistive sensors, the matrix pressure-sensing array can be cut as per the shoe geometry, and can be used as the shoe insole itself. Depending on the number of copper rows and columns used, each copper line would require a signal amplification circuit similar in design to Figure 4 - but may require different resistance values. A linearity range between weight and voltage will have to be established for each of the copper lines within the matrix pressure-sensing array similar to the methods used in Section 3.1.

A new fragment in the Frontend application is planned to process received data and give out warnings to users about their foot postures based on the data. Effort on receiving different channels' data will also be made so that users will be able to see all the sensor data at once without going through different fragments.

References

[1] A. H. Abdul Razak, A. Zayegh, R. K. Begg, and Y. Wahab, "Foot Plantar Pressure Measurement System: A Review," Sensors, vol. 12, no. 7, pp. 9884–9912, Jul. 2012, doi: 10.3390/s120709884.

[2] Espressif Systems, "ESP32 Series," ESP32-U4WDH datasheet, Aug. 2016 [Revised Oct. 2021].

[3] Hirose Electric, "DM3 Series Data Sheet", DM3AT – SF – PEJM5 Datasheet

[4] IEEE, "IEEE Code of Ethics," *ieee.org*, 2018. https://www.ieee.org/about/corporate/governance/p7-8.html.

[5] Interlink Electronics, "FSR 400 Series Data Sheet," 30-81794 datasheet

[6] Random Nerds Tutorial, "Getting Started with ESP32 Bluetooth Low Energy", Updated May 16, 2019. https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/

