MONITORED COMPRESSION THERAPY: USING SMART TECHNOLOGY TO OPTIMIZE THE TREATMENT OF LOWER EXTREMITY SWELLING

By

James Skelly

Bachelor of Science in Electrical Engineering
University of Nevada, Las Vegas
2020

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Engineering – Electrical Engineering

Department of Electrical and Computer Engineering
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
December 2021

©Copyright 2022 by James Skelly

All Rights Reserved

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

November 5, 2021

This thesis prepared by

James Skelly

entitled

Monitored Compression Therapy: Using Smart Technology to Optimize the Treatment of Lower Extremity Swelling

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Engineering – Electrical Engineering
Department of Electrical and Computer Engineering

R. Jacob Baker, Ph.D.
Examination Committee Chair

Sarah Harris, Ph.D.
Examination Committee Member

Brendan Morris, Ph.D.
Examination Committee Member

Dustin Hines, Ph.D.
Graduate College Faculty Representative

Kathryn Hausbeck Korgan, Ph.D.
Vice Provost for Graduate Education &
Dean of the Graduate College

ABSTRACT

Chronic Venous Insufficiency (CVI) and Venous Stasis Ulcers (VSUs) are symptoms which stem from diabetes – a disease effecting over 34 million people in the United States alone as of 2020. This Thesis details the design of a pressure-sensing garment used to enhance the treatment of CVI. The garment uses small force sensors (four levels: the insole of the foot, the lower leg, the lower calf, and the upper calf) to sense the pressure applied by a compression stocking. The sensed data is transmitted wirelessly via Bluetooth to a smartphone application that was developed to display the data and interface with the electronics. The data is displayed on the smartphone application and can be monitored by the patient and/or nurse to ensure that the proper pressure gradient is applied to the leg. The gradient starts at around 30-50 mmHg at the foot and decreases linearly to about 6 mmHg just below the knee. The proper application of this pressure gradient best promotes blood flow and is predicted by medical experts to potentially cut healing time for VSUs from 6 months to as little as just 60 days.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my parents Kami and Jim Skelly, who have spent countless dollars supporting and feeding me, countless hours encouraging and praying for me, and nearly twenty-four years believing in me. Without your endless support and guidance, I would be nowhere near where I am today physically, academically, professionally, or spiritually. I would also like to thank my brother, Ryan Skelly, for keeping me young and for being an extra voice of reason when I get into heated sports debates with my dad and uncle. Mom, dad, and Ryan, I love you.

I would like to thank Dr. Baker for the wide variety of opportunities I have been provided in my nearly 4 years in his research group. Dr. Baker has been a role model for me as an engineer and an educator ever since I first took his *Circuits I* course in the spring of 2017. His passion for teaching, and his mastery of the subject matter which he teaches, have inspired me to challenge myself to be the best engineer that I can be–to never fear testing out ideas I may have, to embrace opportunities when they come my way, and to seek and value knowledge and gained experience in all my engineering-related endeavors. I would also like to thank Dr. Harris for always going above and beyond to help me both inside and outside of the classroom. Dr. Harris has a special, kind heart and puts her students at the top of her enormous and impressive list of priorities.

I would like to thank the rest of my advisory committee for being a part of this defense and for their time and investment in my education – Dr. Brendan Morris and Dr. Dustin Hines. A special thanks to Dr. Morris for always being available to answer questions I may have had about a class I was taking, or about the future of my education and career. I would also like to thank Dr. Todd Meyrath for the enormous impact he has had on my education in electrical engineering, and for the practical experience I gained working for him at *Vorpal*. Todd has also been incredibly helpful in the process of applications for PhD programs and has offered priceless career advice to me in exchange for nothing.

A special thanks to my colleague and friend Francisco Mata. Francisco and I have worked as partners tirelessly on countless projects. I could not be more grateful to have worked in the same research group as someone with such impeccable work ethic. Francisco doesn’t leave until the job is done, and he only knows how to give 100% effort in everything he does. His effort and his desire to learn are contagious and have rubbed off on me, both in academia and elsewhere. I would also like to thank Dr. Sachin Namboodiri from the bottom of my heart for never even hesitating to drop what he is doing to help me with a project, an assignment, or a question I may have had about any number of things. Sachin has had an immeasurable impact on my education, and I could never thank him enough for his efforts. Another special thanks to Daniel Senda for being a great colleague and a great friend. Daniel taught me how to solder surface mount components, helped me with my consulting work, and has been a great study partner throughout several courses we have taken together. I would like to thank Jazmine Boloor for her friendship and constant encouragement throughout my time in this group. Jazmine always provides me with helpful feedback on projects, presentations, and emails, and she even let me use half of her shelving at her desk to store parts for my projects for at least a year and a half. Lastly, I would like to thank the rest of the past and present members of the Baker group for guidance, support, and friendship throughout my time in the group.

TABLE OF CONTENTS

ABSTRACT .... **iii**

ACKNOWLEDGMENTS .... **iv**

TABLE OF CONTENTS .... **vi**

LIST OF TABLES .... **ix**

LIST OF FIGURES .... **x**

LIST OF ABBREVIATIONS/ACRONYMS .... **xiii**

LIST OF CONVERSION FACTORS .... **xv**

CHAPTER 1: INTRODUCTION **1**

1.1 THE PROBLEM 1

1.2 THE PROPOSED SOLUTION 5

CHAPTER 2: SYSTEM DESIGN OVERVIEW **6**

2.1 PROTOTYPE BUILD 6

2.2 TESTING AND RESULTS 10

2.3 FROM SENSOR TO SMARTPHONE 14

2.4 CHAPTER 2 SUMMARY 16

CHAPTER 3: SENSING **17**

3.1 SENSING FORCE 17

3.2 AMPLIFYING THE SENSOR OUTPUT VOLTAGE 20

3.3 FORCE SENSOR CANDIDATES 22

3.4 FORCE SENSOR/AMPLIFIER CIRCUIT BOARD DESIGN 26
3.5 FORCE SENSOR HOUSING MECHANISM 28
3.6 CHAPTER 3 SUMMARY 29
CHAPTER 4: DATA PROCESSING 30
4.1 THE MICROCONTROLLER (PIC18LF26K22) 30
  4.1.1 THE ADC (ANALOG-TO-DIGITAL CONVERTER) 30
  4.1.2 DATA PROCESSING 34
  4.1.3 UART COMMUNICATION 39
  4.1.4 HARDWARE IMPLEMENTATION 42
4.2 THE BLUETOOTH MODULE 44
  4.2.1 CONFIGURATION WITH AT-COMMANDS 45
  4.2.2 DATA TRANSMISSION AND RECEPTION 48
4.3 CHAPTER 4 SUMMARY 51
CHAPTER 5: THE SMARTPHONE APPLICATION 52
5.1 INTERFACE AND LAYOUT 52
5.2 RECEIVING AND DISPLAYING DATA 56
5.3 CHAPTER 5 SUMMARY 63
CHAPTER 6: POWERING THE SYSTEM 64
6.1 POWER SOURCE 64
6.2 VOLTAGE REGULATION 65
6.3 POWER CONSUMPTION AND BATTERY LIFE 67

6.4 CHAPTER 6 SUMMARY 73

CHAPTER 7: FUTURE WORK AND CONCLUSION 74

7.1 FUTURE WORK AND IMPROVEMENTS 74

7.1.1 MAIN MCU/BLE CIRCUIT 74

7.1.2 SENSOR/AMPLIFIER CIRCUIT AND SENSING MECHANISM 78

7.1.3 SOFTWARE 80

7.1.4 POWER AND BATTERY LIFE 81

7.1.5 OTHER IMPROVEMENTS 83

7.2 CONCLUDING REMARKS 86

APPENDIX A: SCHEMATICS 88

APPENDIX B: SINGLE UNIT COST BREAKDOWN 90

APPENDIX C: MASS PRODUCTION (1000+) UNIT COST BREAKDOWN 91

APPENDIX D: MICROCONTROLLER (C LANGUAGE) SOURCE CODE 92

APPENDIX E: MIT APP INVENTOR BLOCKS VIEW 98

REFERENCES 99

CURRICULUM VITAE 102

LIST OF TABLES

Table 2.1 List of Parts for Prototype Build 9

Table 2.2 Test Data Results Analysis and Error Metrics 13

Table 6.1 Current Draw Distribution for Major System Components 68

Table 6.2 Buck-Boost Efficiency for Constant 16.2 mA Load, Input Voltage Swept 72

Table 7.1 Imperial SMD Package Types and Dimensions 75

Table 7.2 Changes in LSB Voltage with Different ADC Resolutions, VDD = 3.3V 76

Table 7.3 List of BLE Modules, Typical On-Currents, and Cost 83

Table B.1 Cost Breakdown by Part for One Single Unit 90

Table C.1 Cost Breakdown by Part for One Single Unit in Mass Production 91

LIST OF FIGURES

Figure 1.1 Illustration of Compression Assisting a Varicose Vein [4] 1

Figure 1.2 20 Year Obesity Trends in the USA [3]... 2

Figure 1.3 Compression Gradient for Best Promotion of Blood Flow [4]................................ 3

Figure 2.1 Compression Sensing Garment, Prototype Rev. 1 6

Figure 2.2 Mechanical Pressure Gauge and Smartphone Interface for Pressure Readings 11

Figure 2.3 Pressure Comparison Test Results for Each Sensor Level on Prototype Build...... 12

Figure 2.4 Block Diagram Overview of System 15

Figure 3.1 The Wheatstone Bridge Circuit 17

Figure 3.2 Force Sensor Diaphragm Deformation for Force Sensing [8] 18

Figure 3.3 Wheatstone Bridge Circuit with External Amplifier Connected 20

Figure 3.4 Ideal Force vs. Voltage Plot with Zero Bias, Gain of 100, VDD = 3.3V 21

Figure 3.5 Comparison of Size, Force Application Between Sensors [8],[9] 23

Figure 3.6 Model and Dimensions of HSFPAR003A [8] 24

Figure 3.7 Model and Dimensions (in mm) of HSFPAR303A [9] 25

Figure 3.8 Force Sensor/Amplifier Circuit Schematic from DipTrace 26

Figure 3.9 Force Sensor/Amplifier Board Version 1 (Right) and 3D Model (Left) 27

Figure 3.10 Force Sensor Housing Mechanism 3D Model, Hidden Lines Showing 28

Figure 4.1 Visual Representation of Analog-to-Digital Conversion and the LSB Voltage 31

Figure 4.2 Data Processing Sequence 34

Figure 4.3 Conditional Statement to Take Care of Bias Voltage 36

Figure 4.4 Function to Calculate Force Applied from Sensed Voltage 37

Figure 4.5 Function to Calculate Pressure from Applied Force.. 39

Figure 4.6 Illustration of UART Connection Between MCU and BLE Module 40

Figure 4.7 UART Initialization Function 41

Figure 4.8 Complete Task Function, Completes Task According to Received Character....... 42

Figure 4.9 Main MCU/BLE Printed Circuit Board 3D Model... 43

Figure 4.10 HM-10 Bluetooth Low Energy Module and Pinout [6]... 44

Figure 4.11 HM-11 Bluetooth Low Energy Module and Pinout [6]... 45

Figure 4.12 AT Commands Syntax for Configuring Bluetooth Module 46

Figure 4.13 AT Commands Acknowledgement Message Format 47

Figure 4.14 Visual Diagram of Serial and Wireless Communication....................................... 48

Figure 4.15 Example of FSK Signals for Transmitting “1” or “0” 49

Figure 5.1 Smartphone App Home Screen and Technician Mode Screen, Designer View..... 52

Figure 5.2 BACK Button Logic from Technician Screen, Blocks View 53

Figure 5.3 Home Screen Breakdown by Section (Left – Disconnected, Right – Connected) . 54

Figure 5.4 Connecting to a Device Using CONNECT Button and Device List 55

Figure 5.5 Character Array Variable Declarations for Data Labels.. 57

Figure 6.1 3.7V, 380mAh Li-Po Rechargeable Battery 64

Figure 6.2 Drainage of 3.7V Li-Po 380mAh Battery Over Time, $I_{LOAD} = 50mA$.................... 65

Figure 6.3 4-Switch Mode Operation for LTC3531 Buck-Boost Regulators [7] 66

Figure 6.4 Efficiency vs. V_{IN} for LTC3531 Buck-Boost SPS [7], I_{LOAD} = 100 mA............... 67

Figure 6.5 Efficiency vs. Load Current for Different Input Voltages LTC3531 [7]................ 69

Figure 6.6 Battery Current Out vs. Battery Voltage, I_{LOAD} = 16.2 mA................................... 71

Figure 7.1 Pie Chart of Current Draw Distribution by System Component 82

Figure 7.2 Schematic Snippet Showing Unused ADC Input Channel Pins on MCU............. 85

Figure A.1 Main MCU/BLE PCB Schematic 88

Figure A.2 Force Sensor/Amplifier PCB Schematic 89

Figure D.1 MCU C Code for Data Reception, Processing, Transmission 97

Figure E.1 Blocks View for Home Screen Design, MIT App Inventor 98

LIST OF ABBREVIATIONS/ACRONYMS

ADC	Analog-to-Digital Converter
ASCII	American Standard Code for Information Interchange
BLE	Bluetooth Low Energy
BOM	Bill of Materials
CVI	Chronic Venous Insufficiency
DAC	Digital-to-Analog Converter
DC	Direct Current
DIP	Dual In-line Package
EMI	Electromagnetic Interference
EUSART	Enhanced USART
FHSS	Frequency-Hopping Spread Spectrum
FM	Frequency Modulation
FSK	Frequency-Shift Keying
FSR	Force Sensing Resistor
FVR	Fixed Voltage Reference
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
IC	Integrated Circuit
IDE	Integrated Development Environment
LED	Light Emitting Diode
LSB	Least Significant Bit
MCU	Microcontroller Unit

MD	Doctor of Medicine
MIT	Massachusetts Institute of Technology
NC	Not Connected
PC	Personal Computer
PCB	Printed Circuit Board
RN	Registered Nurse
SBC	Single Board Computer
SMD	Surface Mount Device
SMT	Surface Mount Technology
SOC	System-on-Chip
SOIC	Small-Outline IC
SPI	Serial Peripheral Interface
SPS	Switching Power Supply
SSOP	Shrink Small-Outline Package
TSSOP	Thin-Shrink Small-Outline Package
UART	Universal Asynchronous Receiver and Transmitter
UMC	University Medical Center
USART	Universal Synchronous/Asynchronous Receiver and Transmitter
USB	Universal Serial Bus
VSU	Venous Stasis Ulcer

LIST OF CONVERSION FACTORS

CURRENT

1000 mA = 1 A

1000 µA = 1 mA

VOLTAGE

1000 mV = 1 V

1000 µV = 1 mV

RESISTANCE

1000 Ω = 1 kΩ

1000000 Ω = 1 MEGΩ

PRESSURE

7500.62 mmHg = 1 N/mm^2

DATA

8 bits = 1 bytes

2^{10} bytes = 1 kilobytes (KB)

2^{20} bytes = 1 megabytes (MB)

2^{30} bytes = 1 gigabytes (GB)

CHAPTER 1: INTRODUCTION

1.1 THE PROBLEM

Venous stasis disease effects as many as 20 million people in the United States alone every year [4], and of those 20 million around 2.5 million (1 out of every 8) suffer from Venous Stasis Ulcers (VSUs). The ulcer itself is typically a red rash of irritated skin which eventually develops into a painful open wound. Some VSUs are treated with leg elevation and compression therapy (see Fig. 1.1 below), while in severe cases, surgery is necessary. Though any of these methods could be used on a case-by-case basis, compression therapy is the standard of care in the United States healthcare system. Compression therapy works because the compression of the fatty layer of skin on the patient's leg pushes the blood that has pooled up in artificial veins back into the deep venous system. Once blood is back into the deep venous system, it has no problem flowing back to the heart. Even after ulcers have healed, medical professionals recommend lifelong maintenance of compression therapy, which has proven to reduce to the risk of the recurrence of ulceration. The area most affected by venous skin ulcers is the lower leg, from the top of the foot to the middle of the shin, though the entire lower leg from below the knee to the foot is susceptible.

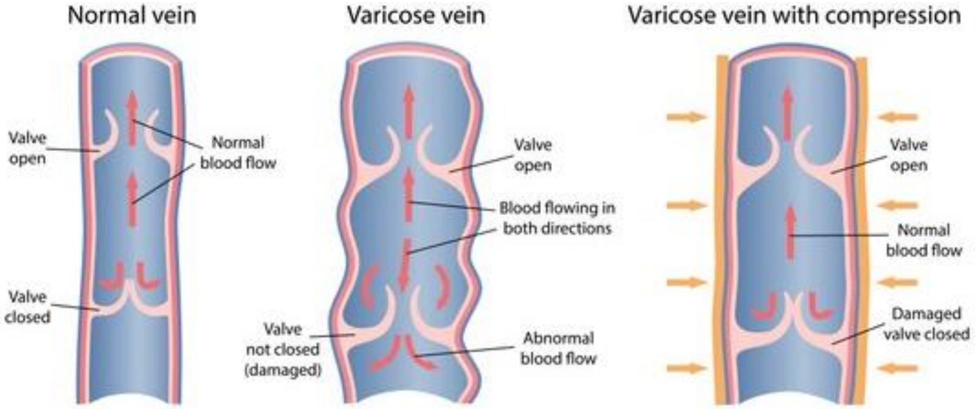


Figure 1.1 *Illustration of Compression Assisting a Varicose Vein [4]*

Venous stasis disease is also commonly known as chronic venous insufficiency, or CVI. Per Cleveland Clinical medical professionals, “chronic venous insufficiency (CVI) is a condition that occurs when the venous wall and/or valves in the leg veins are not working effectively, making it difficult for blood to return to the heart from the legs. CVI causes blood to ‘pool’ or collect in these veins, and this pooling is called *stasis*,” [1]. Figure 1.1 above shows what a normal vein looks like while it conducts the flow of blood back to the heart, alongside a varicose or *enlarged* vein with abnormal blood flow and pooling. As Dr. Debra Jaliman puts it in her article regarding varicose veins, “any condition that puts excessive pressure on the legs or abdomen can lead to varicose veins. The most common pressure inducers are pregnancy, obesity, and standing for long periods,” [2].

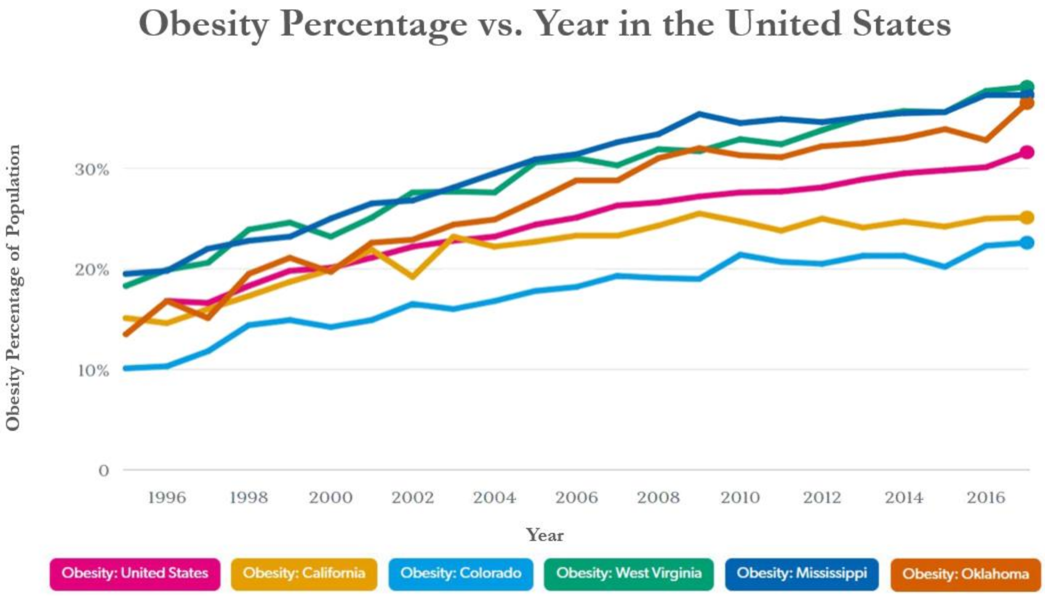


Figure 1.2 *20 Year Obesity Trends in the USA [3]*

Shockingly, according to data collected in July of 2020 [3], over two-thirds (about 69%) of adults in the United States are considered overweight. That translates to over 225 million adults in the US alone that are overweight or obese. Of the overweight portion of the population, over

52% are considered obese. That means 36.5% of the US adult population is considered obese, or more than 1 in every 3 adults. Of the younger population in the US, more than 12.7 million children ages 2 to 19 are obese, or 17% of American children. Aside from the physical difficulties obesity poses, people who are obese have a greatly increased risk for a variety of chronic diseases and other health complications, including diabetes and, of course, venous stasis (or CVI). The plot of the obesity trends in the United States from 1996 through 2016 is shown on the previous page in Fig. 1.2. A brief analysis of the overall trend of the country (pink trace) is quite convincing that the problem of obesity in the country does not seem to be vanishing any time in the near future. Since CVI and diabetes are directly linked to obesity, it is safe to say they are here to stay for a while as well.

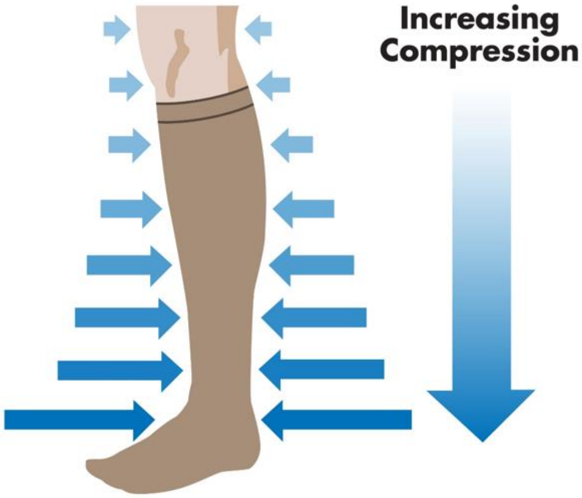


Figure 1.3 *Compression Gradient for Best Promotion of Blood Flow [4]*

While health officials and experts continue to promote good health and attempt to reverse the trends of Fig. 1.2, medical experts are consumed with the issue of treatment and prevention of the resulting physical complications. As mentioned previously, compression therapy is the standard of care for VSUs. Again, compression therapy pushes the blood that has pooled up in spider or varicose veins back into the deep venous system so that it can properly flow back to the heart. The fastest healing times for VSUs are achieved when an increasing gradient of pressure is applied to the patient’s leg, with the lowest pressure at the knee and the highest pressure at the foot, as seen in Fig. 1.3 above. The highest pressures, applied at the ankle and foot area, should nominally be between 30-50 mmHg and decrease linearly down to around 6 mmHg just below the knee [4]. Medical experts predict that the application of such a gradient could decrease the time required for ulcers to heal from up to 6 months down to as little as 60 days, while also assisting in the prevention of recurrence.

Though this method of applying a linear pressure gradient works very well in promoting the best blood flow, applying a known amount of absolute pressure to the leg, or to anything for that matter, is not such a trivial task. Modern treatment consists of the application of passive wraps and compression sleeves to the patient’s leg with no knowledge of exactly how much pressure is being applied to the leg. Though this is the standard of treatment, “no study has shown that application of wrapped compression garments can obtain specific pressures with accuracy or precision, and none have shown that those providers applying bandages can reliably adjust for specific situations,” [4]. A pilot study conducted at UMC proved this delivery of inaccurate levels of pressure. Results of the study showed that registered nurses (RNs) and residents both applied high pressures (often above 100 mmHg) when applying passive bandages and wraps. Some RNs practice placing their own finger under the wrap to assure they do not apply too much pressure, and ironically these same RNs overtightened even more often, while also neglecting to apply any sort of gradient. The overpressure application results in longer healing time, as the high pressure applied does not allow blood to flow back to the heart as it should. The RNs themselves were stunned by the results, making such comments as, “I can’t believe I’ve been doing this incorrectly for so long,” [4].

1.2 THE PROPOSED SOLUTION

From section 1.1, knowledge of the pressure being applied to the leg is desired, if not essential, to the effective treatment and prevention of VSUs. The proposed solution to the problem is a thin pressure-sensing garment to go underneath a compression sleeve or wrap to sense the amount of pressure being applied to the leg by the wrap. Since the pressure sensors directly contact the skin, the pressure applied to the sensors is theoretically near equal to the pressure applied to the leg itself, with the difference in the actual and measured pressures being negligible. The sensing device optimally has sensors at four levels: the upper calf, the lower calf, the lower leg, and the insole of the foot. Pressure sensors at different levels allows for the assurance of the application of the correct gradient. The sensed pressure values are processed by a microcontroller and transmitted wirelessly to a smartphone via Bluetooth, and the data is viewable on a smartphone application. The smartphone application can also be used to send commands to the microcontroller and request different data that is not displayed on the home screen of the application. The chapters to follow detail the design of a power-efficient fully functional prototype of the proposed solution which is capable of accurately sensing pressure applied to a patient’s leg with a maximum error of around 5 mmHg, a typical error of around 2.5 mmHg, and lasting roughly 8 hours running continuously on a single battery charge.

CHAPTER 2: SYSTEM DESIGN OVERVIEW

2.1 PROTOTYPE BUILD

The prototype of the proposed solution which was built is displayed in the image below in Fig. 2.1. The prototype has sensors at the upper calf, lower calf, and lower leg, but excludes the sensor at the insole of the foot, since the Velcro calf sleeve does not cover the foot. The entire system is powered by a 3.7V, 380mAh Lithium-Polymer rechargeable battery that can power the system for about 4 hours. The main circuit board contains power regulation circuitry to regulate the battery voltage to 3.3V to power the microcontroller and the Bluetooth module with 3.3V DC. There are 3 wires routed from the main board to each of the sensing devices for a total of 9 wires. The red wires are 3.3V power to the sensors and amplifiers on the sensing devices. The black wires are the circuit ground. The yellow wires connect the output of each sensing device to the microcontroller's on-chip ADC input channels for data conversion and processing.

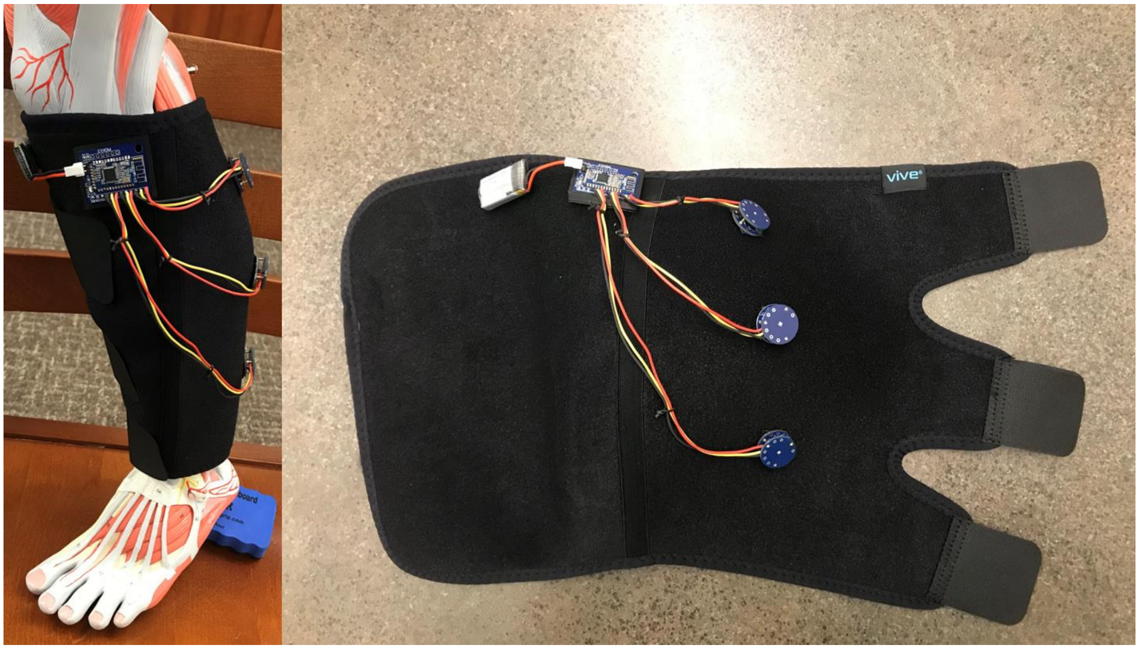
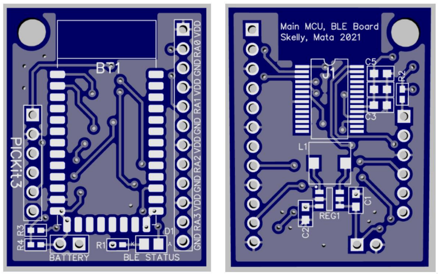
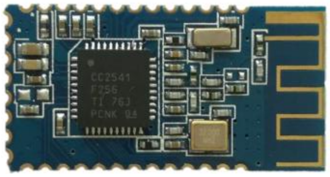
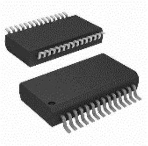
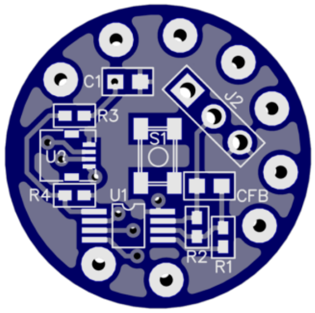


Figure 2.1 *Compression Sensing Garment, Prototype Rev. 1*

The table below provides a bill of materials (BOM) or parts list for the prototype build described previously. The part pictures are not to scale. Part dimensions are discussed later in the chapters concerning these parts.

PART DESCRIPTION	PART PICTURE	QUANTITY
MAIN MCU/BLE PCB	 BT1 PICKit3 BATTERY BLE STATUS Main MCU, BLE Board Skelly, Mata 2021 J1 L1 REG1	1
Consists of the regulator circuit, the MCU, the Bluetooth module, programming pins, and connections out to each of the sensor/amplifier boards. (Microcontroller Unit – MCU)		
HM-10 BLUETOOTH LOW ENERGY MODULE		1
Receives serial data from the MCU and transmits that data wirelessly to the phone. (Bluetooth Low Energy – BLE)		
BLUETOOTH STATUS LED 0805 (BLUE)		1
Indicates Bluetooth connection status. (Blink – Disconnected | Solid – Connected)		
LTC3531-3.3 BUCK-BOOST REGULATOR IC		1
Regulates VDD at 3.3V from battery voltage.		
LFT4022T-100M-D 10µH INDUCTOR		1
Used for energy storage in the buck-boost voltage regulator circuit.		
PIC18LF26K22 MICROCONTROLLER		1
Processes all data and all user commands, transmits requested data to the BLE module.		
PASSIVES: CAPACITOR 0603, VALUES VARY		7
Various capacitors used for decoupling at the microcontroller and regulator ICs.		

PASSIVES: RESISTOR 0402, VALUES VARY Various resistors used for current limiting and gain setting of the amplifier.		8
TENERGY 3.7V 380MAH LIPO BATTERY Powers all the electronics in the system, lasts for about 4-5 hours in the prototype setup and is rechargeable.	TENERGY 31496 3.7V 380mAh 25C Li-Polymer Rechargeable Battery	1
2 X 100 MIL RIGHT ANGLE MALE HEADER Mounted to the main control board for easy connection to the battery's power and ground wires through the connector.		1
COPPER WIRE, RED, STRANDED Connects 3.3V VDD from the main control board to each of the sensor/amplifier boards to power the sensors and amplifiers.		3
COPPER WIRE, BLACK, STRANDED Connects ground of the sensor/amplifier boards to the ground of the main control board.		3
COPPER WIRE, YELLOW, STRANDED Connects the amplified output of the sensor/amplifier board to its ADC input channel on the microcontroller board.		3
SENSOR/AMPLIFIER PCB Consists of the force sensor, amplifier, and a number of passives. Circuit is used to sense an applied force and amplify the signal prior to ADC conversion. Also contains through holes around the rim for soldering the base of the pogo pins.		3

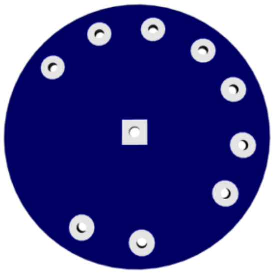
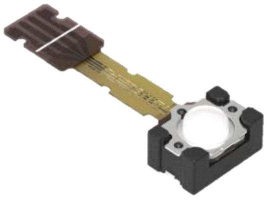
SENSOR MECHANISM HOUSING PCB		3
Consists of holes for soldering the tops of the pogo pins and one centralized pad for the turret to be soldered to. No copper pours or traces are on this board, is simply for mechanical stability while sensing.		
TERM TURRET SINGLE L=1.79MM		3
Makes physical contact with the force sensor actuator, assuring that force is not applied at a damaging angle.		
HSFPAR303A FORCE SENSOR		3
Force sensor which outputs a linear voltage difference for linear changes in force applied to the actuator. Output signal amplitude is small and needs to be amplified.		
MAX4208 INSTRUMENTATION AMPLIFIER		3
Very low offset instrumentation amplifier with settable gain, used to amplify the sensor output voltages prior to ADC conversion on the MCU.		
GOLD-PLATED COPPER POGO PIN CONN.		9
Connected between the sensor/amplifier PCB and the mechanism housing PCB to provide spring action between the two boards. Assures no pressure reading when no pressure is being applied.		
VIVE LOWER LEG COMPRESSION WRAP		1
Holds the battery, the main board, and all the sensor boards using Velcro for position adjustability. Velcro straps at three levels for tightening and loosening on the patient’s leg.		

Table 2.1 *List of Parts for Prototype Build*

The prototype build is only used for testing the operation of the sensors and proof of concept for the final build. Chapter 7 extensively discusses the tradeoff decisions and improvements that could be made at each different stage of the system. However, it is important here to introduce a few key improvements that need to be made in order for the design to effectively solve the problem posed in section 1.1. First, medical professionals prefer for there to be foot coverage since the highest pressures need to be applied to the top of the foot and ankle area [4]. The prototype build does not cover the foot. Simply adding another sensing mechanism and three more wires from the main board poses no difficulty, but the garment itself needs to be improved to cover the foot while still being adjustable in size (cannot be a sock for compliance reasons). Next, the sensing mechanism needs to be designed smaller; both the height and the diameter need to decrease significantly. In order for the pressure to be properly sensed, the height of the sensing device should be minimized. The area of the PCB should also be minimized for the sake of the patient's comfort and compliance. Other key improvements are related to device aesthetics, such as hiding the battery and wired connections, or at least making them less noticeable or visible.

2.2 TESTING AND RESULTS

Though plenty of improvements are necessary for the production of an effective, market-ready device, preliminary testing shows the prototype build performs well against a blood pressure cuff for measurement comparison. The blood pressure cuff used is a *Greater Goods Sphygmomanometer Manual Blood Pressure Monitor Kit* which uses a hand pump to both pressurize the cuff and read the pressure back to the user in units of mmHg. Since the prototype is designed to also read pressure in units of mmHg, the pressure measurement of the hand pump gauge can be directly compared to the pressure measurement of the prototype which is displayed on the smartphone application home screen once the device is connected to the phone via Bluetooth. To perform the pressure comparison tests, the prototype sleeve was wrapped loosely around a human leg, and the blood pressure cuff was wrapped snug around the leg as well, on top of the prototype sleeve and covering just one sensor at a time. Three separate plots were generated (one for each sensing mechanism) comparing the pressure read by the gauge with the pressure displayed on the smartphone application. The left side of Fig. 2.2 below shows the mechanical pressure gauge from the sphygmomanometer and the right side of the figure shows the smartphone with pressure values ready to be sensed.

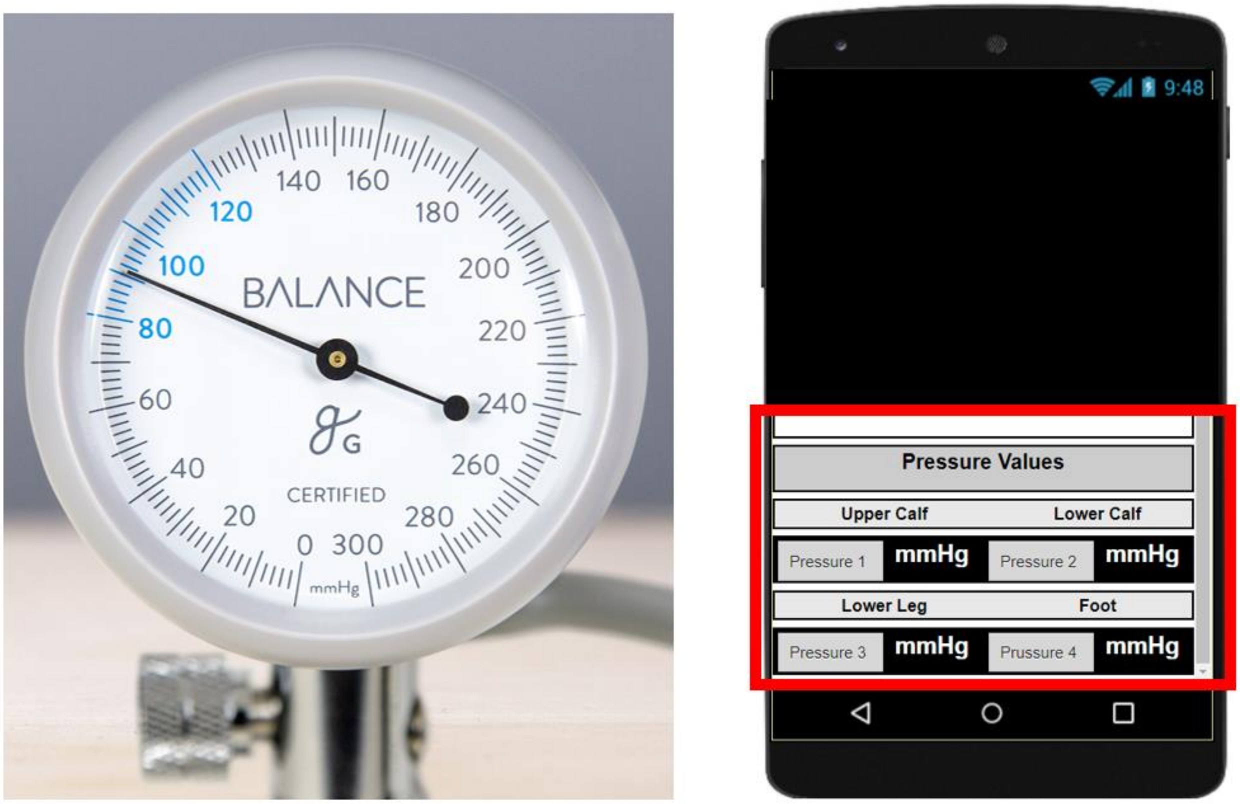


Figure 2.2 *Mechanical Pressure Gauge and Smartphone Interface for Pressure Readings*

The data was obtained visually and noted in an excel file at 2 mmHg decrements starting at an initial pressure of 40 mmHg and letting air out little by little to decrease the pressure applied.

A best fit line (red) was generated using least squares fit of the data coming from the smartphone display for each sensor. The following plots were obtained:

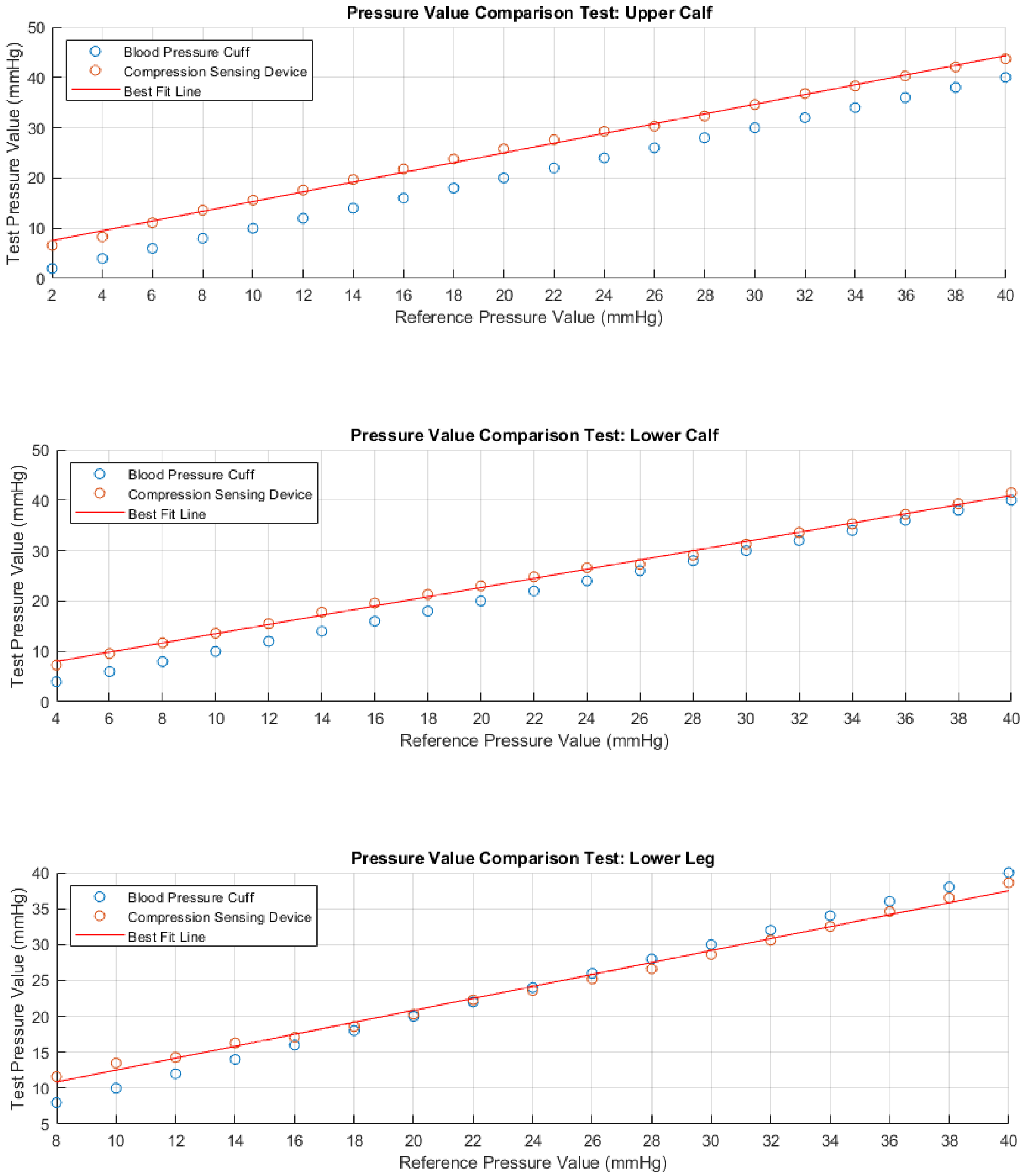


Figure 2.3 *Pressure Comparison Test Results for Each Sensor Level on Prototype Build*

In the plots above, the blue data points represent the reading on the pressure gauge, and the orange data points represent the reading on the smartphone display from the prototype. The pressure gauge readings (blue data points y-value) are always equal to reference pressure value (ticks on the x-axis) because the point at which the pressure reading from the smartphone is noted is directly dependent upon the pressure reading on the gauge in multiples of 2 mmHg. Thus, the gauge pressure is literally the reference pressure (line with a slope of 1, $y = x$), and the testing is used to check just how close or "accurate" the prototype pressure reading is to the gauge pressure readings over a span of about 40 mmHg, the sensing range of interest for treating VSUs.

SENSOR LEVEL	BEST FIT SLOPE	AVERAGE ERROR	IN-SAMPLE ERROR
Upper Calf	0.9676	+ 5.0 mmHg	25.1 mmHg
Lower Calf	0.9146	+ 2.5 mmHg	7.3 mmHg
Lower Leg	0.8325	– 0.2 mmHg	3.1 mmHg

Table 2.2 *Test Data Results Analysis and Error Metrics*

The table above gives some metrics for analyzing the plots of Fig. 2.3 shown previously. These metrics include the slope of the best fit line, the average error, and the in-sample error. The slope (also called *rate of change*) of the best fit line gives an idea of how accurately the sensing device measures *changes* in applied pressure. (Recalling that the reference line has a slope of 1) The closer the best fit line slope is to a value of 1, more accurately the sensing device measures changes in applied pressure. Thus, from the data, we can see that the sensor at the upper calf most accurately senses the changes in applied pressure, while the sensor at the lower leg is the least accurate at sensing changes. The second metric is the average error. This metric gives insight about the magnitude of the offset or actual difference (on average) between the gauge pressure and the sensing device pressure readings. The average error is found by simply subtracting the reference pressure or the *expected* pressure value from the obtained or *actual* pressure value to obtain a difference. The differences from each measurement are then *averaged* or summed and then divided by the total number of data points. Note that if, on average, the actual pressure value is *less than* the expected pressure value, the average error will be negative. If the actual pressure value is *greater than* the expected pressure value, the average error will be positive. The final metric in the table is the *in-sample error.* The in-sample error is a different way to measure error in which the sign of the error is neglected by squaring the difference between the *expected* and *actual* pressure readings. The average of these squared differences is then obtained and called the in-sample error.

The plots obtained from testing show that the prototype sensing units have linear behavior over a repeated number of trials and across three independently constructed sensing units. The trend of the sensed data matches the reference trend very closely for all three sensing levels, proving that each of the three sensing devices can accurately sense changes in pressure in the range of interest. The current problem of overpressure application by residents and RNs in the medical field proved to show pressures applied to the leg exceeding 100 mmHg in many cases. Since the maximum pressure applied should not exceed ~50 mmHg, this is more than a 100% percent error in the pressure applied by wound care professionals. With the sensing device beneath the compression wrap having a maximum error of 6 mmHg, the **worst case** for overpressure application would be 56 mmHg applied to the patient’s leg, or a 12% error. Thus, even the first revision of the sensing device **at the very worst** reduces the error in some cases by at least 88%.

2.3 FROM SENSOR TO SMARTPHONE

This section provides an overview for the data path from the sensing performed by the force sensor to the display of the processed data on the smartphone application. Figure 2.4 shows a block diagram overview of the entire system. The direction of the arrows provides a guide for which direction the data can travel between two components of the system. For example, data travels *from* the force sensor *to* the amplifier, but not the other way around. There are two arrows (one in each direction) between the MCU (Microcontroller Unit) and the BLE (Bluetooth Low Energy) module, as well as between the BLE module and the smartphone. This indicates that data can travel in either direction between the two components of the system. The arrows do not necessarily indicate a wired connection. For example, the BLE module and the smartphone communicate wirelessly.

Starting from the left side of the figure, force is first applied to the sensor's actuator, and the output voltage of the sensor is then amplified by an instrumentation amplifier. The amplified voltage is converted from analog to digital by the on-chip ADC of the microcontroller. The microcontroller processes the data and performs mathematical operations on the voltages to calculate the force applied to each sensor in Newtons, and the associated pressure in mmHg. The MCU then sends the data serially to the Bluetooth module using UART (Universal Asynchronous Receiver/Transmitter), and the Bluetooth module finally transmits the received data wirelessly to the smartphone application for the user to view and analyze.

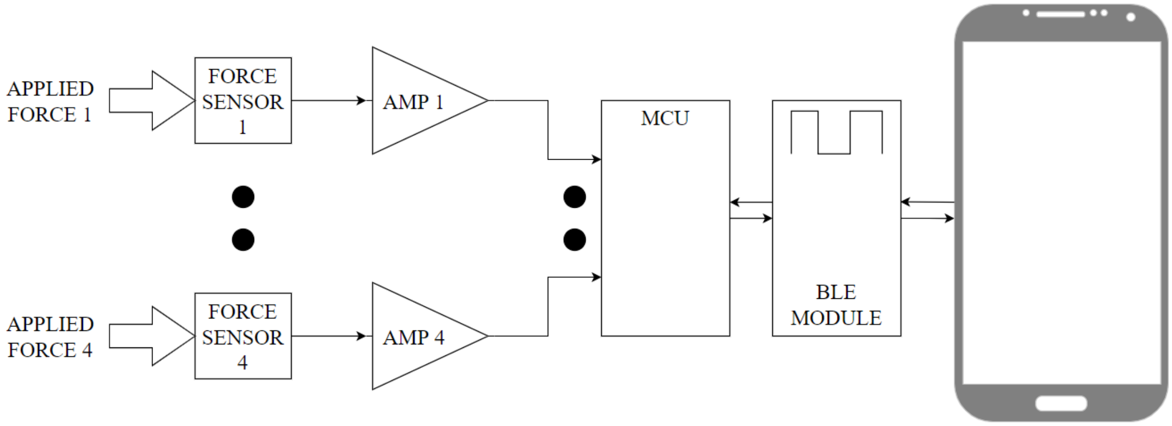


Figure 2.4 *Block Diagram Overview of System*

2.4 CHAPTER 2 SUMMARY

Chapter 2 provides a detailed overview of the system, including a comprehensive parts list from the prototype build with descriptions of each component, along with test results and data analysis.

- **Section 2.1** breaks down the hardware component of the prototype build and provides a brief discussion of future improvements for the device, which is later expanded upon in Ch. 7.
- **Section 2.2** details the testing process for determining the accuracy of the device pressure readings against a blood pressure cuff (sphygmomanometer). Plots are given to provide a visual aid for the accuracy of the device measurements, along with calculated error metrics for each plot.
- **Section 2.3** is a brief overview of the path data takes from the applied force at the sensing mechanism to the pressure displayed on the smartphone application. Data transfer and processing are expanded upon in Ch. 4.

CHAPTER 3: SENSING

3.1 SENSING FORCE

By definition, pressure is the force applied to an object per unit area over which the force is applied, or

$$P = {}^{F}\!/_{A} \tag{3.1}$$

where ***P*** is pressure, ***F*** is force, and ***A*** is the area over which the force is applied or distributed. From equation (3.1) it is apparent that in order to determine pressure, it is important to first determine the precise value of an applied force, and then determine exactly what area that force is distributed over. Thus, the first task for the system is to sense the force being applied to the leg at each of the four sensor levels. The system uses the *HSFPARx03A* force sensor (manufactured by Alps Alpine) to sense force using a Wheatstone bridge circuit, see Fig. 3.1 below.