Appendix A: Final Schematic

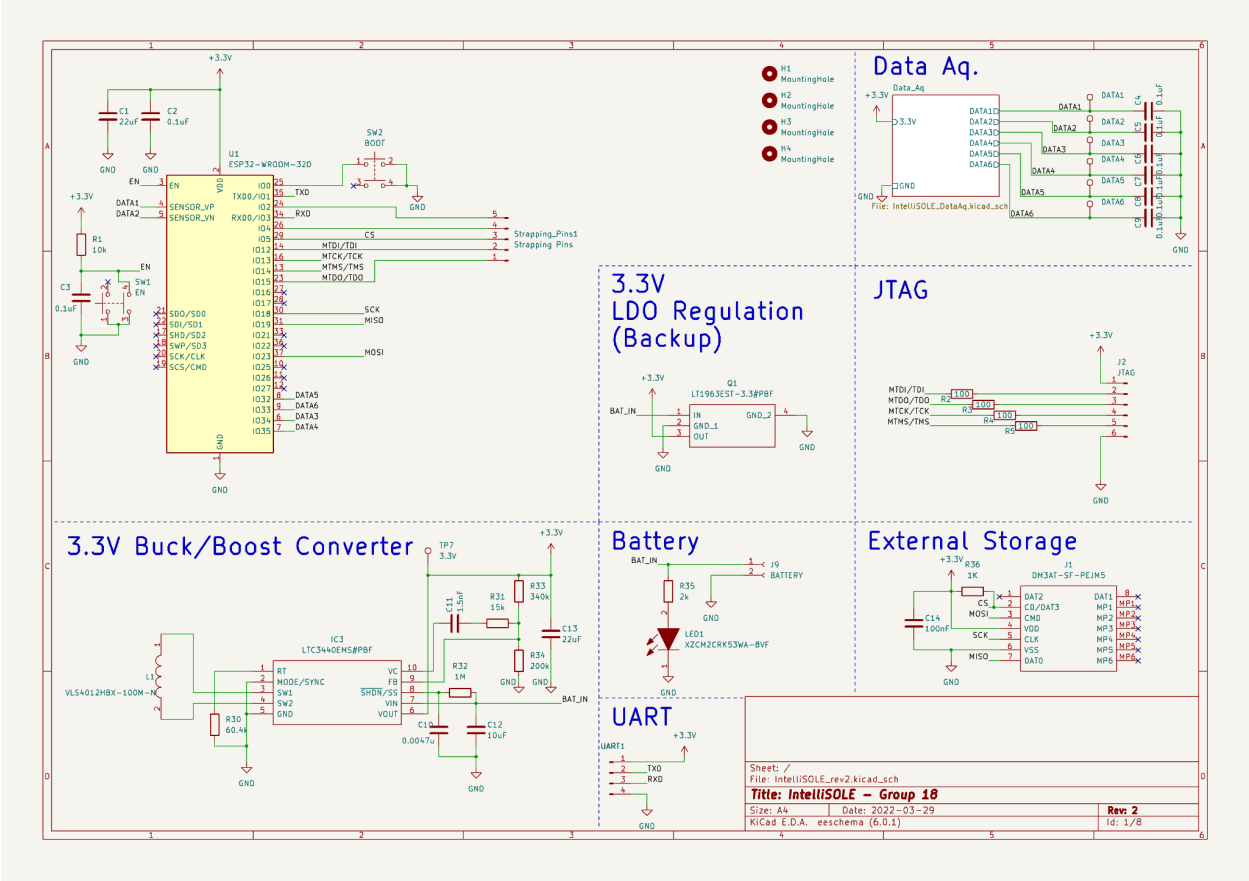


Figure 15: Final Board Schematic

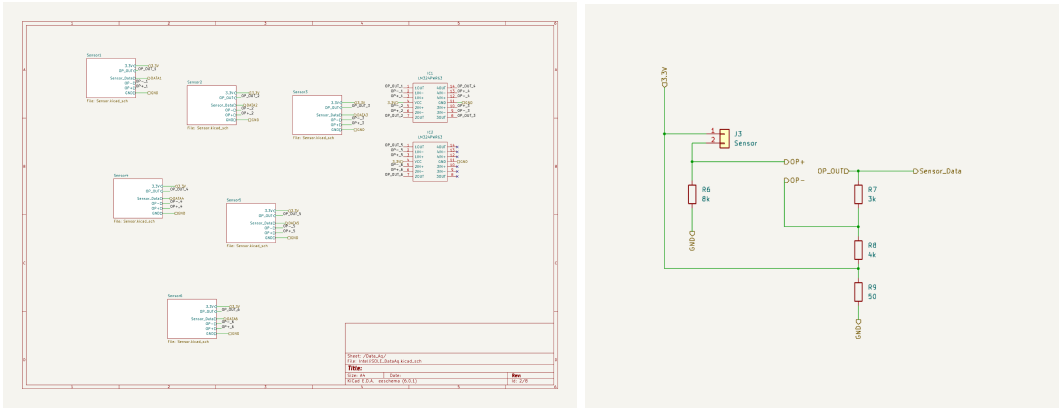


Figure 16: Data Acquisition Subsystem Schematics

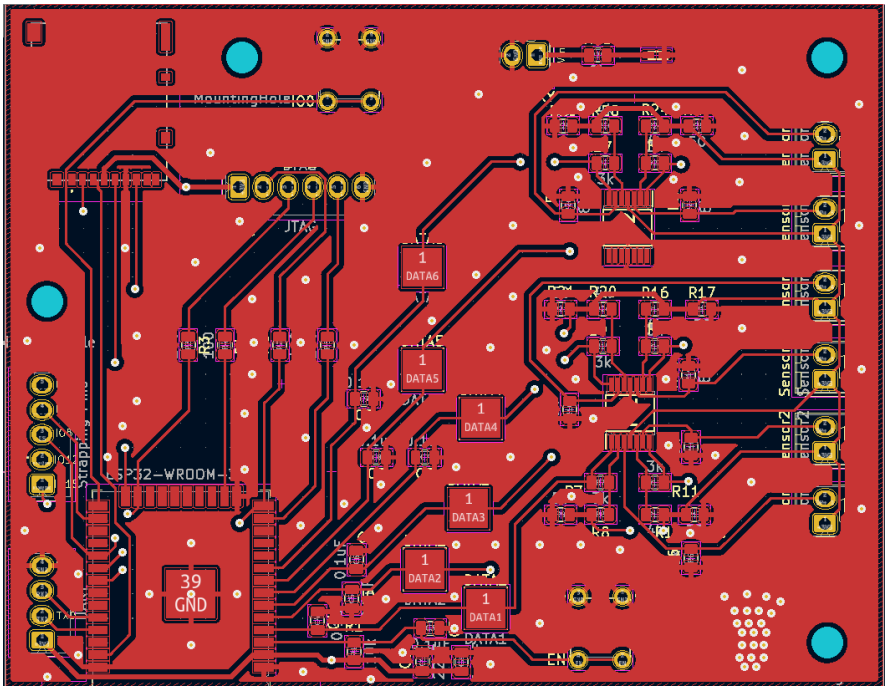


Figure 17: Final PCB Layout (Front)