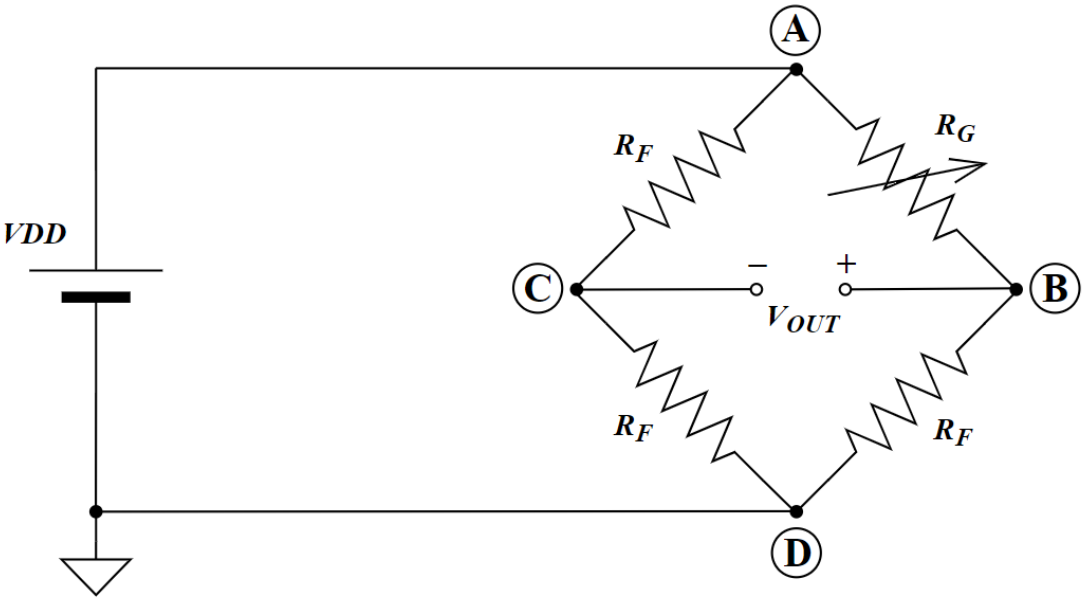


Figure 3.1 *The Wheatstone Bridge Circuit*

The Wheatstone bridge circuit consists of two resistive voltage dividers. From the figure, it should be noted that 3 of the 4 resistors are of equal value, R_F, while the fourth resistor has a variable resistance, R_G. This variable resistor can be replaced with a variety of materials so that the Wheatstone bridge circuit can be used to sense many different interesting metrics. However, in the *HSFPARx03A* family of force sensors, the variable resistor is simply a piezo-resistive element whose resistance *decreases* when a force is applied to the sensor. The figure below is from the device datasheet and shows how the diaphragm deforms with applied force. This deformation of the diaphragm as force is applied results directly in a linear decrease in resistance for R_G.

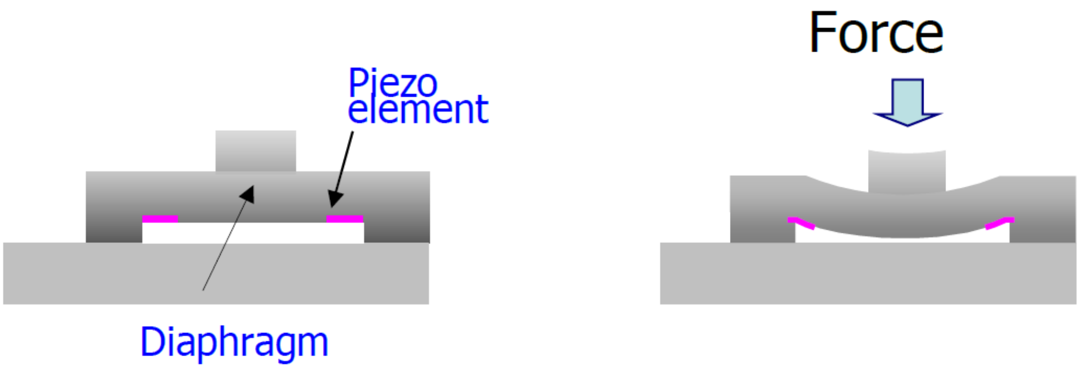


Figure 3.2 *Force Sensor Diaphragm Deformation for Force Sensing [8]*

In the circuit of Fig. 3.1 on the previous page, before any force is applied to the sensor, ideally, the voltage at point B is equal to the voltage at point C. The path from point A to point C to point D makes up a resistive voltage divider where point C would have a voltage of ***VDD/2.*** The path from point A to point B to point D makes up a separate resistive voltage divider in parallel with the first, where point B would also have a voltage of ***VDD/2***, assuming R_G has a resistance equal to R_F before any force is applied to the sensor. These voltages can be calculated using the voltage divider equation, since the two voltage dividers are connected in parallel. Calculating the voltage at point C first,

$$V_{C,init} = VDD\left(\frac{R_F}{R_F + R_F}\right) = VDD\left(\frac{R_F}{2R_F}\right) = VDD/2 \tag{3.2}$$

Calculating the voltage at point B, let $\boldsymbol{R_{Gi}}$ be the resistance of $\boldsymbol{R_G}$ before any force is applied. Then,

$$V_B = VDD\left(\frac{R_F}{R_{Gi} + R_F}\right) \tag{3.3}$$

Assuming, for proof of the concept, an ideal system where the initial resistance $\boldsymbol{R_{Gi}}$ is equal to that of $\boldsymbol{R_F}$, then

$$V_{B,init} = VDD\left(\frac{R_F}{R_F + R_F}\right) = VDD\left(\frac{R_F}{2R_F}\right) = VDD/2 \tag{3.4}$$

Thus, it is proven by equations (3.2) and (3.4) that initially, $\boldsymbol{V_B}$ and $\boldsymbol{V_C}$ are equal. Since $\boldsymbol{V_{OUT}}$ is the difference between $\boldsymbol{V_B}$ and $\boldsymbol{V_C}$, the initial value of $\boldsymbol{V_{OUT}}$ is calculated as

$$V_{OUT,init} = V_{B,init} - V_{C,init} = 0V \tag{3.5}$$

Taking another look at equation (3.3), the only variable is $\boldsymbol{R_G}$, the piezo-resistive element. Recalling that the resistance $\boldsymbol{R_G}$ of the piezo-resistive element *decreases* as the force applied *increases*, the operation of the sensor is characterized by the increase in output voltage as force is applied to the sensor. Since V_C is fixed, any increase in the voltage V_B directly results in the same increase in the voltage V_{OUT}. A linear force-voltage relationship is observed based on the sensitivity of the force sensor (3.7mV/V/N) and the initial output voltage with no force applied to each sensor.

3.2 AMPLIFYING THE SENSOR OUTPUT VOLTAGE

The output voltage signal V_{OUT} from the sensor of Fig. 3.1 is a small signal relative to the supply voltage, with a sensitivity of just 3.7mV/V/N. To obtain the optimal results from the ADC, the signal output is amplified using an instrumentation amplifier. The *MAX4208* [11] instrumentation amplifier is chosen for this application because of its very small input offset voltage (±20µV max). Other standard operation amplifiers can have input offset voltages in the 10's of millivolts range, which in this case would dominate the small signal output of the sensor. Figure 3.3 below shows the Wheatstone bridge circuit of the force sensor from Fig. 3.1, but with the outputs of the bridge circuit connected to the inputs of the instrumentation amplifier. Resistors R_1 and R_2 form a voltage divider whose output is fed back to the amplifier to set the gain.

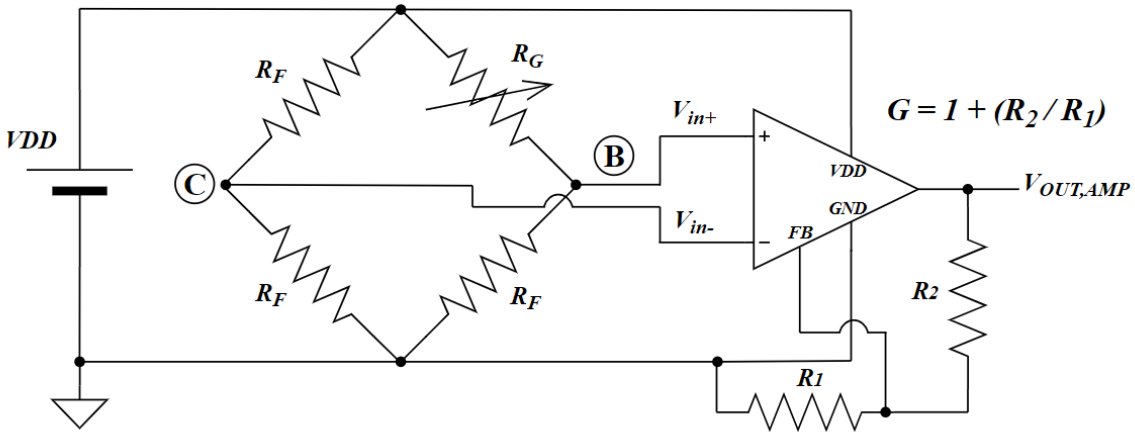


Figure 3.3 *Wheatstone Bridge Circuit with External Amplifier Connected*

Figure 3.4 below shows what the force-voltage relationship would look like with V_{DD} = 3.3V, an amplifier gain of 100, and a bias voltage (the voltage output V_{OUT} of the sensor with 0N force applied) of 0V. A bias voltage of 0V is only obtained if the initial resistance of the piezo-resistive element R_{Gi} is equal to that of R_F. For this application, the bias for the force sensors used is typically observed to be less than 0V due to mismatch between the piezo-resistive element and the other resistors in the bridge circuit. However, for the sake of explanation and proof of concept for the sensor's operation, the bias is assumed to be 0V.

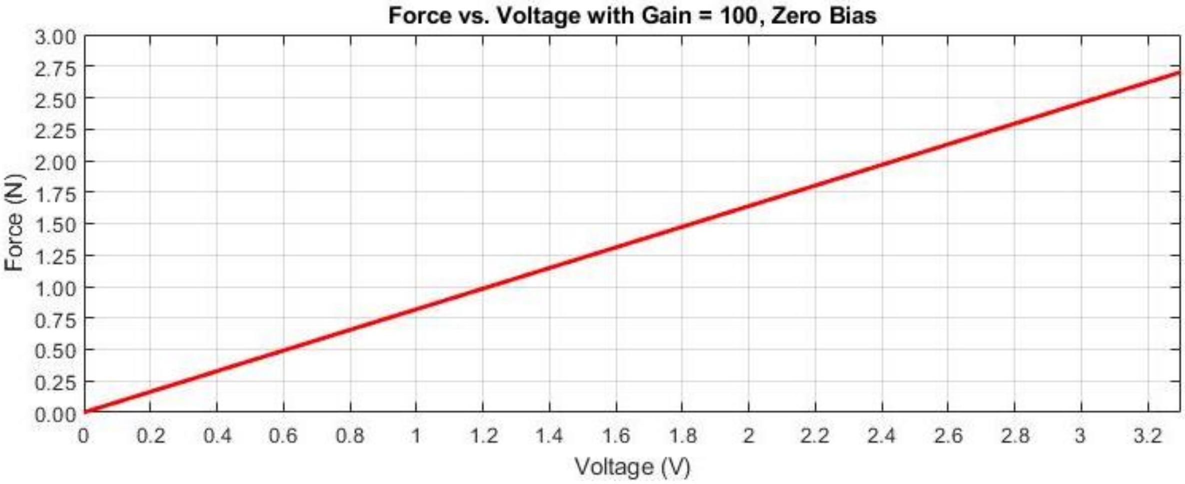


Figure 3.4 *Ideal Force vs. Voltage Plot with Zero Bias, Gain of 100, VDD = 3.3V*

The *HSFPARx03A* family of force sensors has two different types of force sensor: *HSFPAR003A* and *HSFPAR303A*. Moving forward, each sensor will only be referred to by the last four characters of its part number. Each of the two sensors has a force range with a minimum of 0N and is capable of sensing forces as small as 0.01N. The *003A* has a maximum force rating of 8N, while the *303A* has a maximum force rating of 7N. Regardless which sensor is used, the gain of the amplifier should be selected such that its output voltage will not rail (reach the power supply voltage level) so long as the force applied to the sensor is within the necessary force range for the application. It is also important to be mindful of the ADC's least significant bit voltage (or LSB voltage) when deciding how to set the amplifier gain. The LSB voltage of an ADC is the minimum resolution of the ADC's digital output signal. The ADC on the *PIC18LF2X/4XK22* family of microcontrollers (the MCUs used in the design of this device) is a 10-bit ADC. The LSB voltage of an N-bit ADC can be calculated by

$$V_{LSB} = {}^{VDD}/_{2^N} \tag{3.6}$$

where ***VDD*** is the supply voltage to the microcontroller and ***N*** is the number of bits. Thus, from equation (3.6), the relationship of the LSB voltage to the number of bits is made clear. The resolution of an ADC gets better (the LSB voltage gets smaller) as the number of bits, ***N***, increases. The operation of the ADC and the importance of the LSB voltage will be further discussed in Ch. 4. Since the device is designed to accurately sense pressures up to around 40mmHg (0.5N/cm^2), the gain of 100 above is practical. The gain could be decreased if it were desirable to allow for higher pressures to be sensed during ambulatory states for the patient.

3.3 FORCE SENSOR CANDIDATES

The most efficient method of correctly determining *absolute* pressure is to first measure force and then calculate pressure from force and area. Absolute pressure is opposed to relative pressure, where a pressure is measured "relative" to some other reference of the same medium, be it air, water, etc. As mentioned previously, the first challenge is to correctly sense the force being applied. With the challenge of force sensing comes the secondary challenge of keeping the cost low. Many force sensors exist that have large, robust casings around them and are fairly simple to power up and use. However, finding force sensors that are simple to use, robust, accurate, and inexpensive is a difficult task. Since the device uses a total of four sensors per unit, the cost of a sensor is four times more important. Nonetheless, the force sensors detailed in this section are relatively cheap (less than $10 per unit) and provide a linear increase in output voltage with linear increases in applied force.

Figure 3.5 below shows what the each of the *x03A* force sensors looks like, along with the principle of their operation. It is important to note that the manufacturers specify how a force needs to be applied by a body whose surface area is greater than that of the sensor's actuator. The *003A* has a much smaller actuator area than the *303A*, so a much smaller body can be used to apply force to the sensor. It is also important to note that the *303A* device is simply the *003A*, but with a casing built around it for overforce protection, and a rubber actuator to provide a larger surface area for force application. Though either device would work well in the sytem, each has its own pros and cons, and requires different external hardware in order to properly operate.

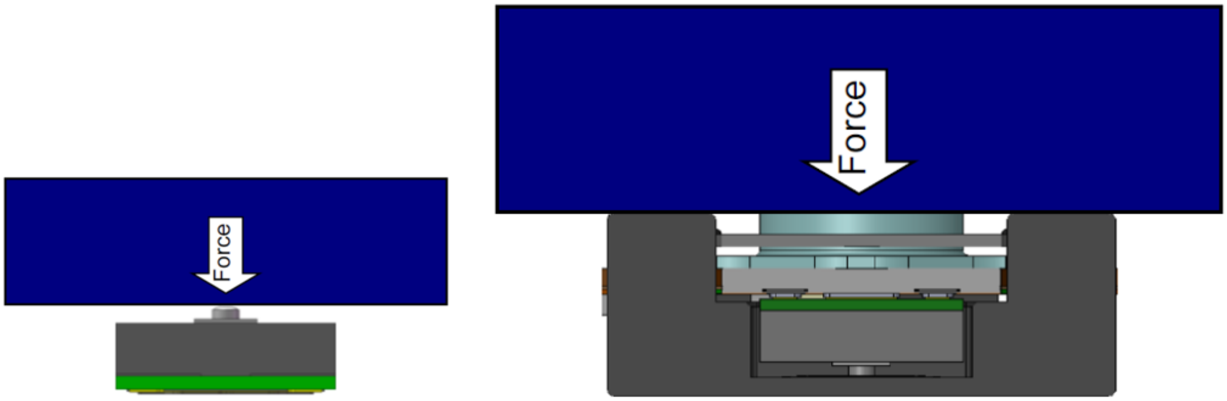


Figure 3.5 *Comparison of Size, Force Application Between Sensors [8],[9]*

The left side of Fig. 3.5 shows the *003A* with a block applying a force to its actuator. According to the device datasheet, applying a force at an angle greater than 5° can damage the sensor due to the delicacy of the very small, low profile actuator. This is the main drawback of the *003A*. Despite this one issue, the *003A* boasts a number of benefits. One major benefit is the extremely small size of the sensor. The figure below shows the dimensions of the sensor, including both the body and the actuator. The sensor also has surface mount pads beneath it so it can easily be reflow soldered to a PCB for ease of high-volume manufacturing, and signals can be routed directly into the PCB and out through a connector for further processing on the MCU.

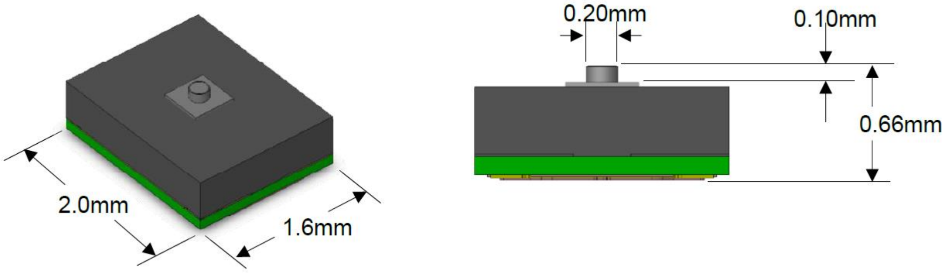


Figure 3.6 *Model and Dimensions of HSFPAR003A [8]*

The *303A* similarly has some benefits, but many more drawbacks than its counterpart. Because it has a protective casing surrounding the sensing device and a larger actuator, an obvious downside to the *303A* is its larger area and taller profile. The larger area means that it requires a larger PCB to mount to, and the taller profile means that the mechanical apparatus surrounding the sensor (discussed in detail later in this chapter) needs to be taller or thicker. Further, the device casing does not come with the convenient surface mount pads for routing signals out of the sensor. Rather, the signals are routed out with a long flex PCB which requires an external connector to access the signals. This connector adds cost, and the *303A* (around $9 per unit) is already nearly twice as expensive as the *003A* (around $5 per unit). Not only is cost an issue with the *303A*, but the flex PCB also adds to the total length of the sensor. Since the sensing device needs to be in the center of the PCB it is mounted to, the entire length of the flex PCB proportionally adds to the required radius of the board that houses the sensor. Lastly, the *303A* cannot be soldered to a board, but rather needs to be epoxied or ahered to the board using a different means than soldering, since it is a plastic case with no exposed pads for soldering.

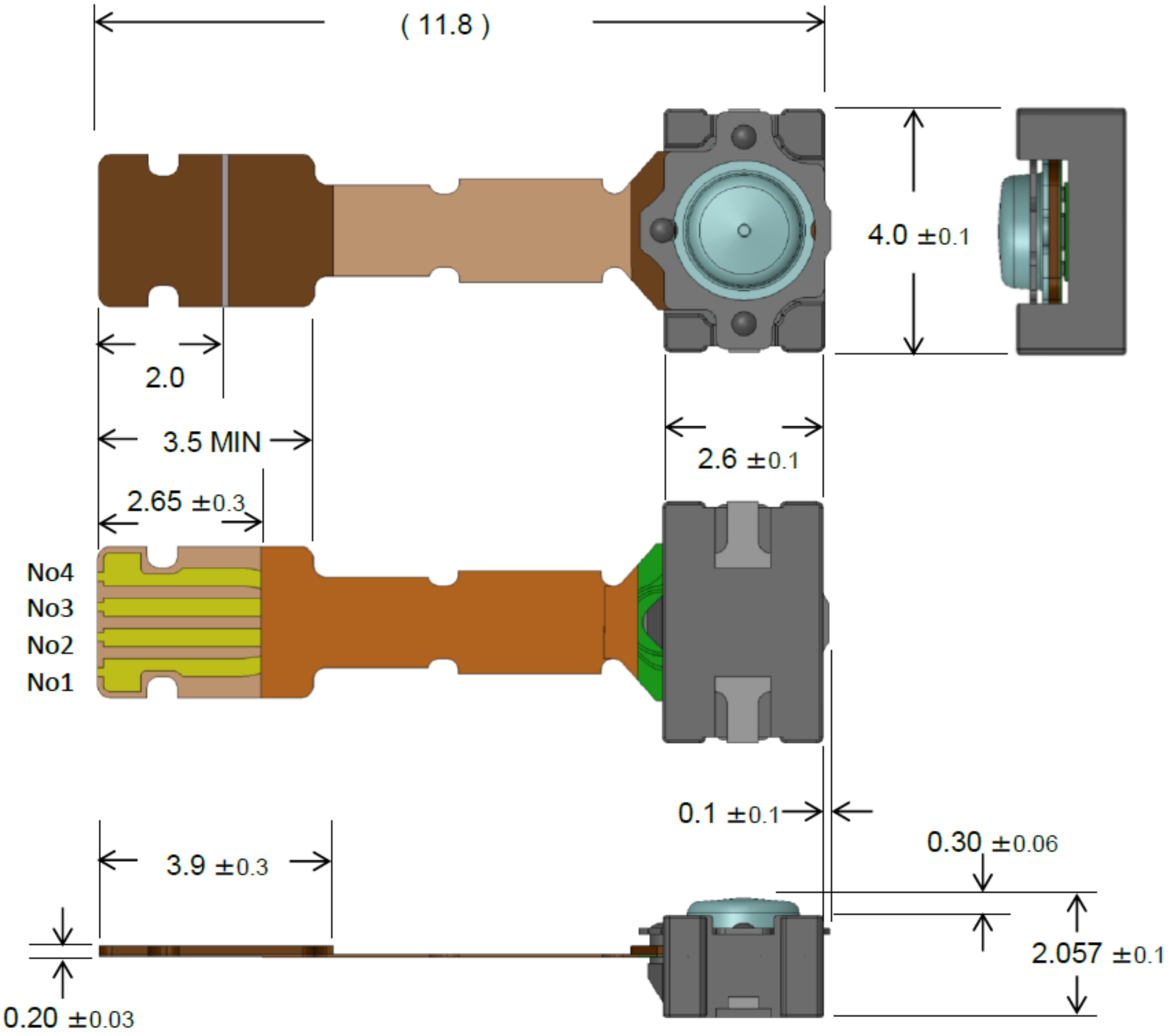


Figure 3.7 *Model and Dimensions (in mm) of HSFPAR303A [9]*

Figure 3.7 above shows a model of the *303A* along with dimensions of the device. No dimensions are shown for the actuator, but it is measured to have a diameter of about 2 mm.

Though a plethora of drawbacks are mentioned for the *303A*, there are still some benefits of the device. One of the benefits is that the surrounding case and overforce protection mechanism protect the sensing device (the *003A* inside) from damage. The rubber actuator also prevents forces applied at an angle from harming the actual actuator of the *003A* inside. Aside from these benefits, the *003A* has the *303A* beat in terms of cost, size, and ease of production.

3.4 FORCE SENSOR/AMPLIFIER CIRCUIT BOARD DESIGN

To limit the size of all the boards used in a single unit of the compression sensing device, the amplifier is placed on the same circuit board as the force sensor. Each of the four force sensors in the system and their associated amplifier circuits have their own individual boards. Thus, the device consists of 5 total PCBs: 4 force sensor/amplifier boards, and 1 main board consisting of the voltage regulation circuitry, the microcontroller, and the bluetooth module. Figure 3.8 above shows the schematic of the force sensor/amplifier circuit board. The board consists of the sensor,

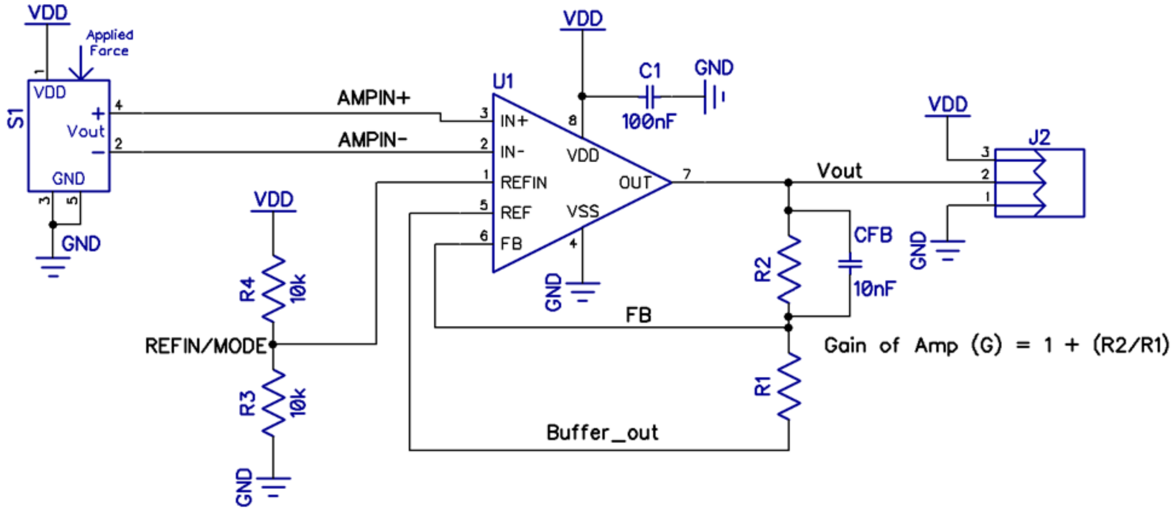


Figure 3.8 *Force Sensor/Amplifier Circuit Schematic from DipTrace*

the amplifier, the external passive components required for stability (and to set the gain of the amplifier), and a 3-pin 0.1” header to connect power, ground, and the voltage output of the sensor up to the main board. The first revision of the circuit board can be seen in the figure below. The left side of the figure shows the 3D model of the board from the DipTrace 3D Viewer, while the right side of the figure shows the actual fabricated PCB.