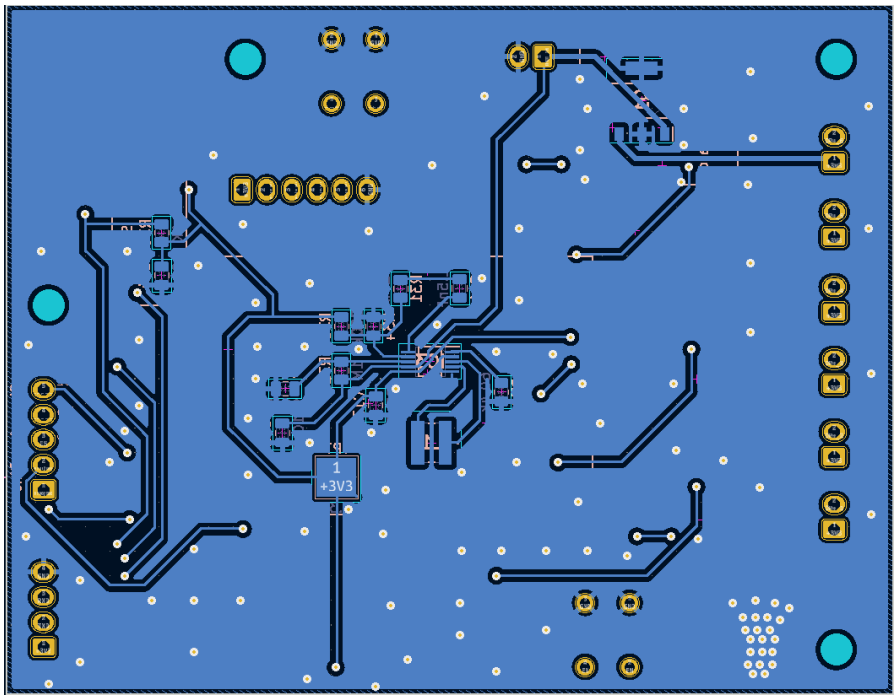


Figure 18: Final PCB Layout (Back)

Appendix B: Requirements

Table 2: Pressure Data Acquisition Requirements

Requirement	Verification	Result (Verified - 'V', Unverified - 'UV')
1. The maximum output voltage of the FSR circuit must be within a range of 2V ± 5% when the FSR is under maximum load.	**1A.** The FSR circuit must be prototyped on a breadboard using a power supply to generate 2V± 5% . **1B.** The output must be measured using an oscilloscope/voltmeter while the sensor is fully pressed.	**V**
2. FSR linearity must be observed between force and resistance while the sensor load is less than 10 kg.	**2A.** FSR circuit must be fully assembled on breadboard. **2B.** Use a multimeter to verify that each sensor exhibits an output less than 1 V when no force is applied. **2C.** Incrementally add weight to the FSR array and record corresponding voltage output. **2E.** Record at least two data points where linearity relationship is lost (ie. no changes in output voltage with increasing weight).	**V**

Table 3: Power Distribution Requirements

Requirement	Verification	Result (Verified - 'V', Unverified - 'UV')
1. DC-DC step down battery voltage from 3.5V - 4.2V to 3.3V ±5%.	**1A.** Use an oscilloscope to measure the output voltage is within 3.3V ±5%.	**V**
2. Ensure the circuit functions within a current limit of 600mA ± 50mA.	**2A.** Use an amp meter built into the power supply to verify the total current draw is within 600mA ± 50mA.	**V**