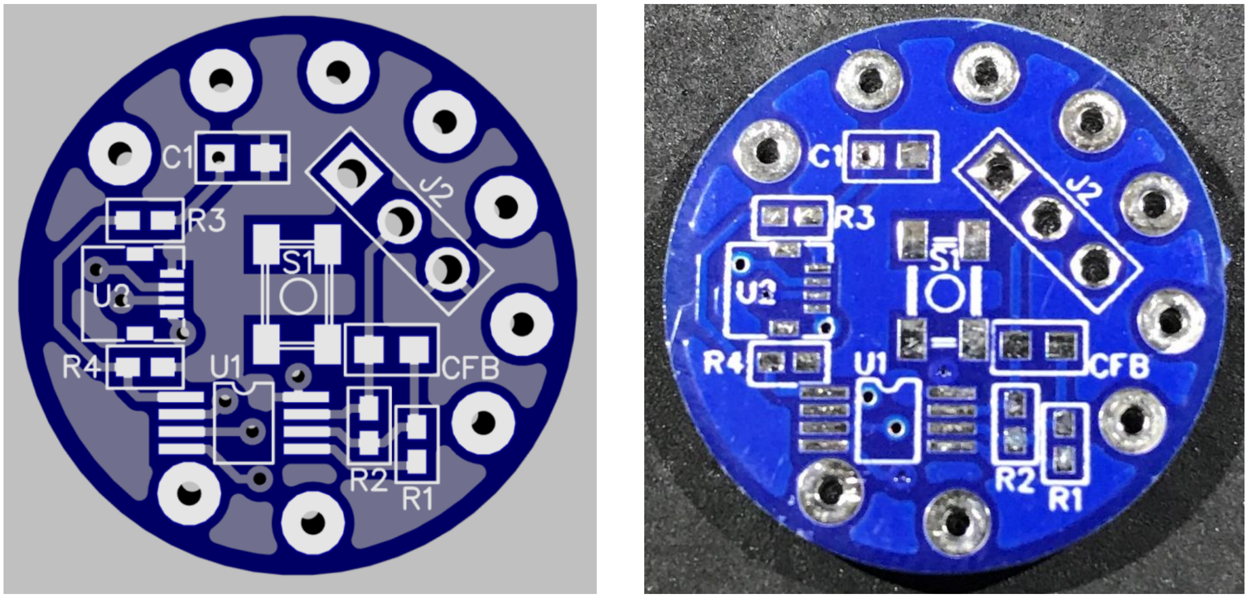


Figure 3.9 *Force Sensor/Amplifier Board Version 1 (Right) and 3D Model (Left)*

The PCB was designed with a round shape to avoid any sharp edges that may irritate the patient, since four of these PCBs are going to physically contact the leg. The board has a physical diameter of 21mm and a thickness of 1.6mm. The sensor (component *S1* in the PCB) used for this revision of the board is the *HSFPAR303A*, discussed in depth in the previous section. The connector which is required for the *303A* is also present on the board (component *U2*, the “2” being blocked by a via). Resistors *R1* and *R2* are the gain-setting resistors, while *R3* and *R4* set the reference voltage for the amplifier. The amplifier can also be seen as component *U1*, and *J2* is the 3-pin header connection for power, ground, and the amplifier’s output voltage.

3.5 FORCE SENSOR HOUSING MECHANISM

The force sensor/amplifier board discussed in the previous section is paired, using low-profile brass pogo pins, with a second PCB (containing no electrical components) of identical dimensions to form a spring-loaded housing unit for the force sensor. The motivation and philosophy behind building such a unit lies in the need to protect the sensor from damage and to assure that the sensor is sensing the correct applied force. The housing unit provides the force sensor with protection from damaging forces applied at any angle greater than 5° by limiting the forces applied to the sensor to only be applied by the turret pin in the mechanism of Fig. 3.10 below (5). The mechanics of the unit also guarantee that the sensor reads no pressure before a compression wrap is applied to the leg. When no pressure is being applied, the spring force of the pogo pins is plenty strong enough to lift the top PCB (4) completely so that the turret pin is not contacting the sensor. From the figure below, the force sensor pictured is the *HSFPAR303A* (3) and the force sensor PCB (1) is pictured with no other components on the board for the best visibility of the mechanism itself. The spring-loaded pogo pins (2) are evenly spaced 90° around the circular PCB.

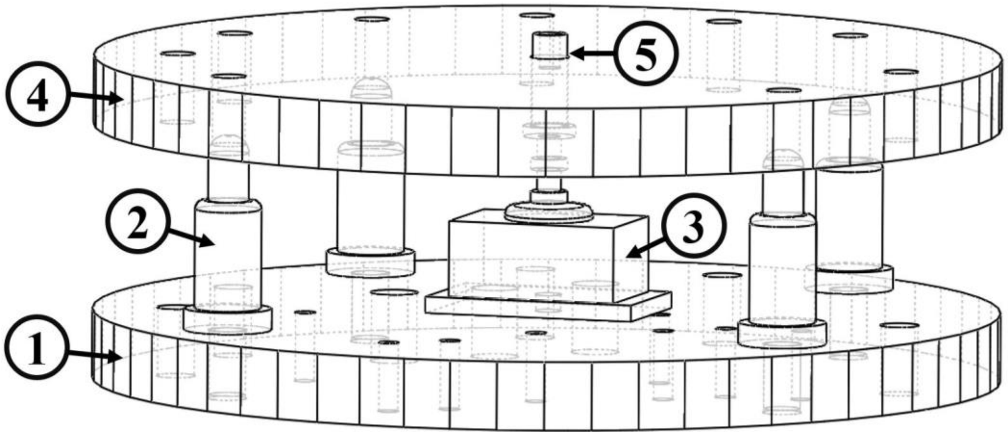


Figure 3.10 *Force Sensor Housing Mechanism 3D Model, Hidden Lines Showing*

3.6 CHAPTER 3 SUMMARY

Chapter 3 breaks down the force sensing using schematic diagrams and equations to show how applied force is converted into an analog output voltage that can be processed and converted into a pressure reading.

- **Section 3.1** introduces the Wheatstone bridge circuit and the piezoresistive elements of the force sensor. Equations are used to prove that the sensor output voltage is 0V when no force is applied and increases linearly as force is applied to the sensor.
- **Section 3.2** gives the motivation for the use of an instrumentation amplifier at the output of the force sensor.
- **Section 3.3** compares characteristics and parameters of two candidates for the force sensor used in the device, the first being the HSFPAR003A [8], and the second being the HSFPAR303A [9].
- **Section 3.4** provides an overview and analysis of the schematic and PCB design for the board containing the force sensor and amplifier circuitry.
- **Section 3.5** discusses the need for a housing mechanism around the force sensor and provides a 3D model view of the mechanism designed to both protect the force sensor from damage and to allow it to accurately measure applied force.

CHAPTER 4: DATA PROCESSING

4.1 THE MICROCONTROLLER (PIC18LF26K22)

The main control board for the system is comprised of three major components: the microcontroller, the Bluetooth module, and the buck-boost regulator. The microcontroller is discussed in depth in this section, while the Bluetooth module is discussed in depth in the next section. The buck-boost regulator and supporting components are discussed in detail in Ch. 6 on power consumption and battery life. The role of the microcontroller in the system can be broken down into three sequential tasks. The first task is to receive the data from each of the four force sensors, discussed in detail in Ch. 3. The second task is to process the data from the force sensors. The data processing includes calculation of the force applied given both the sensor output voltage and the sensitivity of the sensors (3.7mV/V/N), and the calculation of the pressure from the calculated force and predetermined area. The final task is to retransmit the processed data serially to the Bluetooth module, so that it can then be transmitted wirelessly to the smartphone application, discussed in detail in Ch. 5.

4.1.1 THE ADC (ANALOG-TO-DIGITAL CONVERTER)

The microcontroller (PIC18LF26K22) used in the system has a built-in 10-bit Analog-to-Digital Converter (ADC) with 19 input channels [12]. The system uses only 5 of the input channels: 4 for the force sensor outputs, and 1 for the battery voltage. These extra ADC input channels go unused and therefore a different, lower pin-count microcontroller could replace the PIC18LF26K22 in the system to decrease the size of the main control PCB. This section will focus on the operation of the on-chip ADC of the microcontroller, and the conversion of the analog output voltages from the sensors into meaningful data for the user to view on the smartphone application. Considerations include the ADC's resolution, quantization error when digitizing the sensor/amplifier output voltages, and the speed of the ADC.

The resolution of an ADC is typically described in terms of the LSB voltage or Least Significant Bit voltage of that ADC. Simply put, the LSB voltage of an ADC is the minimum magnitude of voltage change on the digital output. While the analog input signal can take on any value (a continuous range of values), the ADC *digitizes* the analog signal, and the digitized signal can only take on a finite number of discrete values between the positive and negative reference voltages of the ADC (usually VDD and ground). The LSB voltage is simply the magnitude of voltage between these discrete values in the digitized signal. Figure 4.1 below shows what the digitization of a sinewave might look like. Note that in the figure, the digital representation of the analog waveform follows the behavior of the signal but is not equal to the signal at every point. This error between the original analog signal and the digital representation of the signal is called the *quantization error.* The quantization error can be minimized by using a high-resolution ADC, or an ADC with a very small LSB voltage.

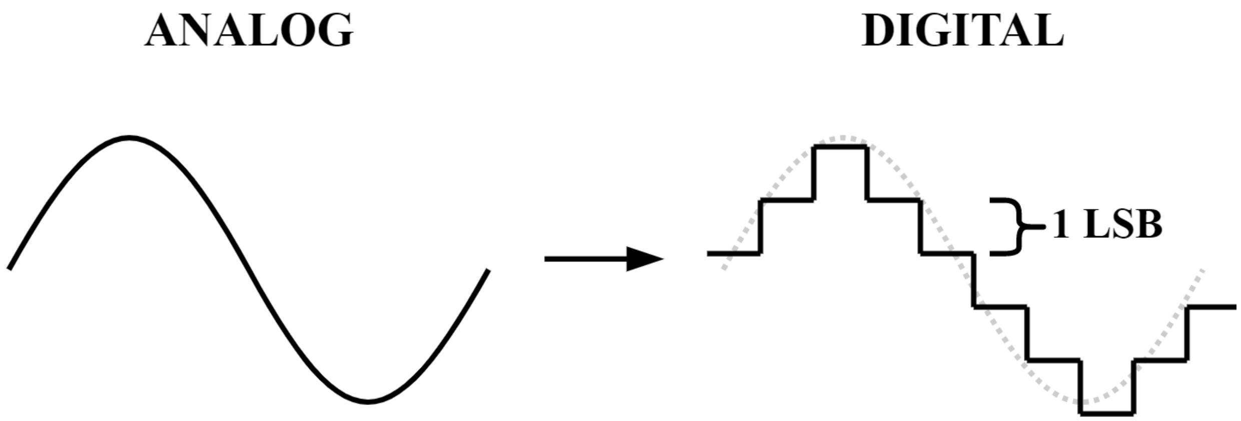


Figure 4.1 *Visual Representation of Analog-to-Digital Conversion and the LSB Voltage*

The LSB voltage of an ADC is a function of the number of bits of the ADC, and the positive and negative ADC reference voltages. Typically, the positive reference voltage is VDD, and the negative reference voltage is ground. If the positive reference is VDD and the negative reference is ground or 0V, then the LSB voltage is calculated by

$$V_{LSB} = \frac{V_{REF+} - V_{REF-}}{2^N} = \frac{VDD}{2^N} \tag{4.1}$$

where N is the number of bits of the ADC. The PIC18LF26K22 on-chip ADC is a 10-bit ADC, and the power supply voltage of the chip is nominally 3.3V. Using these values, the LSB voltage of the ADC can be calculated using the equation above.

$$V_{LSB} = \frac{VDD}{2^N} = \frac{3.3V}{2^{10}} = 3.22mV \tag{4.2}$$

What this means conceptually is that there are 2^{10} different discrete levels that the digitized signal can take on, evenly spaced by 3.22mV, or the LSB voltage. This also means that the sensor/amplifier board's output signal must vary by at least 3.22mV in order for the ADC output to change at all. Using this LSB voltage, along with the gain of the amplifier and the sensitivity of the force sensors, the minimum sensible change in pressure for the system can be calculated.

The theoretical minimum sensible change in pressure is calculated using a gain of 20 V/V and an area of 346.36 mm^2, the area of the PCB of Fig. 3.9. The gain of the amplifier and the supply voltage can be factored into the sensitivity of the force sensors by

$$Sensitivity * VDD * G = (3.7mV/V/N) * 3.3V * 20 = 244.2mV/N \tag{4.3}$$

The unit of Volts from the supply voltage cancels with the unit of Volts from the force sensor's sensitivity to yield a simpler sensitivity value of 244.2mV/N for the sensor/amplifier board. This means that for every 1N of force applied to the sensor apparatus, the ADC will see a 244.2mV change in the output voltage of the sensor/amplifier board. From eq. (4.2), the on-chip ADC can sense changes in voltage as small as 3.22mV. Thus, dividing the simplified sensitivity value of eq. (4.3) by the LSB voltage of eq. (4.2), the **smallest sensible change in applied force** is obtained.

$$\frac{(244.2mV/N)}{(3.22mV/LSB)} \cong 75\frac{LSB}{N} = 0.0133\frac{N}{LSB} \tag{4.4}$$

From the equation, the smallest change in force that can be sensed by the system is theoretically 0.0133 N. This force value is converted to a pressure value when it is divided by the area of the force sensor/amplifier PCB, or

$$\frac{0.0133N}{346.36mm^2} = 3.84 \cdot 10^{-5}\frac{N}{mm^2} \tag{4.5}$$

Using the conversion factor from mmHg to N/mm^2, the minimum sensible pressure change in mmHg is calculated: 1 N/mm^2 = 7500 mmHg. Therefore,

$$\left(3.84 \cdot 10^{-5}\frac{N}{mm^2}\right) * \left(7500\frac{mmHg}{N/mm^2}\right) = 0.285\ mmHg \tag{4.6}$$

To be effective, the device needs to sense pressure changes around 1mmHg accurately. From eq. (4.6), the smallest sensible change in pressure is slightly less than 1/3 mmHg.

4.1.2 DATA PROCESSING

To obtain a pressure reading from an applied force requires a sequence of data processing steps. The sequence of steps is displayed in Fig. 4.2 below, where each box represents a different state for the data, and each arrow represents the processing that takes place between two states.

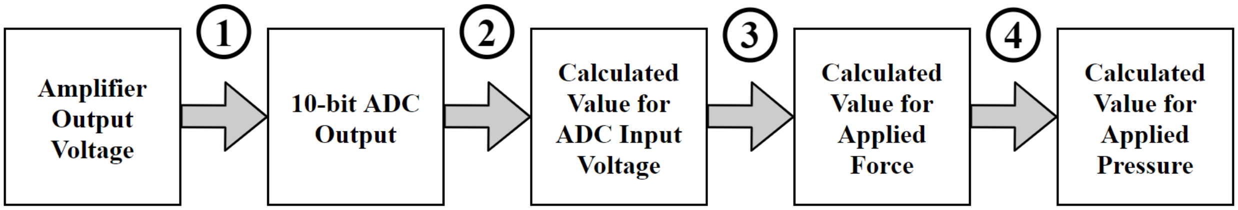

Amplifier Output Voltage
①
10-bit ADC Output
②
Calculated Value for ADC Input Voltage
③
Calculated Value for Applied Force
④
Calculated Value for Applied Pressure

Figure 4.2 *Data Processing Sequence*

First, the force sensor's output voltage is amplified using the instrumentation amplifier, discussed in detail back in Ch. 3. Once amplified, the signal is sent to one of the input channels of the MCU's on-chip 10-bit ADC. **(1, Fig. 4.2)** The ADC converts the analog voltage into a digital value between 0 and 1023, where 0 corresponds to 0V and 1023 corresponds to a voltage of 1 LSB less than the supply voltage, or VDD – 1LSB. **(2, Fig. 4.2)** The corresponding voltage $V_{ADC,OUT}$ for *any* digital output value can be obtained by multiplying the digital output value by the LSB voltage, or

$$V_{ADC,OUT} = ADC_{OUT} * 1\ LSB \tag{4.7}$$

where ADC_{OUT} is a unitless value between 0 and 1023. From eq. (4.1), the LSB voltage is simply VDD divided by 2^N, where N is the number of bits of the ADC. Since the on-chip ADC has 10 bits, the LSB is calculated by dividing VDD by 2^{10}, or 1024.

Once the voltage corresponding to the ADC output is obtained, **(3, Fig. 4.2)** the applied force can be calculated using the sensitivity value of the force sensor, calculated using eq. (4.3). The sensitivity value of 244.2 mV/N means that if a 244.2mV change in output voltage is sensed, then 1N of force is applied to the sensor. Thus, taking $V_{ADC,OUT}$ and dividing by the sensitivity value would theoretically yield the applied force in Newtons. However, the MAX4208 instrumentation amplifier used in this system has an output voltage offset of VDD/2 to allow for the amplification of both positive and negative voltages without the need for a negative voltage reference. This results in

$$V_{ADC,OUT} = (VDD/2) + \Delta V_{Force} \tag{4.8}$$

where ΔV_{Force} is the change in amplifier output voltage resulting from the force applied. Rearranging eq. (4.8),

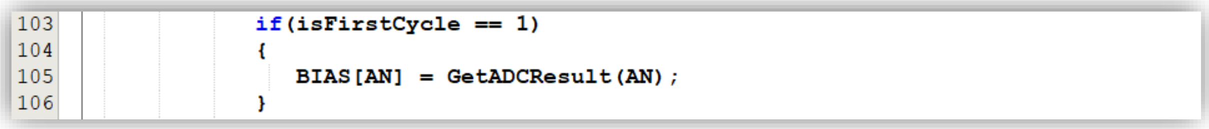
$$\Delta V_{Force} = V_{ADC,OUT} - (VDD/2) \tag{4.9}$$

where now multiplying ΔV_{Force} by the sensitivity value (244.2mV/N) yields the force applied to the force sensor in Newtons.

Ideally, when no force is applied to the sensor at all, the ADC output should be VDD/2 (corresponding to a digital value of 512), assuming the initial force sensor output in eq. (4.10), $\Delta V_{Force,init}$ is 0V. However, due to mismatch in the piezo-resistive elements of the force sensor, the initial force sensor output voltage is nonzero, typically slightly negative. This poses a pair of issues. The first issue is that the initial force sensor output, $\Delta V_{Force,init}$, is different for every sensor depending on the mismatch of the piezo-resistive elements. Datasheets for the sensors give a range, but do not give an exact value because the exact value is unknown. The second issue is that the initial force sensor output gets amplified by the instrumentation amplifier, resulting in an even larger offset. Thus, this *bias voltage* cannot be ignored, and needs to be factored into calculation. The ADC output voltage value including the initial sensor voltage is then calculated by

$$V_{ADC,OUT} = (VDD/2) + (\Delta V_{Force} \pm \Delta V_{Force,init}) \tag{4.10}$$

Clearly, in order to obtain accurate readings, the value of $\Delta V_{Force,init}$ needs to be known. A small conditional statement is used in software to assist in the elimination of this problem. Figure 4.3 below shows an *if* statement used in the MCU program which checks which number of cycle the program's *while* loop is currently on.



```
if(isFirstCycle == 1)
{
    BIAS[AN] = GetADCResult(AN);
}
```

Figure 4.3 *Conditional Statement to Take Care of Bias Voltage*

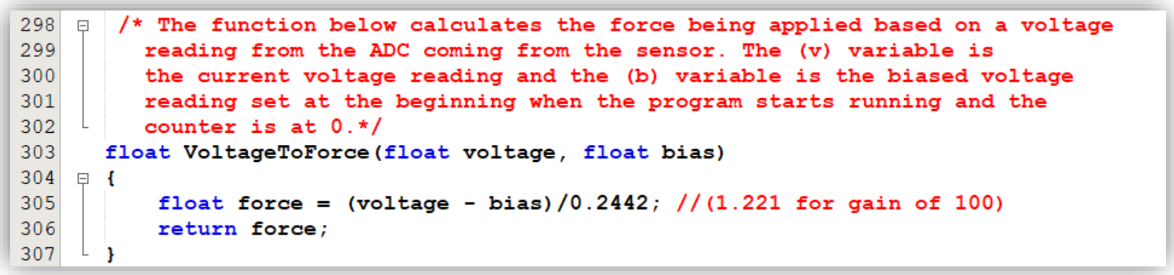
If and only if the program is on the first cycle through the while loop (right when the device first powers up), the ADC output voltage at that moment is saved and used as the *bias* for the sensor connected to that ADC input channel for the duration of time until the MCU is reset or powered down. This *bias voltage* is given by

$$V_{bias} = (VDD/2) \pm \Delta V_{Force,init} \tag{4.11}$$

Note how the *bias* is simply two of the three terms on the right-hand side of eq. (4.10). Rearranging eq. (4.10) and using the *bias*, we can obtain a more accurate form of eq. (4.9), or

$$\Delta V_{Force} = V_{ADC,OUT} - V_{bias} \tag{4.12}$$

Eq. (4.12) is the equation that is used in software to calculate the force applied for a given ADC output voltage, $V_{ADC,OUT}$. The software implementation of eq. (4.12) is shown in the figure below, where the 0.2442 has a unit of Volts per Newton (V/N) and corresponds to the force sensor/amplifier sensitivity value of eq. (4.3).



```
 /* The function below calculates the force being applied based on a voltage
   reading from the ADC coming from the sensor. The (v) variable is
   the current voltage reading and the (b) variable is the biased voltage
   reading set at the beginning when the program starts running and the
   counter is at 0.*/
float VoltageToForce(float voltage, float bias)
{
    float force = (voltage - bias)/0.2442; //(1.221 for gain of 100)
    return force;
}
```

Figure 4.4 *Function to Calculate Force Applied from Sensed Voltage*

In the function **VoltageToForce** above, the *bias* variable value is obtained via the software method described previously in Fig. 4.3 and can be solved for using eq. (4.11). The *voltage* variable value is obtained using eq. (4.7) in software. The function takes in the float-valued voltage and the float-valued bias and outputs the applied force corresponding to the calculated ADC output voltage.

Once the force is obtained, the corresponding applied pressure value is obtained by dividing the force by an area **(4, Fig. 4.2)** specifically the area over which the force is being applied.

In the case of the sensor apparatus, the area is theoretically the area of the circular PCB shown in Fig. 3.9. This PCB has a diameter of 21 mm. To find the area, the area of a circle formula can be applied using half the diameter (10.5 mm) as the radius. Since medical pressure readings use mmHg (millimeters of Mercury) as their unit, whatever unit results from the division of force by area will need to be converted into units of mmHg. The area of the PCB, A_{PCB}, is calculated first by

$$A_{PCB} = \pi \cdot r^2 = \pi \cdot (10.5\, mm)^2 = 346.36\, mm^2 \tag{4.13}$$

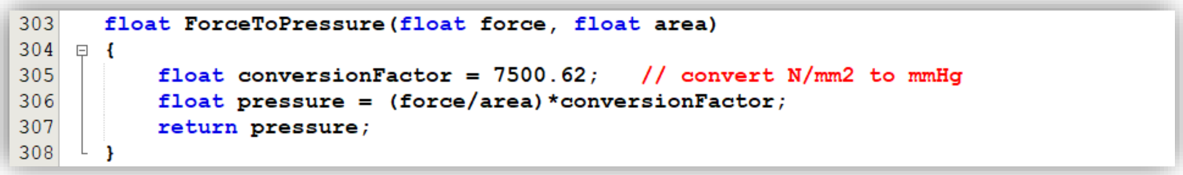
With the calculated area of the PCB, the pressure applied to the patient's leg for a given applied force can be calculated by dividing the area of the PCB by an applied force, or

$$P_{APP} = F_{APP}(N)/A_{PCB}(mm^2) \tag{4.14}$$

where P_{APP} is the pressure applied to the patient's leg and F_{APP} is the force applied to the sensing mechanism. This quotient yields units of N/mm^2, which can easily be converted to mmHg through a conversion factor.

$$1\ N/mm^2 = 7500.62\, mmHg \tag{4.15}$$

Equations (4.14) and (4.15) describe the operation of the code block or function presented in Fig. 4.5 below. The function takes in a float-valued *force* from the **VoltageToForce** function described previously and a predefined *area* which is a global constant float value in the program. The function then divides *force* by *area* and multiplies the quotient by the conversion factor. The conversion factor is that of eq. (4.15), and the N/mm^2 from both the conversion factor and the quotient cancel out to yield a *pressure* value with units of mmHg, which the function returns.



```
float ForceToPressure(float force, float area)
{
    float conversionFactor = 7500.62;   // convert N/mm2 to mmHg
    float pressure = (force/area)*conversionFactor;
    return pressure;
}
```

Figure 4.5 *Function to Calculate Pressure from Applied Force*

The **ForceToPressure** function is the final step for data processing before it is ready to transmit to the Bluetooth module. The *pressure* value that is returned by the function will be displayed on the main page for the smartphone application, discussed in detail in Ch. 5.