Table 4: Signal Processing Requirements

Requirement	Verification	Result (Verified - 'V', Unverified - 'UV')
1. Convert analog data to a digital reading.	**1A.** Simulate data using a signal generator to create a sine wave with a frequency of 60Hz with an amplitude of 1.5V and an offset of 1.5V. **1B.** Verify the converted digital data reads a sine wave ranging from 3.3V ± 10% to 0V ± 10%.	**V**
2. Filter out analog noise and analog-to-digital conversion noise using a digital filter.	**2A.** Simulate data with high-frequency noise or small signal noise by using a signal generator to create a sine wave with a frequency of 3KHz with an amplitude of 1V and an offset of 2V. **2B.** Verify the converted digital data reads a voltage of 2V ± 0.4V.	**V**

Table 5: Bluetooth Transmission Requirements

Requirement	Verification	Result (Verified - 'V', Unverified - 'UV')
1. Given accurate ADC sensor data, transmit the six channels of data via BLE.	**1A.** Simulating acquired data then successfully transmit via BLE and verify using a bluetooth scanner app. **1B.** Successful connection to the BLE server verified by using a bluetooth scanner app. **1C.** Successful data read from each BLE characteristic on the server demonstrated by using a bluetooth scanner app.	**V** - Steady connection between Microcontroller and nRF connect app. **V -** Correct preset string values can be received by nRF connect app.

Table 6: External Storage (OPTIONAL)

Requirement	Verification	Result (Verified - 'V', Unverified - 'UV')
1. The supply voltage needs to be within the range of 3.3V ±5% and power 1.5mA of current during write cycles	**1A.** Using an oscilloscope, ensure the power supplied is above 3.3V ±5%. **1B.** Using a multimeter, verify the power draw during write cycles is below 1.5mA	**UV**
2. The system would be able to detect and eliminate noises in data.	**2A.** Simulate a noise signal and add it to one of the lines. Then check on an oscilloscope to see if the signal is filtered out.	**UV**

Table 7: User Interface

Requirement	Verification	Result (Verified - 'V', Unverified - 'UV')
1. UI software must retrieve accurate ADC sensor data transmitted via BLE.	**1A.** Simulating acquired data then successfully transmit via BLE. **1B.** Successful connection to the BLE server verified by using breakpoints and a debugger. **1C.** Successful data read from each BLE characteristic on the server demonstrated by breakpoints and a debugger.	**V-** Application is able to receive the sensor data and is able to transform the voltage data into pressure data.
2. Given sensor data, the front end must display the acquired data in the form of a visualization.	**2A.** Use the android studio debugger to step through the visualization. **2B.** After pressing the read button, the user is able to see the visualization on screen for each sensor.	**V-** Application is able to show a graph of pressure v. time

Appendix C: Software Development

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include <Arduino.h>
#include "FS.h"
#include "SD.h"
#include <SPI.h>

// Bluetooth Definitions
#define SERVICE_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC1_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
#define CHARACTERISTIC2_UUID "b2c19c80-b85e-11ec-b909-0242ac120002"
#define CHARACTERISTIC3_UUID "07789ee6-bc6e-11ec-8422-0242ac120002"
#define CHARACTERISTIC4_UUID "1f7b64f6-bc6e-11ec-8422-0242ac120002"
#define CHARACTERISTIC5_UUID "29909344-bc6e-11ec-8422-0242ac120002"
#define CHARACTERISTIC6_UUID "2f7a411a-bc6e-11ec-8422-0242ac120002"
// Pin Definitions
#define DATA1_GPIO 36
#define DATA2_GPIO 39
#define DATA3_GPIO 34
#define DATA4_GPIO 35
#define DATA5_GPIO 32
#define DATA6_GPIO 33
#define SD_CS 5
// Parameter Definitions
#define FILTER_SIZE 16

String dataMessage;
int8_t dataIdx;
uint8_t data1;
uint8_t data2;
uint8_t data3;
uint8_t data4;
uint8_t data5;
uint8_t data6;
uint16_t data1_raw;
uint16_t data2_raw;
uint16_t data3_raw;
uint16_t data4_raw;
uint16_t data5_raw;
uint16_t data6_raw;
BLECharacteristic pCharacteristic1(CHARACTERISTIC1_UUID, BLECharacteristic::PROPERTY_READ);
BLECharacteristic pCharacteristic2(CHARACTERISTIC2_UUID, BLECharacteristic::PROPERTY_READ);
BLECharacteristic pCharacteristic3(CHARACTERISTIC3_UUID, BLECharacteristic::PROPERTY_READ);
BLECharacteristic pCharacteristic4(CHARACTERISTIC4_UUID, BLECharacteristic::PROPERTY_READ);
BLECharacteristic pCharacteristic5(CHARACTERISTIC5_UUID, BLECharacteristic::PROPERTY_READ);
BLECharacteristic pCharacteristic6(CHARACTERISTIC6_UUID, BLECharacteristic::PROPERTY_READ);