4.1.3 UART COMMUNICATION

Voltage, *force*, and *pressure* values obtained using the methods described in section 4.1.2 are all transmitted to the Bluetooth module via the MCU's built-in UART, or Universal Asynchronous Receiver-Transmitter. UART is a means of wired serial communication between two devices. In the case of this system, the UART enables communication between the microcontroller and the HM-10 Bluetooth Low Energy module [6]. In UART communication, each device has a receiver pin and a transmitter pin. The transmitter (Tx) of one device is connected to the receiver (Rx) of the other, and vice versa. Though the UART can be configured for full duplex (data can be both transmitted and received over the same channel simultaneously), for this application, data never needs to be both transmitted and received simultaneously. Either the microcontroller is sending data to the Bluetooth module, or the Bluetooth module is sending commands to the microcontroller via user inputs, but never both simultaneously.

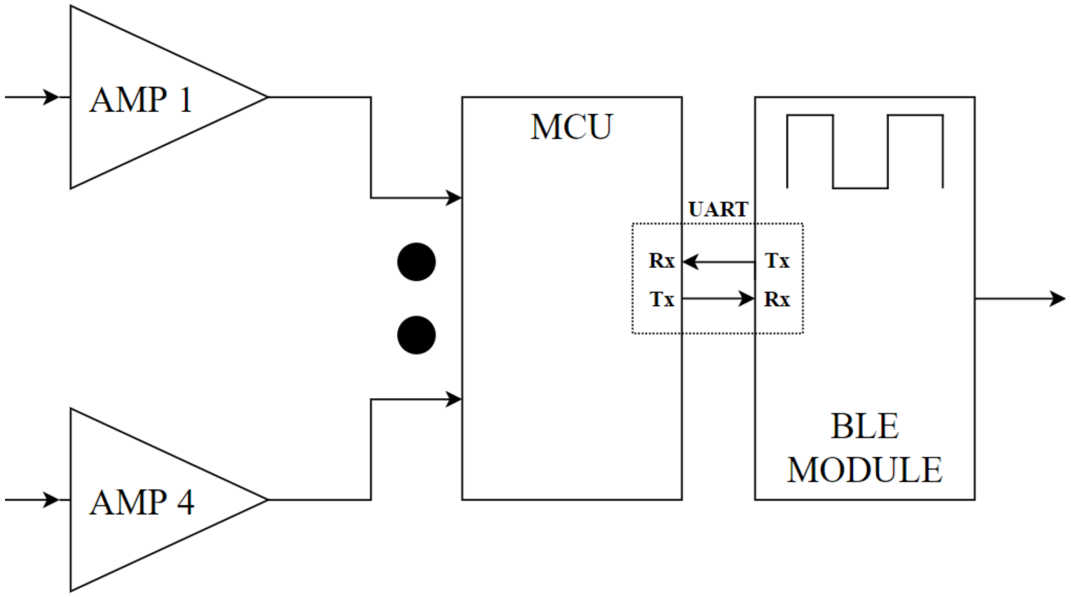
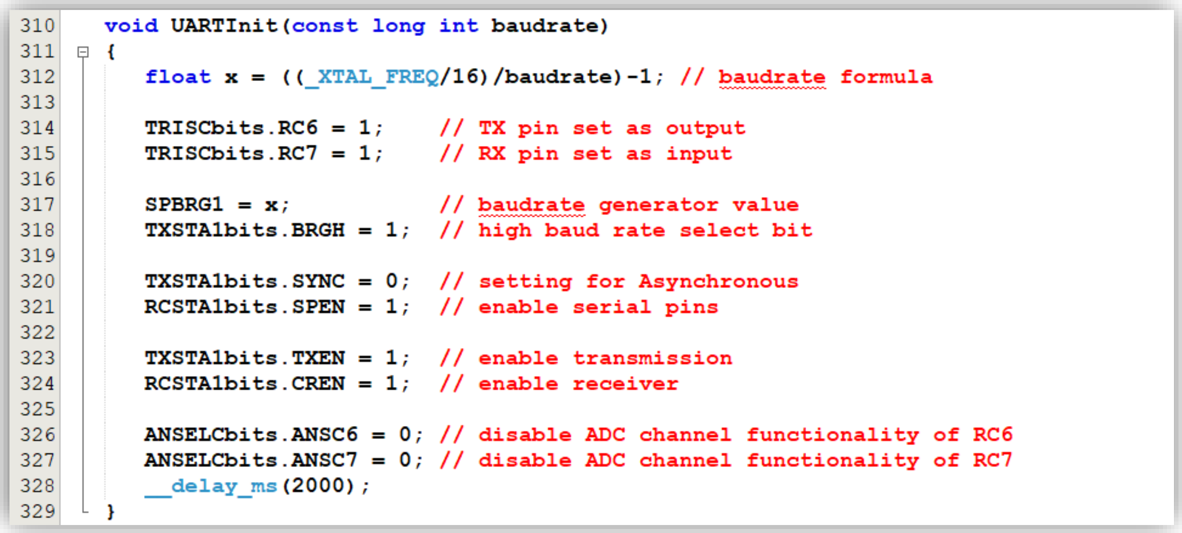


Figure 4.6 *Illustration of UART Connection Between MCU and BLE Module*

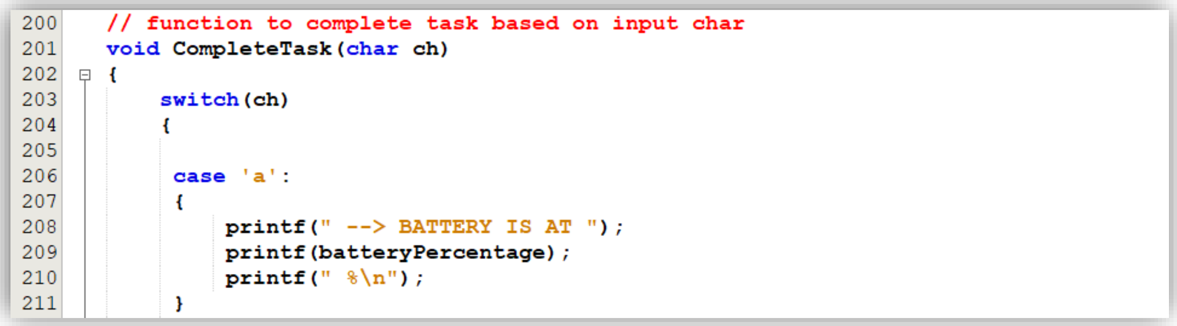
Figure 4.6 above shows a zoomed-in, modified version of the system overview of Fig. 2.1. This version is modified to show how the receiver pin of the MCU is connected to the transmitter pin of the BLE module, and vice versa. The UART is a very convenient way to transfer data from one device to another serially without the burden of following any strict protocol. The microcontroller simply needs to be configured such that the UART is enabled to both transmit and receive data, but as mentioned previously, it will not need to both transmit and receive simultaneously. The microcontroller has several registers used for configuring different peripherals on the chip, including the ADC, the DAC, interrupts, and other peripherals. The same goes for the UART. The configuration registers contain different bits for the user to manipulate in order to configure the UART to their specific needs. For example, in Fig. 4.7 below, bits are set/cleared to select synchronous/asynchronous mode, enable the transmitter and receiver, disable the ADC channel functionalities of the pins containing the receiver and transmitter, select whether the Baud rate is high or low, and set the data direction of the transmitter and receiver pins. Note that this peripheral can be set into synchronous mode, though it was previously referred to as a UART (Universal **Asynchronous** Receiver-Transmitter). This MCU actually has what is known as a EUSART [12], which is an Enhanced Universal Synchronous/Asynchronous Receiver-Transmitter. However, since this application only uses the asynchronous functionality and does not use the enhanced features, for simplicity, this peripheral is referred to as UART, and will be for the remainder of the paper.



```
void UARTInit(const long int baudrate)
{
   float x = ((_XTAL_FREQ/16)/baudrate)-1; // baudrate formula

   TRISCbits.RC6 = 1;    // TX pin set as output
   TRISCbits.RC7 = 1;    // RX pin set as input

   SPBRG1 = x;           // baudrate generator value
   TXSTA1bits.BRGH = 1;  // high baud rate select bit

   TXSTA1bits.SYNC = 0;  // setting for Asynchronous
   RCSTA1bits.SPEN = 1;  // enable serial pins

   TXSTA1bits.TXEN = 1;  // enable transmission
   RCSTA1bits.CREN = 1;  // enable receiver

   ANSELCbits.ANSC6 = 0; // disable ADC channel functionality of RC6
   ANSELCbits.ANSC7 = 0; // disable ADC channel functionality of RC7
   __delay_ms(2000);
}
```

Figure 4.7 *UART Initialization Function*

The MCU sends data to the Bluetooth module via UART so that it can then be transmitted wirelessly to the user's smartphone. The MCU also **receives** data from the Bluetooth module in the form of characters, one byte at a time. The ASCII character table provides each character (all uppercase and lowercase letters, various symbols, digits 0-9, and more) with a number between 0 and 255. This means that each character can be represented by its 8-bit binary code. The smartphone application (discussed in detail in Ch. 5) provides the user with an interface containing convenient buttons for various tasks. For example, if the user wants to check the battery life of the device at any time, they can simply click the button labeled “Battery Life.” What then happens is the smartphone sends a byte of data representing some character (say the letter ‘a’) to the Bluetooth module, which then sends that same byte via UART to the microcontroller. The microcontroller is programmed with a function called **CompleteTask** which then takes in the received character (ch) and completes the task assigned to that character, in this case printing the current battery percentage. The first several lines of this function are shown below in Fig. 4.8.



```
// function to complete task based on input char
void CompleteTask(char ch)
{
    switch(ch)
    {

     case 'a':
     {
         printf(" --> BATTERY IS AT ");
         printf(batteryPercentage);
         printf(" %\n");
     }
```

Figure 4.8 *Complete Task Function, Completes Task According to Received Character*

4.1.4 HARDWARE IMPLEMENTATION

As mentioned at the beginning of Ch. 4, the main PCB for the system contains the microcontroller, the Bluetooth module, and the regulator circuitry. It also contains a number of other peripherals for those circuits, including a Bluetooth connectivity status LED, 2 x 100-mil spaced male headers for connecting the battery, 6 x 100 mil-spaced programming pins for the PIC programmer to connect to during programming, 12 x 100 mil-spaced pins for the wires to be routed out to power and receive data from the sensor boards, and a number of passives, including decoupling capacitors, the inductor for the buck-boost regulator, and resistors for current limiting, battery voltage monitoring, etc.

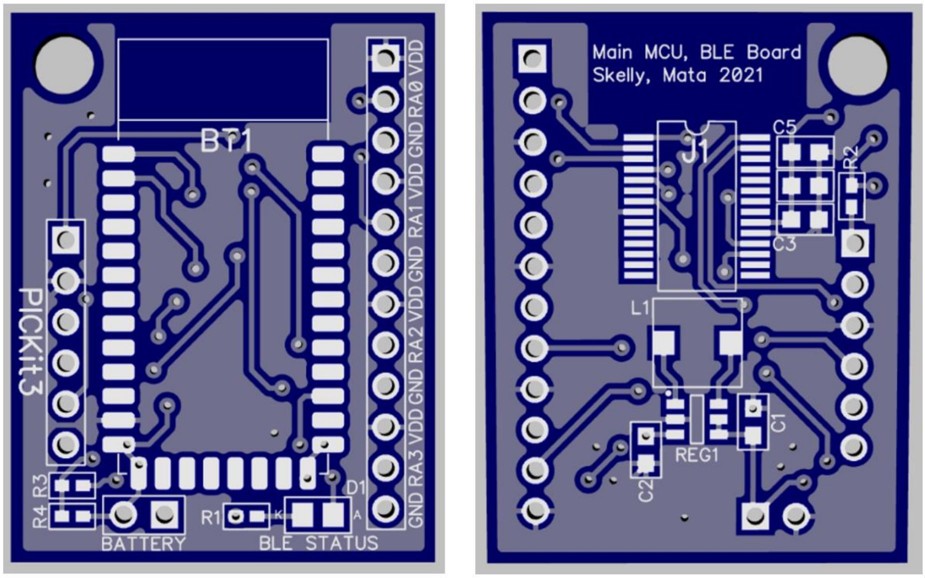


Figure 4.9 *Main MCU/BLE Printed Circuit Board 3D Model*

The left side of the figure above is the top side of the PCB, containing the Bluetooth module, the battery and programmer header pins, the Bluetooth connectivity status LED, and the header pins for routing wires out to the sensor/amplifier boards. The top portion of the board does not have any copper pour where the Bluetooth module antenna will be to reduce EMI. The right side of the figure is the bottom side of the PCB, which contains the buck-boost regulator circuitry, and the microcontroller and its decoupling capacitors. Copper is poured everywhere that traces and components are not located: the top layer pour is connected to VDD, and the bottom layer pour is connected to ground.

4.2 THE BLUETOOTH MODULE

The HM-10 Bluetooth module is a 34-pin surface-mount module containing an on-board BLE microcontroller (CC2541 2.4-GHz Bluetooth Low Energy SOC). The MCU is equipped with both wired serial UART and wireless Bluetooth communication capabilities. The module can both receive serial data via UART and then transmit it wirelessly to a BLE receiver or receive data wirelessly via BLE protocol and then transmit it serially via UART. From Fig. 4.10 below, the pinout shows that many the pins are NC (or not connected). Additionally, there are 12 GPIO pins, only one of which is needed for the system's Bluetooth connectivity status LED. The only pins that need to be used by the system are the VCC, GND, UART_TX, UART_RX, and PIO1 pins. Since this is the case, a smaller Bluetooth module with lower pin count is desirable for the system.

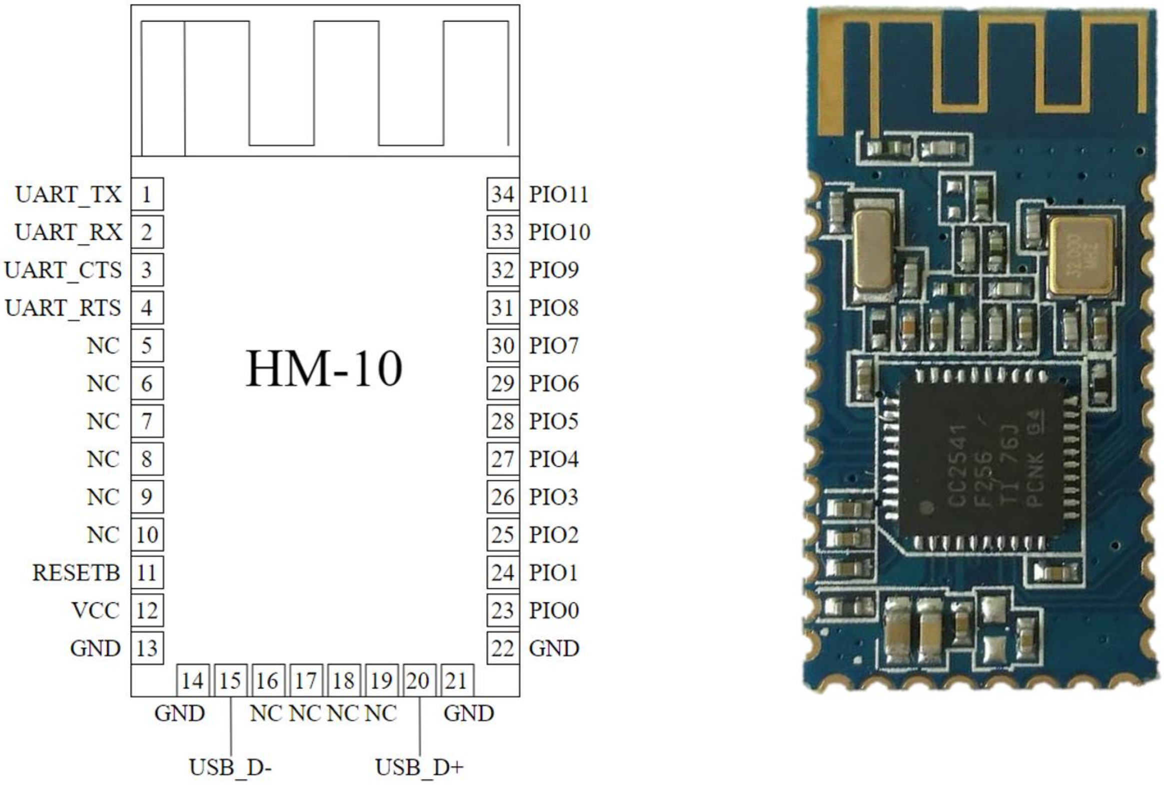


Figure 4.10 HM-10 Bluetooth Low Energy Module and Pinout [6]

An alternative Bluetooth module with the same on-board MCU is the HM-11 module. The HM-11 module has just 16 pins but operates identically to the HM-10 in all aspects necessary for the design of this system. When using the HM-11 in the system, the pins used are the same as those used with the HM-10 (VCC, GND, UART_TX, UART_RX, and PIO1). The HM-11 has the same width as the HM-10, but the vertically height is a little more than half that of the HM-10, making it desirable for managing the size of the main MCU/BLE board. Figure 4.11 below shows the HM-11 and its pinout.

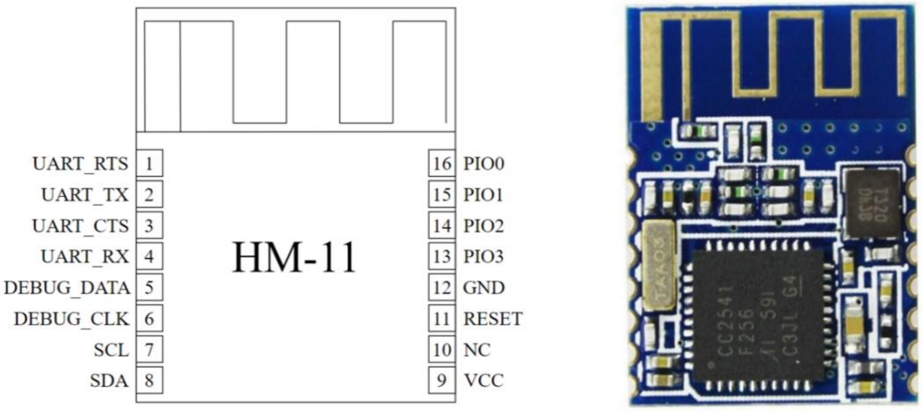
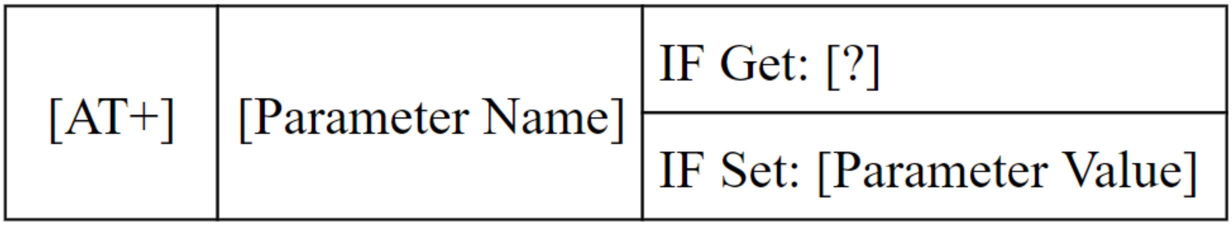


Figure 4.11 *HM-11 Bluetooth Low Energy Module and Pinout [6]*

4.2.1 CONFIGURATION WITH AT-COMMANDS

Configuration for both the HM-10 and the HM-11 modules is performed using AT commands. The configurable settings on the modules follow a “Get-Set” format, where to “Get” data from the module is to read a value out, or check the status of a particular setting, and to “Set” data is to write a value to the module or reconfigure a particular setting. Some parameters are both “Gettable” and “Settable”, and others are only “Gettable” (read-only) or only “Settable” (write-only). The AT commands follow a particular syntax, where each command is made up of three distinct parts, not separated by any spaces. The three-character sequence “AT+” begins every AT command. The “AT+” is followed by the name of the chosen parameter. The final part of the command is what tells the module whether the user wishes to “Get” or “Set” the chosen parameter. If the “Get” is to be used, the command terminates with a question mark. If the “Set” is to be used, the command terminates with a parameter value. The syntax is better described by the diagram in Fig. 4.12 below, showing the sequence of characters used to form both “Get” and “Set” commands. Note that “Get” and “Set” commands are identical up until the way they terminate (both start with “AT+” followed by a parameter name).



[AT+]	[Parameter Name]	IF Get: [?]
		IF Set: [Parameter Value]

Figure 4.12 *AT Commands Syntax for Configuring Bluetooth Module*

The device datasheet for the chosen Bluetooth module contains tables which provide the user with a comprehensive list of all the available settings and their access type (“Get,” “Set,” or both). These tables also assign a single digit to each parameter value to simplify the AT commands that are sent to the device and limit the possibility of user input errors. If the module receives a correctly transmitted AT command, it will acknowledge the command and respond with the acknowledgement (“OK+”), followed by the command type (“Get” or “Set”), and terminating with the corresponding number of the parameter that was read or written. The format for the acknowledgement message is shown in Fig. 4.13. The table in the datasheet also tells the user the default value of each parameter. For example, the Baud rate parameter has a list starting at the default value (9600, parameter number 0), ranging up to 230400 (parameter number 8) and down to 1200 (parameter number 6). Other common Baud rates are also included in the list (such as 19200, parameter number 1 and 115200, parameter number 4).

<table>
<tr><td rowspan="2">[OK+]</td><td>IF Get: [Get:]</td><td rowspan="2">[Parameter Value]</td></tr>
<tr><td>IF Set: [Set:]</td></tr>
</table>

Figure 4.13 *AT Commands Acknowledgement Message Format*

The user can “Get” the current Baud rate setting for the module by sending the string “AT+BAUD?” to the module via UART. If, for example, the module is set to the default Baud rate of 9600, the acknowledgement message would read “OK+Get:0” informing the user that parameter 0 (9600 Baud) is the current Baud rate setting. If the user wishes to change this setting to, say 19200, the command “AT+BAUD1” should be sent to the module. The corresponding acknowledgment message would read “OK+Set:1” notifying the user that the Baud rate has been set to the value associated with parameter value 1, or 19200 (mentioned in the previous paragraph). Other parameters that are both “Gettable” and “Settable” include the module work mode, the device name (as it would appear to a user in a list of available Bluetooth devices), the behavior of the status LED, a pin code for users trying to connect to the device, the usable range of the device, among others. An example of a parameter that is only “Gettable” or read-only is the device MAC address. An example of a parameter that is write-only is the RESET command, which restarts or refreshes the module. Each time AT commands are used to configure or change settings, the device should be reset to update the configuration changes. Until the device is reset, the changes will not go into effect. On the main MCU/BLE board, the MCU's UART transmitter is connected to the BLE module's receiver, and the MCU's receiver is connected to the BLE module's transmitter. Thus, the MCU can be used to send AT commands directly to the BLE module by simply transmitting a string via UART.

4.2.2 DATA TRANSMISSION AND RECEPTION

By default (configured with factory settings) the BLE module can transmit and receive data both serially through UART and wirelessly through Bluetooth communication. The smartphone also plays a large role in the wireless communication, as smartphones in general are equipped with Bluetooth capabilities for both transmitting and receiving data. The operation of the smartphone application with the Bluetooth module will be discussed further in Ch. 5. The antenna of the Bluetooth module is used for both transmitting and receiving data. Figure 4.14 below shows how data takes one of two paths: the first is from the MCU to the BLE module (serially) and then to the phone (wirelessly), and the second is from the phone to the BLE module and then to the MCU.

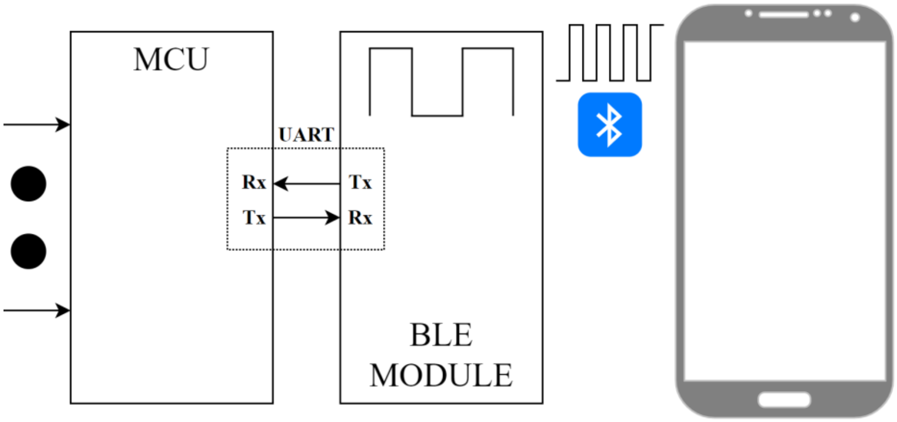


Figure 4.14 *Visual Diagram of Serial and Wireless Communication*

Bluetooth is a wireless communication protocol which uses the frequency band of the electromagnetic spectrum ranging from 2.4 GHz to 2.4835 GHz to communicate [13]. This broader frequency band is split up into 79 smaller frequency bands or "channels." Within each channel, a specific frequency is assigned for a "1" and a different frequency for a "0". The difference between the frequency assigned to a "0" and that which is assigned to a "1" is 740 kHz. This type of communication (where digital information is transmitted using discrete frequency changes) is also called frequency-shift keying (FSK). For Bluetooth 4.0 (the Bluetooth version of the HM-10 and HM-11 modules), the data rate is 1 Mbps (Megabit per second). This means that 1 million bits, or ones and zeros, are transmitted and received every second via Bluetooth [13]. However, the frequencies that are used to represent a "1" or a "0" are within the 2.4 GHz to 2.4835 GHz range.

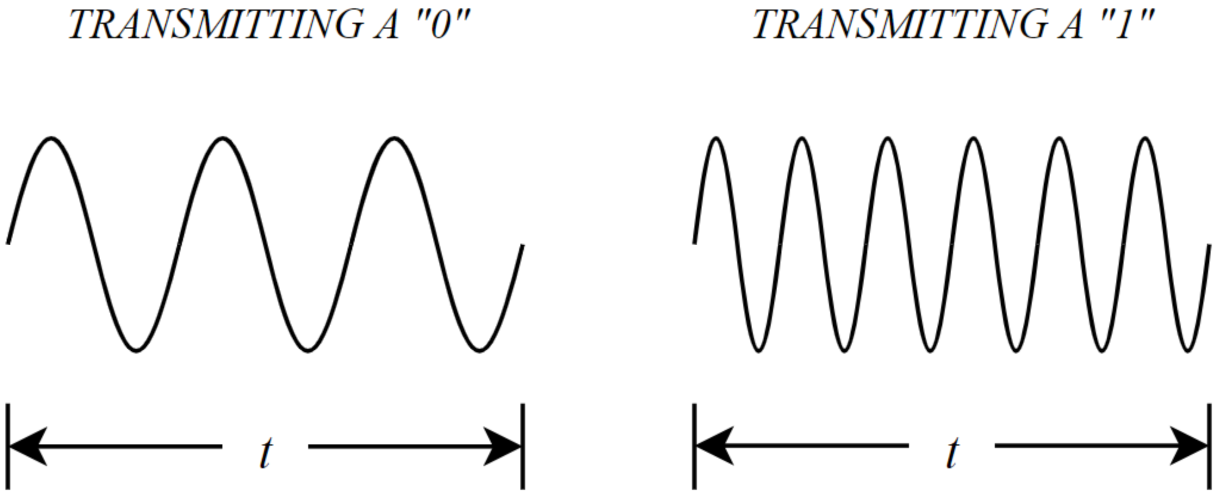


Figure 4.15 *Example of FSK Signals for Transmitting "1" or "0"*

Each channel spans a frequency range of 1 MHz. For example, on channel number 38, a "1" is represented by a 2.44387 GHz signal while a "0" is represented by a 2.44313 GHz signal. On the next channel over, channel 39, a "1" is represented by a 2.44287 GHz signal, and a "0" is represented by a 2.44231 GHz signal. The difference between the "1" in channel 38 and the "1" in channel 39 is 1 MHz. Likewise, the difference between the “0” in channels 38 and 39 is also 1 MHz. As mentioned previously, the “1” and the “0” in any given channel are separated by 740 kHz. This leaves a 130 kHz gap above the frequency of the “1” and below the frequency of the “0” as breathing room in any given channel. Summing the 740 kHz difference between the “1” and “0” and two 130 kHz gaps for breathing room yields a 1 MHz frequency band for any given channel.

The Bluetooth device transmitting the data and the Bluetooth device receiving the data perform what is called frequency-hopping spread spectrum (FHSS) while they communicate [13]. FHSS is a method of RF communication in which the transmitter and receiver devices each follow a predetermined “schedule” for different channels to communicate across during different time slots within the full frequency band of the communication protocol, which in the case of Bluetooth is 2.4 GHz to 2.4835 GHz. Bluetooth time slots for one packet of information to be transmitted and received are 625 microseconds long. This translates directly to 1600 frequency hops per second. FHSS is used in Bluetooth so that many devices can transmit and receive data wirelessly within the same frequency band error-free, and so a single frequency band cannot be tapped into, and data being transmitted cannot be received by any other device other than the device that was meant to receive it. Only the two Bluetooth devices that are paired are aware of the frequency-hopping schedule.

4.3 CHAPTER 4 SUMMARY

Chapter 4 gives a detailed breakdown of the data processing techniques used to convert the applied force into a pressure value displayed on the user’s smartphone. The chapter is broken up into two larger sections, the first being the microcontroller, and the second being the Bluetooth module.

- **Section 4.1** dives into the PIC18LF26K22 MCU’s peripherals that are used in the system design, including the ADC and the UART communication channel. Code snippets are included to show how the processing is implemented in software, and the end of the section discusses the PCB design and hardware implementation of the MCU’s peripherals.
- **Section 4.2** discusses the configuration and operation of the Bluetooth low energy modules that can be used in the system, namely the HM-10 and HM-11 modules. The two modules are also compared on the basis of size since the modules are otherwise identical in terms of operation within this particular system.

CHAPTER 5: THE SMARTPHONE APPLICATION

5.1 INTERFACE AND LAYOUT

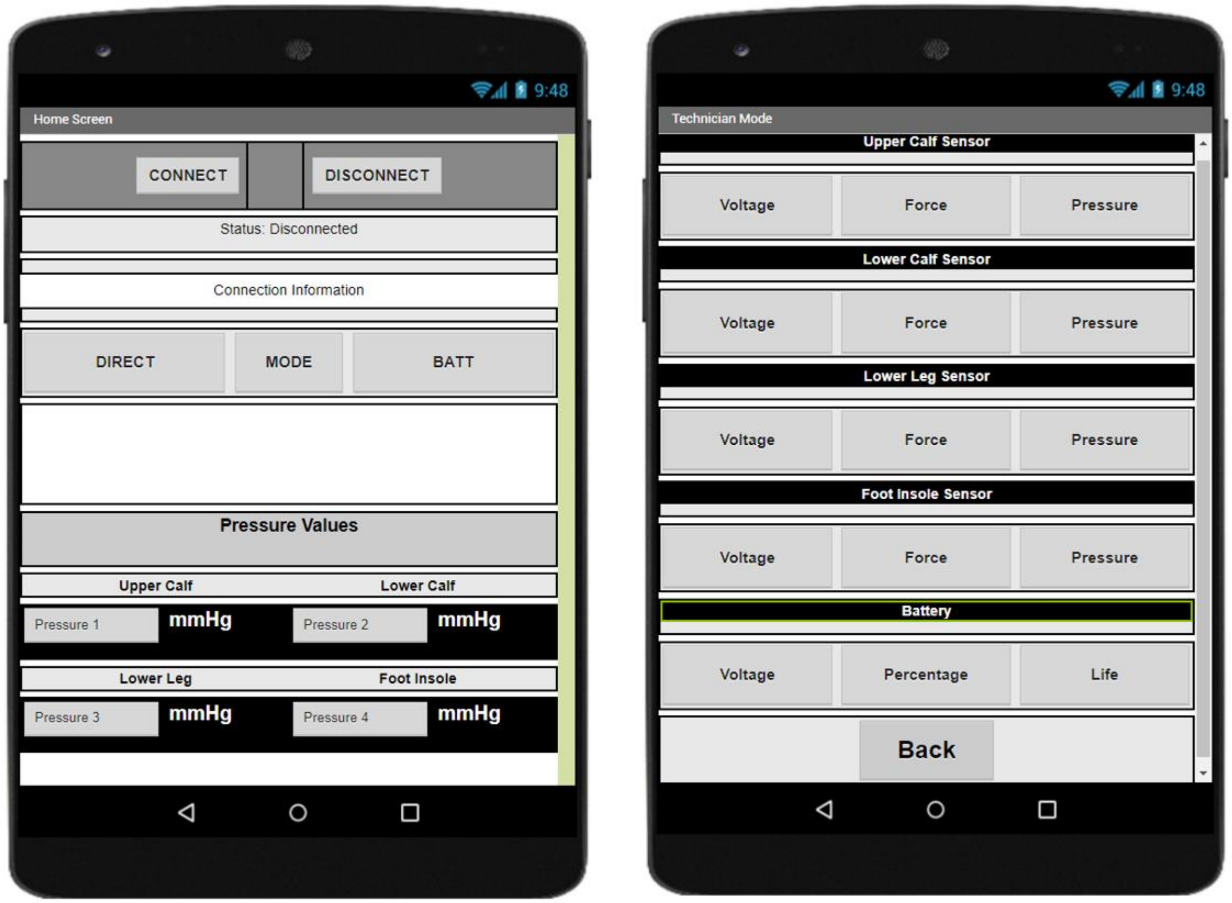
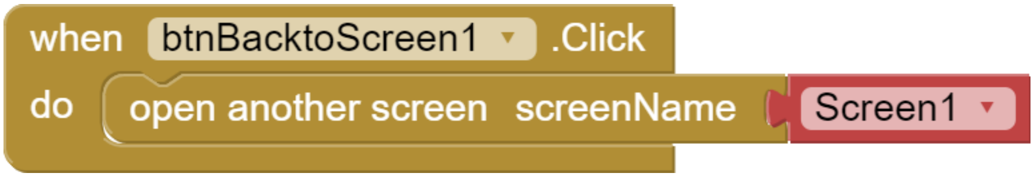


Figure 5.1 *Smartphone App Home Screen and Technician Mode Screen, Designer View*

The smartphone application is designed using the web-based *MIT App Inventor* and is compatible for Android devices version 4.3 and later. Android version 4.3 was released on July 24, 2013 [15] and featured Bluetooth Low Energy support for the first time in Android version history. Since all later versions of Android feature BLE support, the device is compatible with Androids of all versions 4.3 and later. Figure 5.1 shows the *Designer* view in the MIT App Inventor for both screens in the application. The first (left) screen is the home screen (Screen 1), which is the default screen when the application is first opened. The second (right) screen is the "Technician Mode" screen (Screen 2) which is only meant to be accessed by the nurse or technician who is working with the patient who owns the device. The *Designer* view allows the app designer to place buttons, labels, text boxes, switches, and several other user interface items into the current screen for the application layout. Logic is then added to each interface item using the *Blocks* view. For example, Fig. 5.2 below shows the logic blocks for the BACK button on the Technician Mode screen. When the button (named **btnBacktoScreen1**) is clicked, then Screen1, or the Home Screen, is opened. Blocks like these are what drive the entire operation of the app. Each button and label in the interface has different logic to perform different tasks when a particular even occurs, some more complicated than others. The view of all blocks in the entire app design is found in Appendix E.



```
when btnBacktoScreen1 ▾ .Click
do  open another screen  screenName  Screen1 ▾
```

Figure 5.2 *BACK Button Logic from Technician Screen, Blocks View*

Figure 5.3 (next page) breaks the home screen down into four major segments. The first segment (1) is the Bluetooth connect/disconnect controls. The **CONNECT** button pulls up a list view of all the Bluetooth devices that are within range of the smartphone and allows the user to select the device they would like to connect to. Once connected, the **DISCONNECT** button breaks the connection to whichever device the smartphone was connected to. The second segment (2) is the connection status and information display. Once the smartphone is connected to a device, the "Status" bar turns green, and the status changes from "Disconnected" to "Connected." The

“Connection Information” bar will then display the name and MAC Address of the device connected to the smartphone. The third segment (3) is the direct connect and mode select controls, along with the battery status display. The **DIRECT** button allows the user to directly connect to the last (previous) connected device without viewing the entire list of Bluetooth devices within range. The **MODE** button prompts the user to decide if they would like to enter “Technician Mode,” which provides not only the pressure values for each sensor, but the force and voltage values as well, along with a few metrics for viewing battery life. The battery status display currently displays the battery voltage. In future app revisions, this status would be changed to a percentage rather than a voltage, to give the user a better idea of how much longer they can use the device before it needs to be recharged. The final segment (4) is the pressure value display. This is where the pressure values sent from the Bluetooth module are displayed for the user to view and analyze.

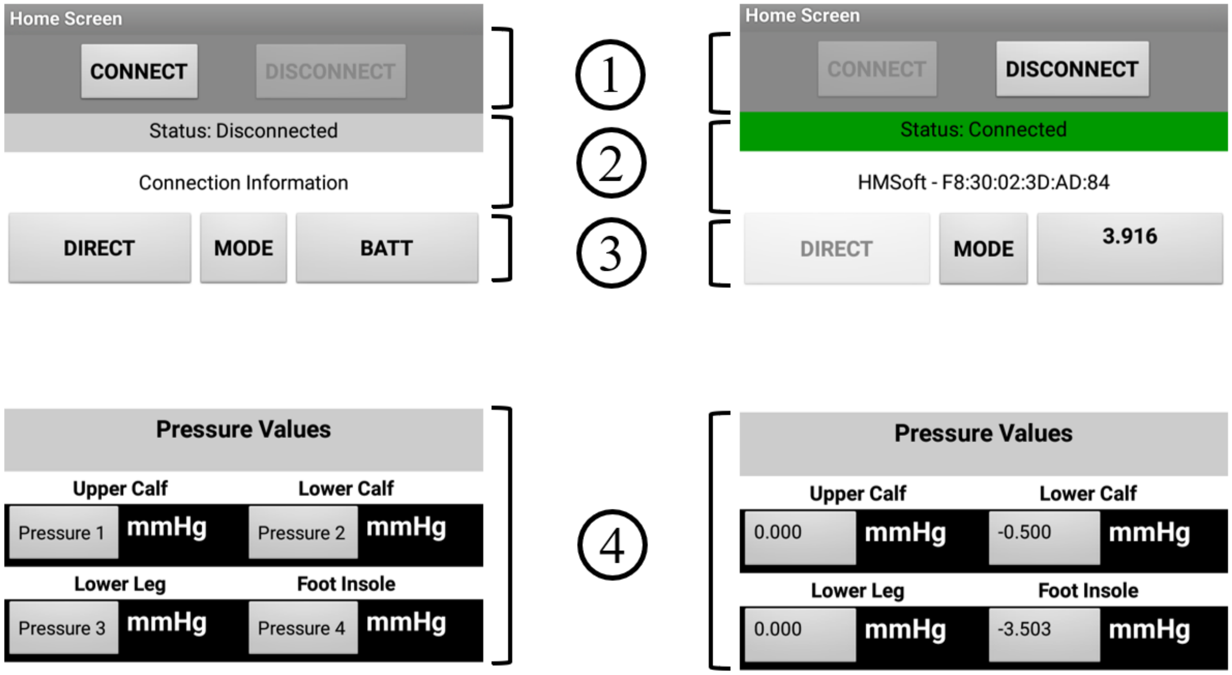
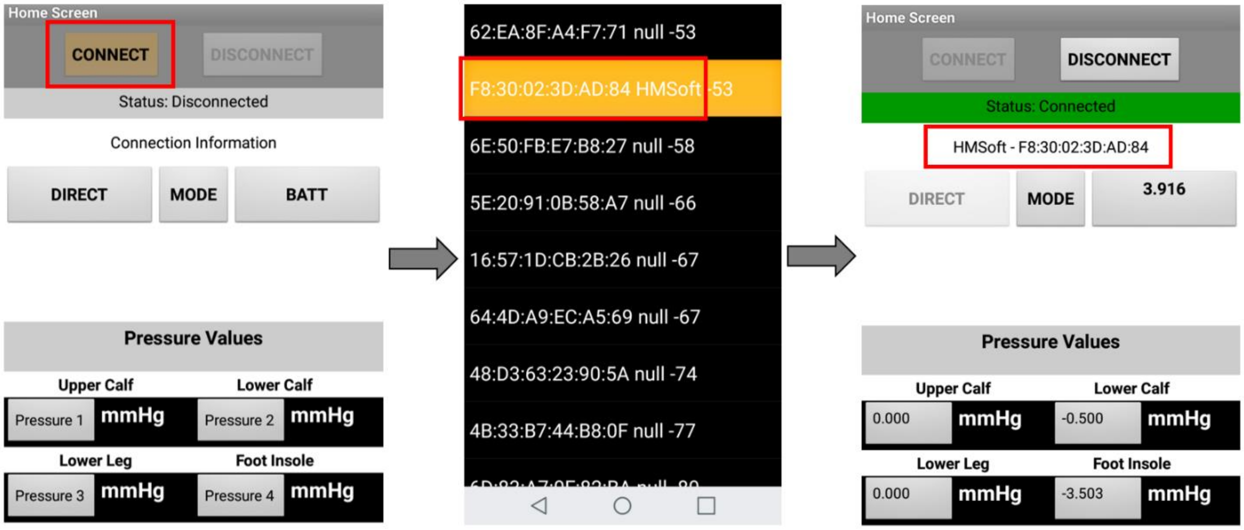


Figure 5.3 *Home Screen Breakdown by Section (Left – Disconnected, Right – Connected)*



Status: Disconnected

```
62:EA:8F:A4:F7:71 null -53
F8:30:02:3D:AD:84 HMSoft -53
6E:50:FB:E7:B8:27 null -58
5E:20:91:0B:58:A7 null -66
16:57:1D:CB:2B:26 null -67
64:4D:A9:EC:A5:69 null -67
48:D3:63:23:90:5A null -74
4B:33:B7:44:B8:0F null -77
```

Status: Connected

HMSoft - F8:30:02:3D:AD:84

3.916

Upper Calf: 0.000 mmHg
Lower Calf: -0.500 mmHg
Lower Leg: 0.000 mmHg
Foot Insole: -3.503 mmHg

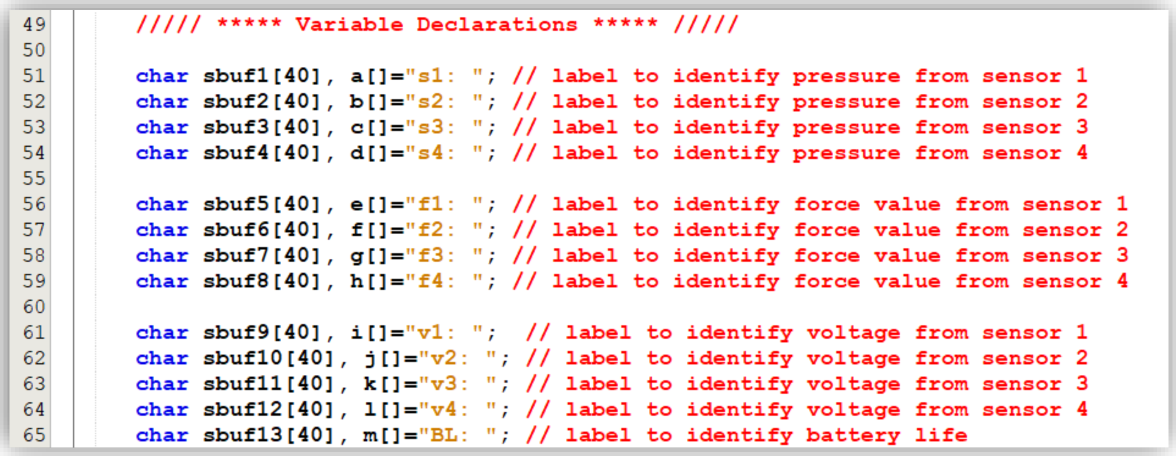
Figure 5.4 *Connecting to a Device Using CONNECT Button and Device List*

Figure 5.4 shows the process of connecting to a Bluetooth device within range of the smartphone using the app. When no device is connected, the status bar is gray, and says “Disconnected.” The DISCONNECT button is unavailable at this time since there is nothing to disconnect from. Once the CONNECT button is clicked, the list view of nearby devices pops up for the user to select the device they would like to connect to. In the list view, each device is characterized by its MAC address, its name, and the signal strength between the smartphone and the device. The signal strength is in units of dBm or decibel milliwatts [25] and is used as a measure of absolute signal power. “Signal strengths can range from approximately -30 dBm to -110 dBm. The closer that number is to 0, the stronger the signal,” [26]. The HM-10 (by default) has the name HMSoft, and when the smartphone is right next to the device, has a signal strength of -53 dBm, which is considered to be good. If the signal strength is, “from 0 to -60 [dBm] it’s a good signal strength. Any value that is -70 [dBm] or less is bad,” [24], and anything lower than -85 dBm is considered unusable [26]. Once the user selects the HMSoft as their device to connect to, the smartphone connects, and the status bar turns green and changes the status to “Connected.” The connection information is also updated at this time to show the device name and the MAC address, or HMSoft – F8:30:02:3D:AD:84. Once connected, the pressure values begin to update from the sensor readings in real time, and the battery voltage appears in the battery status box, as seen in Fig. 5.4 (previous page).

5.2 RECEIVING AND DISPLAYING DATA

In this section, the process by which data is received and displayed on the smartphone application is discussed. A series of steps are carefully taken in both the microcontroller program and the smartphone application program in order for data to properly be received and displayed. The first set of instructions is programmed into the microcontroller C code (full code in Appendix D). Outside of the main "while" loop, 13 individual empty **character arrays** are declared: 4 for the sensor output voltages, 4 for the calculated forces applied to each sensor, 4 for the calculated pressures for each sensor, and 1 for the battery voltage. In the C programming language, character arrays are used in place of strings. An empty character array is therefore synonymous with an empty string. A character array has all the same properties that a string does, and thus, character arrays can be used as input arguments to functions that operate on strings, such as the **printf** and **sprintf** functions.

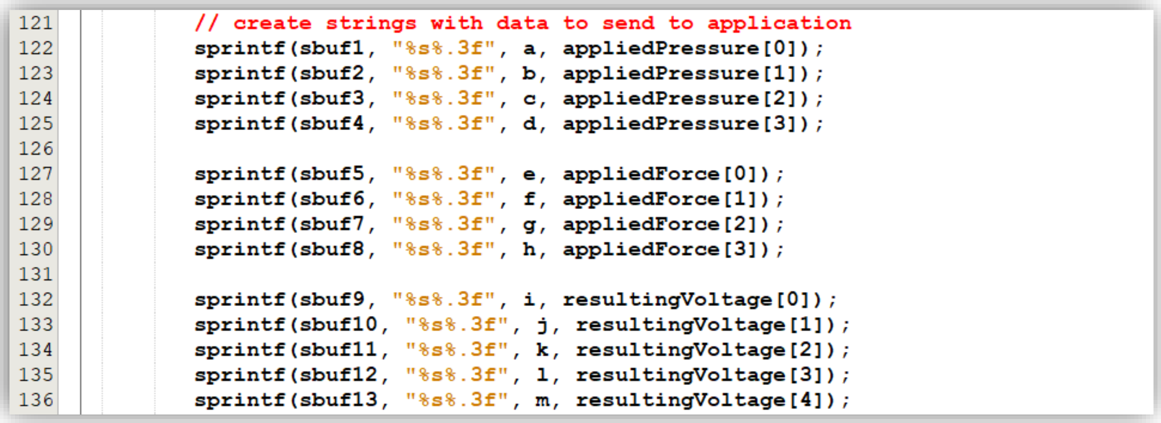
Along with the declarations for the empty character arrays is another set of 13 individual character arrays which are not empty, but instead contain **data labels** for each of the 13 values of interest (4 applied force values, 4 sensor output voltage values, 4 pressure values, 1 battery voltage). These data labels are used to help the smartphone application distinguish between received values and to determine where to properly display the values it receives. For example, Fig. 5.5 below shows the data labels for the pressure values, the force values, the sensor voltage values, and the battery voltage as they are initialized in the C code. The sensor number increases as the sensor location moves further down the leg. The “s1: ” label belongs to the sensor at the upper calf, while the “s4: ” label belongs to the sensor at the insole of the foot (not included in prototype build). The same numbering convention applies for the force data labels and the output voltage data labels. Thus, the labels “s1: ”, “f1: ”, and “v1: ” correspond to the sensor pressure reading, force reading, and output voltage for the sensor at the upper calf, respectively.



```
///// ***** Variable Declarations ***** /////

char sbuf1[40], a[]="s1: "; // label to identify pressure from sensor 1
char sbuf2[40], b[]="s2: "; // label to identify pressure from sensor 2
char sbuf3[40], c[]="s3: "; // label to identify pressure from sensor 3
char sbuf4[40], d[]="s4: "; // label to identify pressure from sensor 4

char sbuf5[40], e[]="f1: "; // label to identify force value from sensor 1
char sbuf6[40], f[]="f2: "; // label to identify force value from sensor 2
char sbuf7[40], g[]="f3: "; // label to identify force value from sensor 3
char sbuf8[40], h[]="f4: "; // label to identify force value from sensor 4

char sbuf9[40], i[]="v1: ";  // label to identify voltage from sensor 1
char sbuf10[40], j[]="v2: "; // label to identify voltage from sensor 2
char sbuf11[40], k[]="v3: "; // label to identify voltage from sensor 3
char sbuf12[40], l[]="v4: "; // label to identify voltage from sensor 4
char sbuf13[40], m[]="BL: "; // label to identify battery life
```

Figure 5.5 *Character Array Variable Declarations for Data Labels*

Within the main “while” loop of the program, the output voltage of the sensor/amplifier unit is converted to digital, processed, and used to calculate the applied force and the associated applied pressure. These variables are stored as **float** type values, and their variables are named appliedPressure[n], appliedForce[n], and resultingVoltage[n], where n is the sensor number minus 1 (if sensor 1 is being referenced, n = 0). Once these values have been obtained, they are appended to their respective data labels and converted to one large string using the **sprintf** function. This function allows multiple data types (in this case a char array and a float value) to be appended and stored as one formatted string [27]. For example, char array **sbuf1** is originally an empty char array of size 40 x 1, initialized in Fig. 5.5 above. Line 122 of the code in Fig. 5.6 below takes the pressure value measured by sensor 1 (since n = 0) and the contents of char array **a** (the data label for pressure at sensor 1) and appends them to the empty char array **sbuf1**. Say for the sake of the example that the value of appliedPressure[0] is 1.200 in units of mmHg. The resulting string or char array is “s1: 1.200” which is now stored in sbuf1. This process is repeated for all the applied pressure values, applied force values, and sensor output voltage values so that all data of interest is appended to the appropriate data label.