void setup()
{
 Serial.begin(115200);

 dataIdx = 0;
 data1 = 0;
 data2 = 0;
 data3 = 0;
 data4 = 0;
 data5 = 0;
 data6 = 0;
 data1_raw = 0;
```

```
 data2_raw = 0;
 data3_raw = 0;
 data4_raw = 0;
 data5_raw = 0;
 data6_raw = 0;

 BLEDevice::init("ECE445Team18");
 BLEServer *pServer = BLEDevice::createServer();
 BLEService *pService = pServer->createService(SERVICE_UUID);

 // BLECharacteristic *pCharacteristic1 = pService->createCharacteristic(
 //     CHARACTERISTIC1_UUID,
 //     BLECharacteristic::PROPERTY_READ |
 //         BLECharacteristic::PROPERTY_WRITE);
 // pCharacteristic1->setValue(&data1, sizeof(data1));
 pService->addCharacteristic(&pCharacteristic1);
 pService->addCharacteristic(&pCharacteristic2);
 pService->addCharacteristic(&pCharacteristic3);
 pService->addCharacteristic(&pCharacteristic4);
 pService->addCharacteristic(&pCharacteristic5);
 pService->addCharacteristic(&pCharacteristic6);

 // BLECharacteristic *pCharacteristic2 = pService->createCharacteristic(
 //     CHARACTERISTIC2_UUID,
 //     BLECharacteristic::PROPERTY_READ |
 //         BLECharacteristic::PROPERTY_WRITE);
 // pCharacteristic2->setValue("Tesing");
 pService->start();

 BLEAdvertising *pAdvertising = pServer->getAdvertising();
 pAdvertising->start();
}

void loop()
{
 // put your main code here, to run repeatedly:
 data1_raw += analogRead(DATA1_GPIO); // read value into a 16 bit holder
 data2_raw += analogRead(DATA2_GPIO);
 data3_raw += analogRead(DATA3_GPIO);
 data4_raw += analogRead(DATA4_GPIO);
 data5_raw += analogRead(DATA5_GPIO);
 data6_raw += analogRead(DATA6_GPIO);

 dataIdx++;
 if (dataIdx >= FILTER_SIZE)
 {
   data1_raw /= FILTER_SIZE; // read value into a 16 bit holder
   data2_raw /= FILTER_SIZE;
   data3_raw /= FILTER_SIZE;
   data4_raw /= FILTER_SIZE;
   data5_raw /= FILTER_SIZE;
   data6_raw /= FILTER_SIZE;
   data1_raw = data1_raw / 16; // recast 16bit to 8bit by right shifting 8 bits
   data2_raw = data2_raw / 16;
   data3_raw = data3_raw / 16;
   data4_raw = data4_raw / 16;
   data5_raw = data5_raw / 16;
   data6_raw = data6_raw / 16;
   data1 = data1_raw;
   data2 = data2_raw;
   data3 = data3_raw;
   data4 = data4_raw;
   data5 = data5_raw;
```

```
    data6 = data6_raw;
    // data1 = 2;
    (&pCharacteristic1)->setValue(&data1, sizeof(data1));
    (&pCharacteristic2)->setValue(&data2, sizeof(data2));
    (&pCharacteristic3)->setValue(&data3, sizeof(data3));
    (&pCharacteristic4)->setValue(&data4, sizeof(data4));
    (&pCharacteristic5)->setValue(&data5, sizeof(data5));
    (&pCharacteristic6)->setValue(&data6, sizeof(data6));

    dataIdx = 0;
    data1_raw = 0; // read value into a 16 bit holder
    data2_raw = 0;
    data3_raw = 0;
    data4_raw = 0;
    data5_raw = 0;
    data6_raw = 0;
  }
}
```