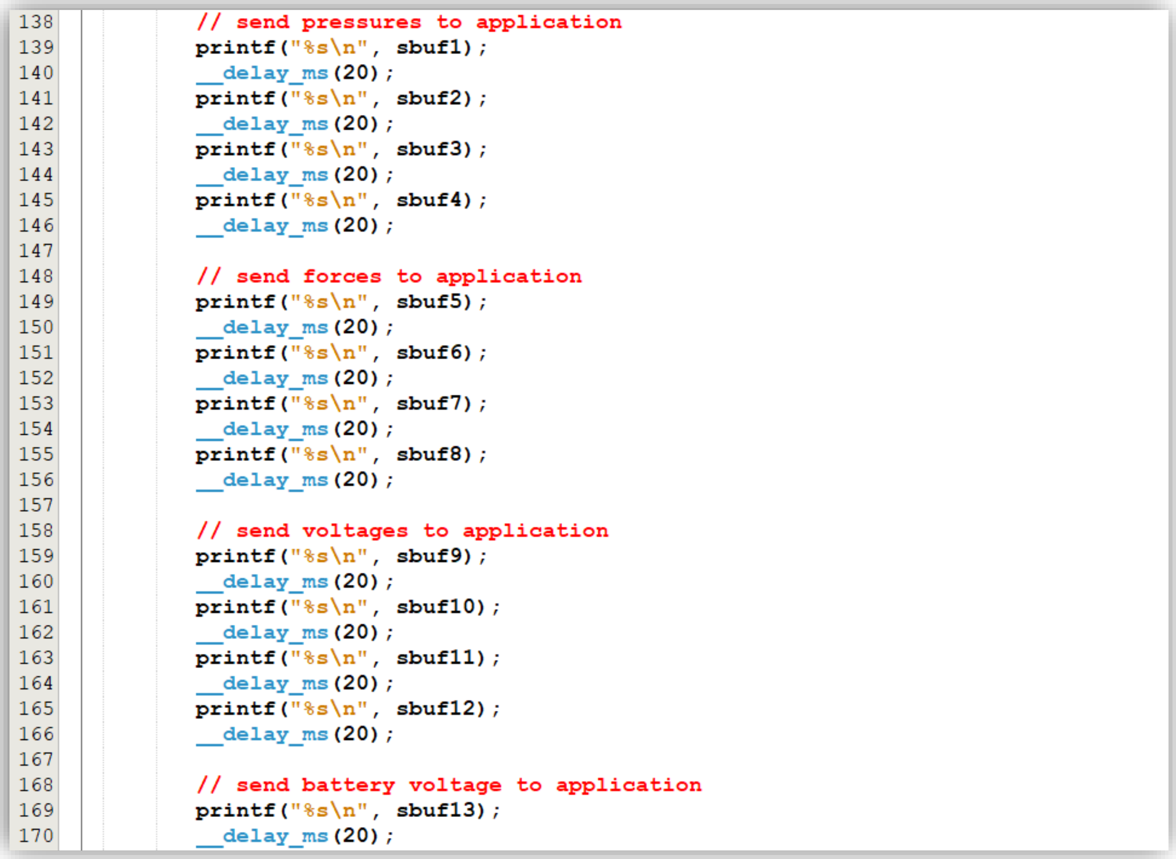


```
121            // create strings with data to send to application
122            sprintf(sbuf1, "%s%.3f", a, appliedPressure[0]);
123            sprintf(sbuf2, "%s%.3f", b, appliedPressure[1]);
124            sprintf(sbuf3, "%s%.3f", c, appliedPressure[2]);
125            sprintf(sbuf4, "%s%.3f", d, appliedPressure[3]);
126
127            sprintf(sbuf5, "%s%.3f", e, appliedForce[0]);
128            sprintf(sbuf6, "%s%.3f", f, appliedForce[1]);
129            sprintf(sbuf7, "%s%.3f", g, appliedForce[2]);
130            sprintf(sbuf8, "%s%.3f", h, appliedForce[3]);
131
132            sprintf(sbuf9, "%s%.3f", i, resultingVoltage[0]);
133            sprintf(sbuf10, "%s%.3f", j, resultingVoltage[1]);
134            sprintf(sbuf11, "%s%.3f", k, resultingVoltage[2]);
135            sprintf(sbuf12, "%s%.3f", l, resultingVoltage[3]);
136            sprintf(sbuf13, "%s%.3f", m, resultingVoltage[4]);
```

Figure 5.6 *Appending Processed Data to Data Labels*

At this stage, all data is ready to be transmitted to the Bluetooth module serially. The **printf** function is used to send formatted text to any peripheral or location on the MCU [27]. In this case, the peripheral being used is the UART, since the MCU UART is connected to the BLE module UART. The Bluetooth module receives the serial data from the microcontroller and transmits whatever it receives wirelessly via Bluetooth to the smartphone (discussed in detail in Ch. 4). The printf instructions in the code of Fig. 5.7 (next page) are separated by delays of 20 milliseconds.

The delay is used to make sure that the smartphone application receives each string individually. If the delay time is too short between transmitted strings, an error occurs as a result of the smartphone application being incapable of distinguishing between two individual strings. It will count two distinct strings as one big string and cause errors in the logic of the application. Different delay times were tested, and the shortest delay time to eliminate errors was determined to be 20 milliseconds.

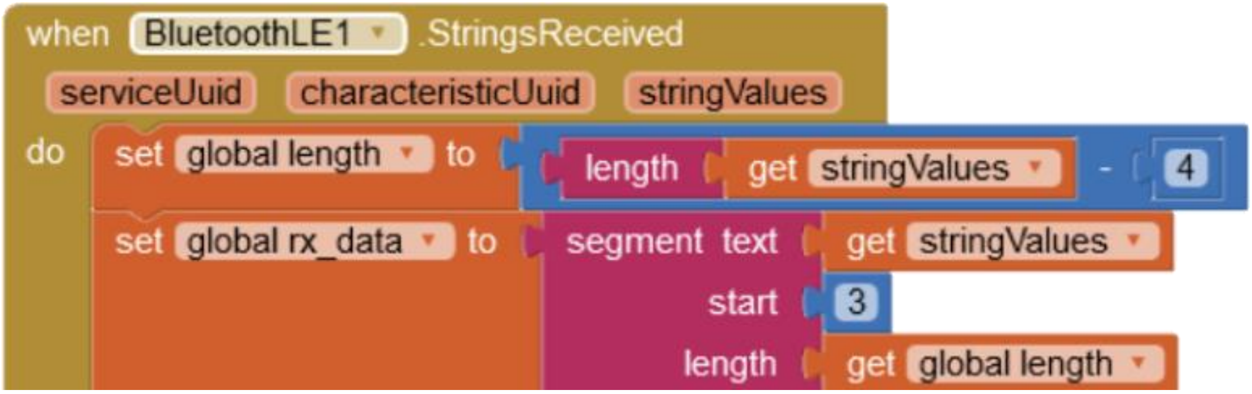


```
// send pressures to application
printf("%s\n", sbuf1);
__delay_ms(20);
printf("%s\n", sbuf2);
__delay_ms(20);
printf("%s\n", sbuf3);
__delay_ms(20);
printf("%s\n", sbuf4);
__delay_ms(20);

// send forces to application
printf("%s\n", sbuf5);
__delay_ms(20);
printf("%s\n", sbuf6);
__delay_ms(20);
printf("%s\n", sbuf7);
__delay_ms(20);
printf("%s\n", sbuf8);
__delay_ms(20);

// send voltages to application
printf("%s\n", sbuf9);
__delay_ms(20);
printf("%s\n", sbuf10);
__delay_ms(20);
printf("%s\n", sbuf11);
__delay_ms(20);
printf("%s\n", sbuf12);
__delay_ms(20);

// send battery voltage to application
printf("%s\n", sbuf13);
__delay_ms(20);
```

Figure 5.7 *Transmitting Data All Serially Through UART*

When the smartphone receives a string, it is looking for a particular data label. The application is programmed to parse through received strings looking for data labels, and to display the data in different locations on the application interface based on the received data labels.

Continuing the same example, say the smartphone has just received the text contained in sbuf1, or ["s1: 1.200"]. The quotation marks and square brackets are added to the text by the MIT app inventor software each time a string is received via Bluetooth. Thus, to get the true length of the string received, 4 needs to be subtracted from the string length (2 for brackets, 2 for quotation marks). The variable **stringValues** stores the full string received from the Bluetooth module including the added brackets and quotation marks. The global **length** variable is used to store the actual length of the string after the value 4 is subtracted for the added brackets and quotation marks. The global **rx_data** variable is used to store the contents of the received string, starting at the third character (after the first bracket and quotation mark) and with the length of the global **length** variable. The steps described previously are shown in block logic format in Fig. 5.8 below.

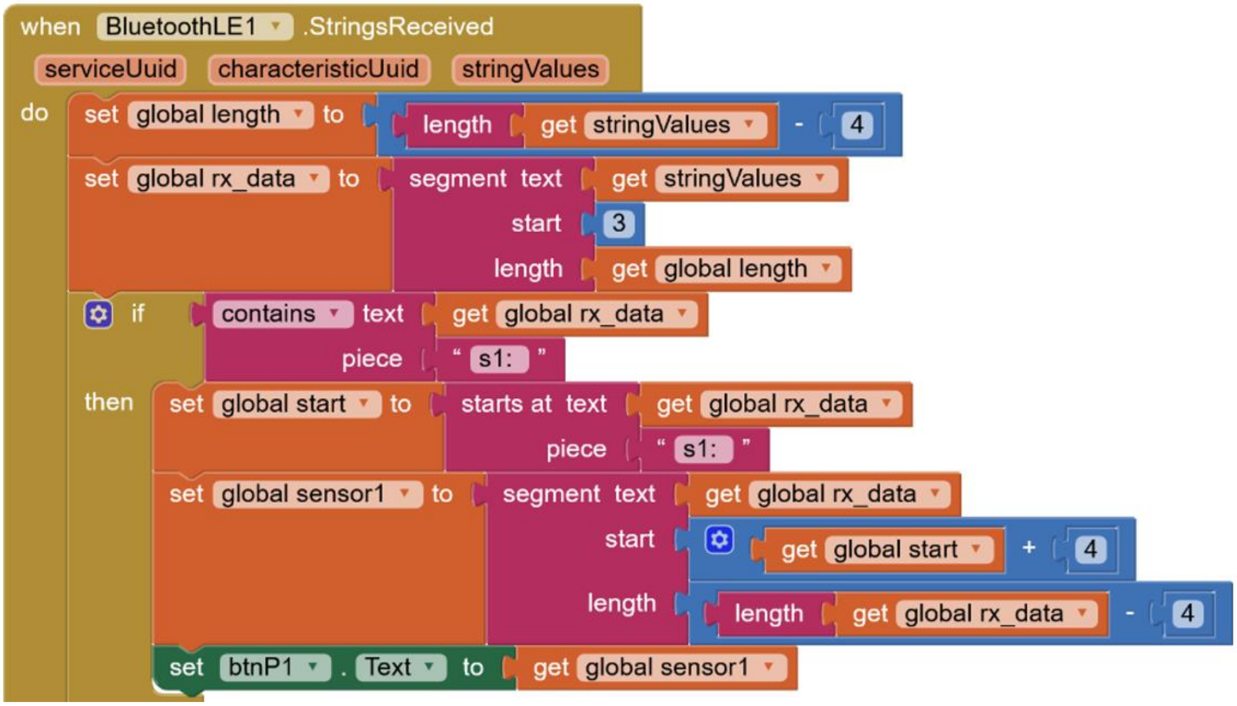


```
when BluetoothLE1 .StringsReceived
  serviceUuid  characteristicUuid  stringValues
do  set global length to  length  get stringValues  -  4
    set global rx_data to  segment  text  get stringValues
                                    start  3
                                    length  get global length
```

Figure 5.8   *Blocks Logic for Receiving a String from BLE Module*

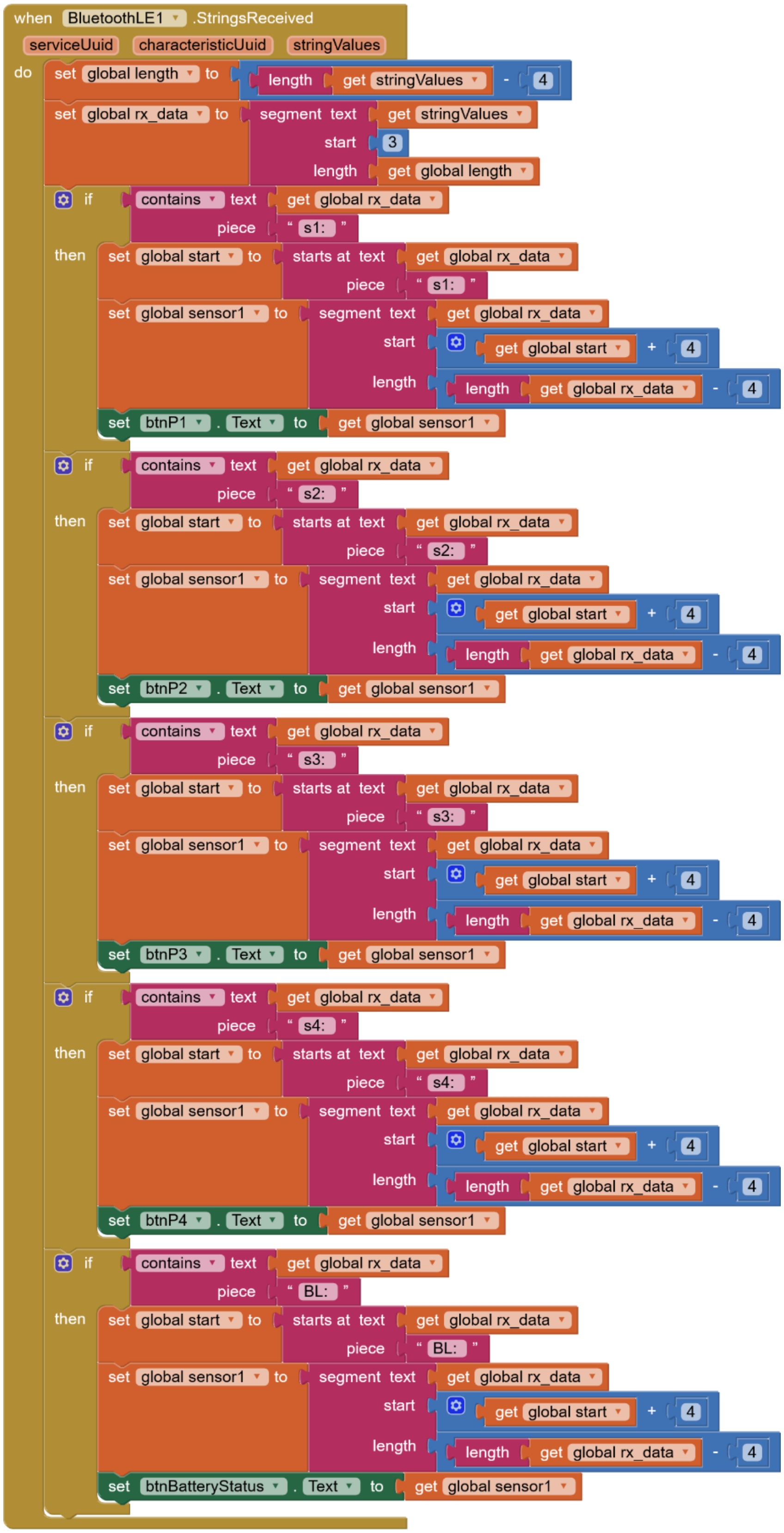
When a string a received and the instructions of Fig. 5.8 have been completed, the application program enters an IF statement, shown in Fig. 5.9 (next page). The IF statement checks the received string for a particular data label, described previously. In the case of Fig. 5.9, the IF statement is looking for the character sequence "s1: " or the data label corresponding to the pressure value from sensor 1. Continuing on with the same example where the pressure at sensor

1 is 1.200 mmHg, IF the string received contains the data label “s1: ”, then whatever text follows the data label is set as the text on the “Pressure 1” label. Logic blocks are used to make sure the value displayed for Pressure 1 does not include any part of the data label or the closing bracket and quotation marks. This process in the ongoing example would result in a displayed pressure of 1.200 mmHg at the Pressure 1 label in the application interface. The blocks logic for this process is shown below in Fig. 5.9. This process is repeated in the format of ELSE IF blocks of logic appended to the blocks of Fig. 5.9. The only difference is that instead of searching for the data label “s1: ”, the ELSE IF blocks search for “s2: ”, “s3: ”, “s4: ”, and “BL: ”, for sensor 2 pressure, sensor 3 pressure, sensor 4 pressure, and battery voltage, respectively. The entire *Blocks* view for the reception and proper display of the data is given by Fig. 5.10 on the page to follow. The *Blocks* view for the entire application is given in Appendix E.



```
when BluetoothLE1 .StringsReceived
  serviceUuid  characteristicUuid  stringValues
do  set global length to  length get stringValues - 4
    set global rx_data to  segment text  get stringValues
                                   start  3
                                  length  get global length
    if  contains text  get global rx_data
                piece  " s1: "
    then  set global start to  starts at text  get global rx_data
                                        piece  " s1: "
          set global sensor1 to  segment text  get global rx_data
                                        start  get global start + 4
                                       length  length get global rx_data - 4
          set btnP1 . Text to  get global sensor1
```

Figure 5.9 *Blocks Logic for Extracting and Displaying Relevant Data*



```
when BluetoothLE1 .StringsReceived
  serviceUuid  characteristicUuid  stringValues
do  set global length to  length  get stringValues  - 4
    set global rx_data to  segment text  get stringValues
                                   start  3
                                  length  get global length
    if  contains text  get global rx_data
                piece  " s1: "
    then  set global start to  starts at text  get global rx_data
                                        piece  " s1: "
          set global sensor1 to  segment text  get global rx_data
                                         start  get global start + 4
                                        length  length  get global rx_data  - 4
          set btnP1 . Text to  get global sensor1
    if  contains text  get global rx_data
                piece  " s2: "
    then  set global start to  starts at text  get global rx_data
                                        piece  " s2: "
          set global sensor1 to  segment text  get global rx_data
                                         start  get global start + 4
                                        length  length  get global rx_data  - 4
          set btnP2 . Text to  get global sensor1
    if  contains text  get global rx_data
                piece  " s3: "
    then  set global start to  starts at text  get global rx_data
                                        piece  " s3: "
          set global sensor1 to  segment text  get global rx_data
                                         start  get global start + 4
                                        length  length  get global rx_data  - 4
          set btnP3 . Text to  get global sensor1
    if  contains text  get global rx_data
                piece  " s4: "
    then  set global start to  starts at text  get global rx_data
                                        piece  " s4: "
          set global sensor1 to  segment text  get global rx_data
                                         start  get global start + 4
                                        length  length  get global rx_data  - 4
          set btnP4 . Text to  get global sensor1
    if  contains text  get global rx_data
                piece  " BL: "
    then  set global start to  starts at text  get global rx_data
                                        piece  " BL: "
          set global sensor1 to  segment text  get global rx_data
                                         start  get global start + 4
                                        length  length  get global rx_data  - 4
          set btnBatteryStatus . Text to  get global sensor1
```

Figure 5.10 *Complete Blocks View for Receiving and Displaying Pressure Data*

5.3 CHAPTER 5 SUMMARY

Chapter 5 provides an overview of the smartphone application used to view the data obtained by the compression sensing garment. A breakdown of the app interface is provided along with an in-depth description of the process of data reception and display.

- **Section 5.1** gives a look at the user interface for the application, along with a brief tutorial-like segment on how to connect to a device, and what the interface should look like when a device is connected to successfully.
- **Section 5.2** details the operation of both the MCU C code to transmit formatted data and the smartphone application block logic used to receive and display the formatted data in the proper location on the app interface.

CHAPTER 6: POWERING THE SYSTEM

6.1 POWER SOURCE

The entire system is powered by a 3.7V rechargeable Lithium-Polymer (Li-Po) battery. The battery has a capacity of 380mAh. This means that the amount of current drawn (in mA) from the battery multiplied by the amount of time (in hours) for which that current is drawn cannot exceed 380mA·h. For example, the battery will supply 380mA for 1 hour, or 38mA for 10 hours, or 3.8mA for 100 hours. Rechargeable Li-Po batteries are a good choice for this application since they are rechargeable, relatively cheap, and come in a variety of voltages, capacities, shapes, and sizes. The battery shown in Fig. 6.1 below is small (35mm x 15mm) and lightweight (11g) and does not heat up to any uncomfortable temperature during normal operation. It is also convenient to connect because the attached connector fits 100 mil-spaced male header pins snug and makes a great contact that is not too loose and not too tight. Other similar LiPo batteries are made by the same manufacturer and other manufacturers in voltages that are multiples of 3.7V (7.4V, 11.1V, etc.) and capacities as great as 5000mAh and greater. These batteries get more and more expensive and larger and larger as the capacity and the voltage increases. In the current system, the battery of Fig. 6.1 can power the system on single charge for around 8 hours while running continuously.

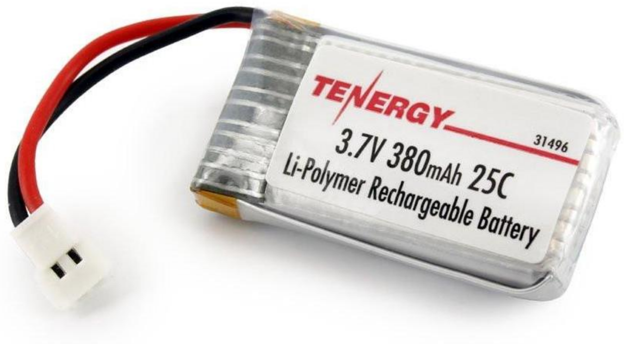


Figure 6.1 *3.7V, 380mAh Li-Po Rechargeable Battery*

6.2 VOLTAGE REGULATION

The microcontroller is a low-power unit which can operate with supply voltages between 1.8V and 3.6V. The HM-10 and HM-11 Bluetooth modules can operate with supply voltages between 2.2V and 3.7V. Thus, a solid supply voltage that works for both the MCU and the BLE module is 3.3V. Though the battery is rated at a voltage of 3.7V, when fully charged, it is typically at a greater voltage, around 4V. As the battery discharges, the voltage output of the battery steadily decreases until the battery is nearly completely discharged, at which point the voltage drops more rapidly and the battery dies. To maintain the proper supply voltage for both the microcontroller and the Bluetooth module, a voltage regulator with a 3.3V output is used. In the case of Fig. 6.2 below, a buck-boost switching power supply [7] is used to supply a constant 3.3V output while being powered by a discharging Li-Po battery.

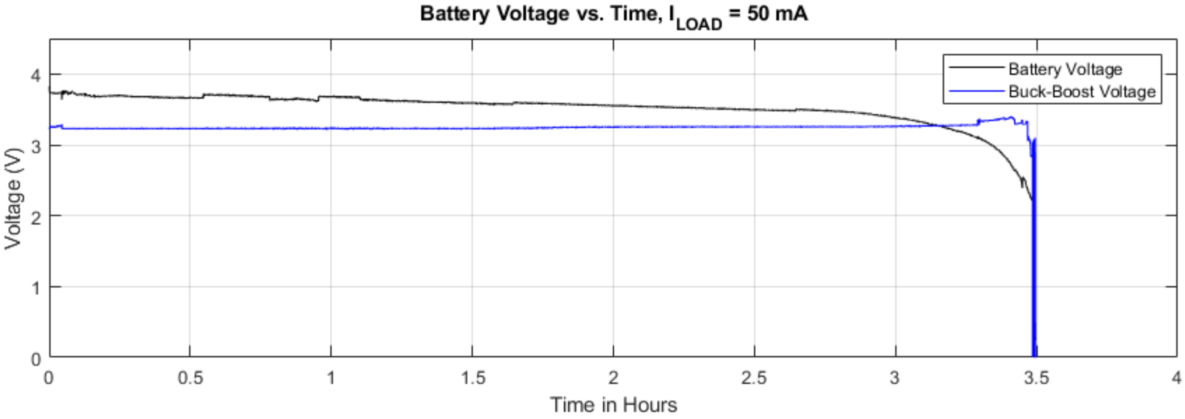


Figure 6.2 *Drainage of 3.7V Li-Po 380mAh Battery Over Time,* I_{LOAD} *= 50mA*

When the battery voltage is greater than the 3.3V desired output, the regulator is in *buck mode*, where it *bucks* the higher input voltage down to the desired output voltage. When the battery voltage is lower than the 3.3V desired output, the regulator is in *boost mode*, where it *boosts* the lower input voltage up to the desired output voltage. The particular buck-boost regulator used in the design of the system and thus used to generate Fig. 6.2 above is the LTC3531-3.3 DC/DC converter by Linear Technology. This buck-boost regulator has a third mode of operation called "4-Switch mode" where 4 on-chip switches connected to the external inductor operate to maintain the 3.3V output voltage when the input is close to the desired output voltage (Vin is less than 400mV less than 3.3V or Vin is less than 800mV greater than 3.3V). The operation of the regulator in "4-Switch mode" is given from the datasheet [7] in Fig. 6.3 below.

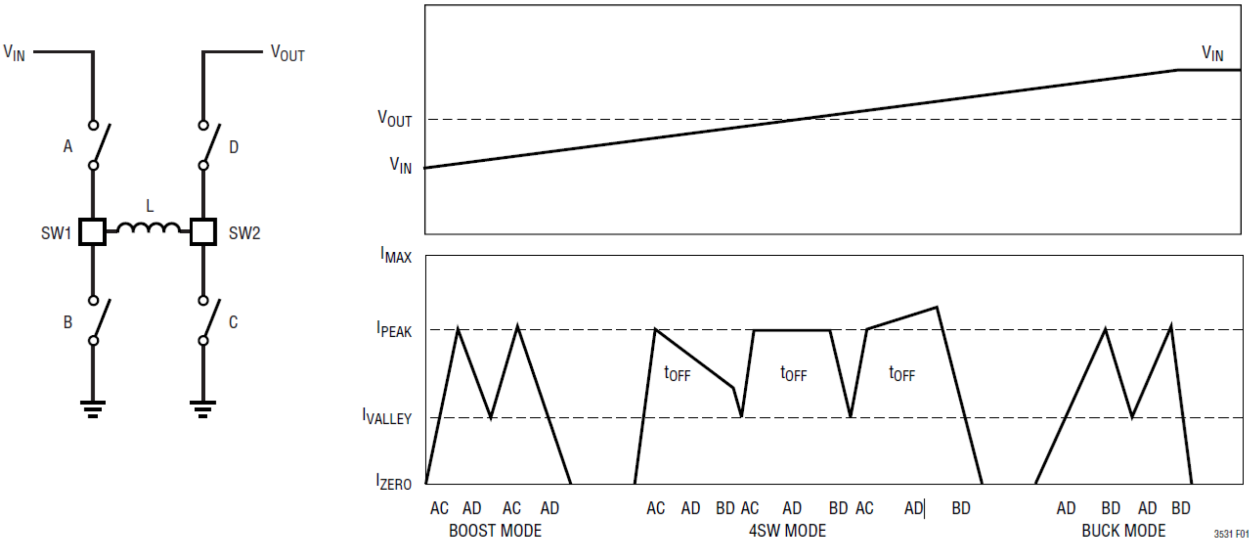


Figure 6.3 *4-Switch Mode Operation for LTC3531 Buck-Boost Regulators [7]*

The LTC3531 family of buck-boost converters can supply 3.3V DC to loads up to 200mA. However, the input current to the buck-boost is a function of the load current and the voltage being supplied to the buck-boost, in this case the battery voltage. When the battery voltage is greater than the desired output voltage, the current pulled from the battery (the buck-boost input current) is less than the load current. This is when the device is operating in *buck mode*. When the battery voltage is near equal to the desired output voltage, the device is in 4-Switch mode and the input and output currents of the buck-boost are nearly the same. When the battery voltage is less than the desired output voltage, the device is in *boost mode* and the current supplied to the buck-boost by the battery is greater than the load current. The lower the battery voltage drops, the more current the buck-boost has to pull from it in order to maintain the desired output voltage while driving the load. Thus, the device is most efficient in *buck mode* and least efficient in *boost mode*. The efficiency is plotted against Vin for each of the three operating modes from the device datasheet and can be seen in Fig. 6.4 below.

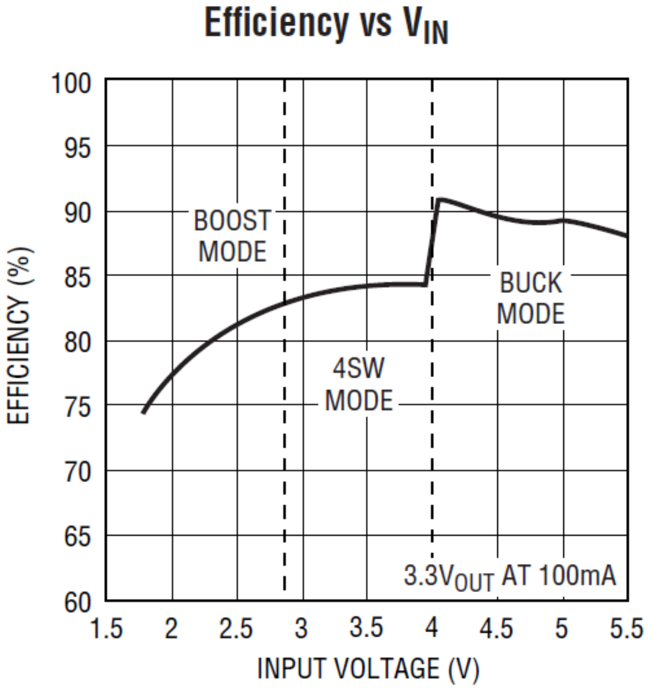


Figure 6.4 *Efficiency vs.* V_{IN} *for LTC3531 Buck-Boost SPS [7],* I_{LOAD} *= 100 mA*

6.3 POWER CONSUMPTION AND BATTERY LIFE

As the battery discharges while powering the circuit, the battery voltage drops steadily, see Fig. 6.2. The buck-boost needs to pull more and more current from the decreasing voltage source to drive the load while maintaining the desired output voltage of 3.3V. As the buck-boost pulls more and more current from the battery, the battery dies faster and faster, pulling its output voltage down even quicker. This is the reason for the voltage drop-off in Fig. 6.2. This cycle repeats until the battery is dead. With a buck-boost load current of around 50mA (much larger than the actual current drawn by the system), Fig. 6.2 shows that a full battery lasts for almost precisely 3.5 hours until it is completely discharged. A battery-monitoring voltage divider is constructed using two series resistors of 976 kΩ resistance (tolerance of ± 5%) each. Since the input to the divider is the battery voltage, and the resistance of each resistor in the divider is ideally equal, the output of the divider is roughly $V_{BATTERY}/2$. The output of the divider is connected to one of the remaining ADC input channels on the MCU and the software multiplies the sensed ADC voltage by 2 to provide the user with a true reading of the current battery voltage at any given time.

COMPONENT	CURRENT DRAW (mA)	CURRENT DRAW (%)
PIC18LF26K22 MCU	1.45 mA	8.71%
HM-10 BLE MODULE	**9.25 mA**	**55.59%**
UPPER CALF SENSOR	1.95 mA	11.73%
LOWER CALF SENSOR	1.98 mA	11.91%
LOWER LEG SENSOR	1.99 mA	11.97%
TOTAL:	**16.62 mA**	**100.0%**

Table 6.1 *Current Draw Distribution for Major System Components*

Table 6.1 gives a breakdown of the major components of the system and how much current each component draws during normal operation. The leftmost column gives the name of the component, the middle column gives the actual DC current value (in mA), and the rightmost column gives the percentage of the total current drawn by each component. Each of the sensor boards draw around 2 mA and just under 12% of the total current per sensor. Combined, the 3 sensor/amplifier units draw around 36% of the total current. The PIC18LF26K22 is a low-power, high-performance microcontroller [12] and only draws around 1.5 mA of current during normal operation, equivalent to less than 9% of the total current. The bulk of current is drawn by the HM-10 Bluetooth module and the associated Bluetooth connection status LED. When the Bluetooth module is connected to another Bluetooth device and the LED is ON, the module pulls around 9.25 mA constantly, or over 55% of the total current. Section 7.1.4 gives further analysis on other BLE modules that can be used to limit this current and will allow the battery to last longer in the system.

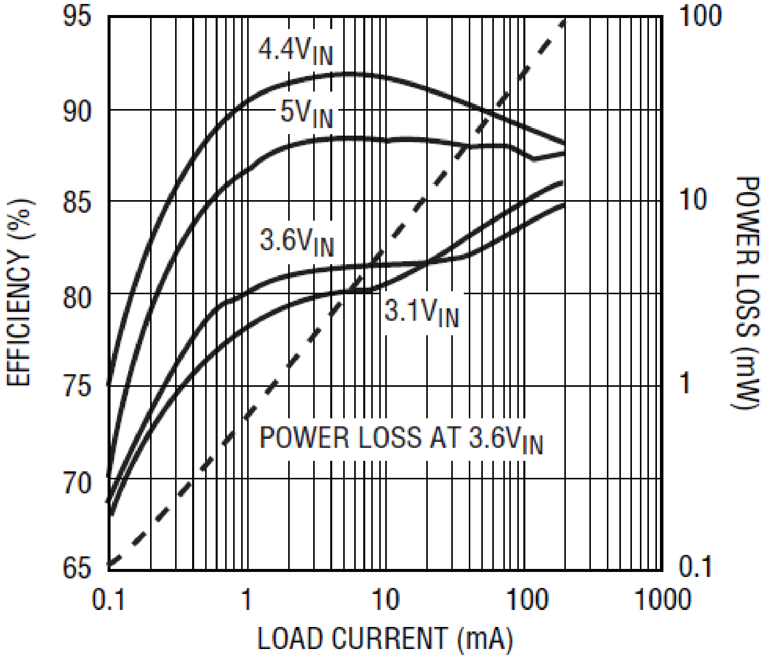


Figure 6.5 *Efficiency vs. Load Current for Different Input Voltages LTC3531 [7]*

From Table 6.1, the load current when the system is ON and connected to a smartphone via Bluetooth is around 16.6 mA. The plot from the buck-boost datasheet seen in Fig. 6.5 above shows the efficiency vs. load current for different values of V_{IN}, where in this case V_{IN} is the battery voltage. For battery voltages above 3V and load currents above 10 mA, the efficiency is around 80% or better. *Efficiency* is “the ratio of total output power to total input power, expressed in percent,” [23]. In other words, the efficiency metric is used to determine how much of the input power is being converted into output power delivered to the load circuitry and not burned up in the power supply. From the plot of Fig. 6.6 (next page), an experimental value can be determined for the efficiency in the system. For example, let the battery voltage be 3.6V and the load current is 16.2 mA. In Fig. 6.6, the current drawn from the battery with a 16.2 mA load is around 20 mA for a battery voltage of 3.6V. Thus, the input power to the buck-boost is calculated by

$$P_{IN} = V_{BAT} \cdot I_{BAT} = 3.6V \cdot 20\ mA = 72\ mW \tag{6.1}$$

The buck-boost output power is calculated by

$$P_{OUT} = V_{OUT} \cdot I_{LOAD} = 3.3V \cdot 16.2\ mA = 53\ mW \tag{6.2}$$

With the input power and the output power calculated, the efficiency can be determined by taking the ratio of output power to input power, or

$$E(\%) = {}^{P_{OUT}}/_{P_{IN}} \cdot 100 = \left({}^{53\ mW}/_{72\ mW}\right) \cdot 100 = 74\% \tag{6.3}$$

The result of eq. (6.3) is an efficiency of 74%, meaning that 74% of input power is delivered to the load, and the other 26% (around 19 mW) are burned up in the buck-boost as it regulates the output voltage to 3.3V while driving the 16.2 mA load.

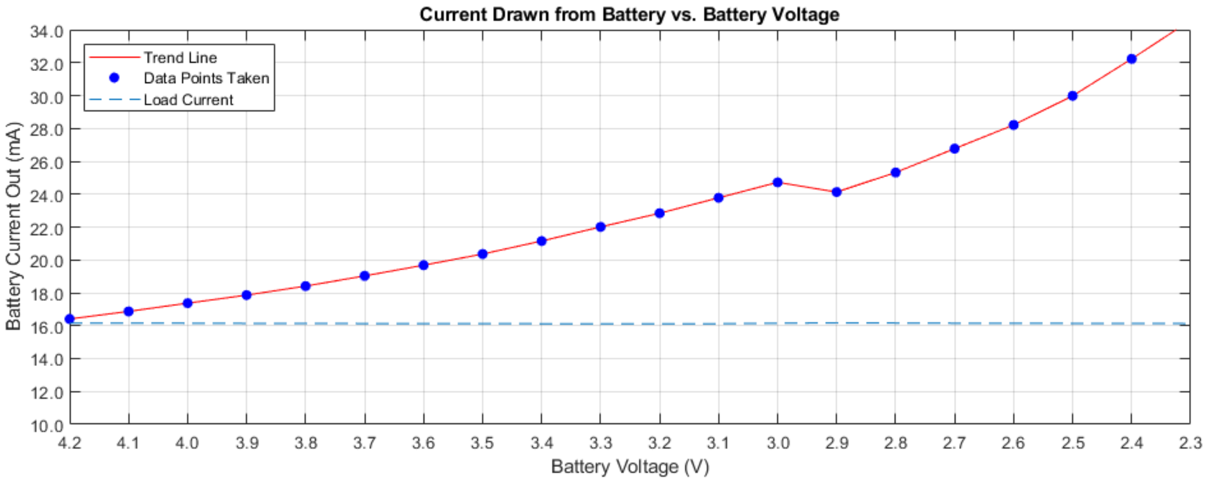


Figure 6.6 *Battery Current Out vs. Battery Voltage, I_{LOAD} = 16.2 mA*

Figure 6.6 was generated using a power supply and a 198.7Ω resistor, used to pull ideally 16.6 mA from the buck-boost to simulate the system load from Table 6.1. The power supply voltage was used to simulate the battery voltage and was decreased in steps of 100 mV to simulate the battery voltage dropping as the battery discharges. The plot shows how the current drawn by the buck-boost increases as the battery voltage decreases:

- At a battery voltage of 4.2V, the buck-boost pulls just over 16.2 mA of current from the battery, and the power supply is **operating in buck mode**.
- At a battery voltage of 3.3V, the buck-boost pulls 22 mA of current from the battery and is **operating in 4-switch mode**.
- At a battery voltage of 2.6V, the buck-boost pulls around 28 mA from the battery and is **operating in boost mode**.

Once the buck-boost enters boost mode, the battery dies very quickly since the current draw is high, and the buck-boost continues to pull larger and larger currents from the battery as its voltage continues to drop, as mentioned at the beginning of section 6.3. The table below shows each of the data points plotted in Fig. 6.6, along with the mode of operation corresponding with the current battery voltage (input voltage to the buck-boost). The hitch in the trend between 3.0V and 2.9V for the battery voltage results from the switching from 4SW mode to boost mode, as the different hardware used in the power supply for boost mode initially draws less current at 2.9V than 4SW mode hardware does at 3.0V. The efficiency is calculated and displayed in the table. The efficiency is at a maximum when the battery voltage is the highest (77.6% at 4.2V) and is at a minimum when the battery voltage is the lowest (67.0% at 2.3V). In general, the efficiency for this 16.2 mA load is never greater than 80%, and a more efficient regulator should be used in future revisions.

Battery Voltage (V)	*Battery Current (mA)*	*Load Current (mA)*	*Operation Mode*	*VDD (V)*	*Efficiency (%)*
4.2	16.41	16.16	BUCK	3.31	77.6%
4.1	16.87	16.16	BUCK	3.31	77.4%
4.0	17.37	16.15	BUCK	3.31	76.9%
3.9	17.86	16.14	4SW	3.31	76.6%
3.8	18.41	16.14	4SW	3.31	76.3%
3.7	19.03	16.13	4SW	3.31	75.7%
3.6	19.68	16.13	4SW	3.31	75.2%
3.5	20.36	16.13	4SW	3.31	74.8%
3.4	21.15	16.12	4SW	3.30	74.0%
3.3	22.01	16.12	4SW	3.30	73.3%
3.2	22.84	16.12	4SW	3.30	72.9%
3.1	23.78	16.12	4SW	3.30	72.2%
3.0	24.72	16.15	4SW	3.31	72.1%
2.9	24.13	16.17	BOOST	3.31	76.6%
2.8	25.32	16.16	BOOST	3.31	75.5%
2.7	26.77	16.15	BOOST	3.31	73.9%
2.6	28.21	16.15	BOOST	3.31	72.9%
2.5	29.98	16.14	BOOST	3.31	71.2%
2.4	32.23	16.14	BOOST	3.31	69.0%
2.3	34.58	16.13	BOOST	3.31	67.0%

Table 6.2 *Buck-Boost Efficiency for Constant 16.2 mA Load, Input Voltage Swept*

6.4 CHAPTER 6 SUMMARY

Chapter 6 overviews the system power considerations, including the power source, proper voltage regulation, and system power consumption in terms of major system components.

- **Section 6.1** highlights the rechargeable battery used (TENERGY 3.7V Li-Po) and provides typical battery metrics, including the battery voltage and capacity.
- **Section 6.2** discusses the design of the voltage regulation circuitry used to regulate the battery voltage input to a constant 3.3V that is used to power the microcontroller and the Bluetooth module. A buck-boost switching power supply (SPS) is used so that voltages both above and below the desired 3.3V can be properly regulated.
- **Section 6.3** breaks down the system power consumption by major component, providing a look at how much current is drawn by each component in the system, along with a plot and discussion about the accelerating battery discharge rate.

CHAPTER 7: FUTURE WORK AND CONCLUSION

7.1 FUTURE WORK AND IMPROVEMENTS

The preceding chapters discuss, in detail, the design and implementation of the first revision of the compression sensing device. Many tradeoffs were considered, and difficult decisions were made in each stage of the design for various reasons and due to specific limitations. Though the device is fully functional and exhibits accurate readings and efficient operation, each different segment of the device can be improved in some way, or in many ways. This section explores each tradeoff and potential design improvement in detail.

7.1.1 MAIN MCU/BLE CIRCUIT

Many aspects of the main MCU/BLE circuit design and PCB can be improved upon, including board size and shape, data processing, and power consumption. There are several minor improvements that can be made to reduce the size of the board without any significant increase in cost, the first of which is using a smaller microcontroller. The PIC18LF26K22 has 28 pins and 19 ADC channels, only 5 of which are used in the design. Thus, a microcontroller with at least 14 less pins (50% less) could theoretically be used to properly operate and control the current system. Future versions of the device may include other sensors (such as temperature sensors to help identify the early onset of skin infection), in which case more ADC channels would be necessary. Nonetheless, a large number of pins on the current microcontroller are NC and therefore a smaller microcontroller would be plenty sufficient.

As mentioned in section 4.2, the HM-11 BLE module is a sufficient, much smaller, much lower pin count alternative for the HM-10 BLE module. The HM-11 is nearly half the size of the

HM-10 in terms of PCB area and has **less than half** the number of pins (HM-10 has 34 pins, HM-11 has only 16). In terms of operation, the HM-11 operates identically to the HM-10. The pins removed from the HM-10 to the HM-11 include a number of GPIO pins as well as NC pins, none of which are necessary for the operation of the system. The HM-11 keeps the GPIO pin used to indicate the Bluetooth device connection status, as well as the necessary UART communication pins. Thus, the HM-10 can be replaced by the HM-11 with no performance downgrade and a significant improvement in overall board size. The HM-11 and HM-10 are also similarly priced, so there is no cost drawback in the HM-11, either.

SMD PACKAGE TYPE	DIMENSIONS (INCHES)	DIMENSIONS (MM)
0201	0.024” × 0.012”	0.60 mm × 0.30 mm
0402	0.040” × 0.020”	1.00 mm × 0.50 mm
0603	0.063” × 0.031”	1.60 mm × 0.80 mm
0805	0.080” × 0.050”	2.00 mm × 1.25 mm
1206	0.126” × 0.063”	3.20 mm × 1.60 mm

Table 7.1 *Imperial SMD Package Types and Dimensions*

The overall area of the main MCU/BLE board can also be decreased by using smaller passives (resistors and capacitors) and smaller header pins for programming and wired connections to the sensor/amplifier boards. The prototype build uses 0603 capacitors, 0402 resistors, and 0.1” spaced header pins for programming and wired connections to the sensing devices. Some of the smaller-valued capacitors could be replaced with 0402 capacitors and all resistors could be replaced with 0201 resistors in order to conserve some space on the PCB. The biggest influence on size, however, is the 0.1” spaced header pins, which are unnecessarily large. A future design would use a fine-pitch, low profile connector and a ribbon cable to connect the main board signals out to the sensor/amplifier boards. Smaller-pitch programming pins can also be used to reduce board size. In production, the programming pins are a one-time use (for the uploading of the MCU program during manufacturing). Though no component is soldered to these pins, they are still an essential footprint on the board for the step of programming. All of these improvements can be made without any significant drawbacks in terms of decreased performance or increased cost.

Outside of making the main board smaller, other improvements can be made to the data processing step and the board's power consumption. The current microcontroller has an on-chip 10-bit ADC, which has a 3.22 mV LSB voltage when VDD = 3.3V. Other MCUs are available with higher-resolution ADCs, where each increase by 1 bit in resolution cuts the LSB voltage in half. If the LSB voltage is cut in half, the minimum sensible change in pressure is cut in half as well. Thus, by using a higher-resolution ADC, smaller changes in pressure can be sensed by the system. This may or may not be necessary depending on clinical trial results, considering that the 10-bit ADC is already able to detect changes in pressure less than 1 mmHg. Table 7.2 below provides a look at the different LSB voltages for high-resolution ADCs, as well as the associated LSB pressure, or minimum sensible change in pressure. One drawback is that higher-resolution ADCs are larger, so MCUs with higher-resolution ADCs are often large and would make the overall board size increase.

ADC RESOLUTION	LSB VOLTAGE	LSB PRESSURE
10-bit	3.22 mV	0.285 mmHg
11-bit	1.61 mV	0.143 mmHg
12-bit	0.81 mV	0.072 mmHg

Table 7.2 *Changes in LSB Voltage with Different ADC Resolutions, VDD = 3.3V*

In the realm of power consumption, there are several resistors on the main board which can be replaced with larger resistors to limit the current drawn and therefore the power consumed by the circuit. For one, the BLE status LED has a series resistor connected to it for current limiting purposes. This resistor, by limiting the current, also sets the brightness of the LED. Lowering the resistance increases both the power consumption of the resistor and the brightness of the LED, while increasing the resistance decreases both the power consumption of the resistor and the brightness of the LED. A battery-monitoring voltage divider is also present on the main board and simply connects the battery voltage to 2 series resistors of equal resistance to form a ½ voltage divider. The output of the divider is used to monitor the battery voltage. The size of these resistors sets the current through the divider directly, and therefore, sets the power consumption as well. If smaller resistors are used, a more accurate reading of the battery voltage will come from the divider, since the effects of mismatch will be much smaller. However, the smaller resistance results in more power consumption. If larger resistors are used, a less accurate reading will result from resistor mismatch, but the power consumption will decrease.

The final future improvement of the main MCU/BLE circuit and PCB design is centered around the use of the CC2541 microcontroller. The CC2541 is a 2.4 GHz Bluetooth Low Energy System-on-Chip MCU made by Texas Instruments [5] and is the MCU present on the HM-10 and HM-11 Bluetooth modules. Peripherals include 23 GPIO pins (2 of which can drive LEDs), 3 general-purpose timers, a watchdog timer, a battery monitor and temperature sensor, two USARTs and a 12-bit ADC with 8 input channels and configurable resolution. All in all, this MCU is essentially the entire MCU/BLE board built into one chip, aside from the voltage regulation. The use of this IC would revolutionize the entire system design, as it costs just $4.42 when in stock per unit, and only $2.76 when purchased in bulk quantities greater than or equal to 500 units. With the HM-10 and HM-11 modules costing around $10 each and the microcontroller costing another $3.38 per unit, the use of the CC2541 could reduce the cost per unit of the sensing device by nearly $10, while also dramatically shrinking the size of the main PCB. The only problem with this MCU is that currently, no major part distributors have it in stock, so none are available for purchase.

7.1.2 SENSOR/AMPLIFIER CIRCUIT AND SENSING MECHANISM

Like the main MCU/BLE circuit, the sensor/amplifier circuit also has room to improve in terms board size, as does the sensing mechanism (discussed in section 3.5 in detail). The mechanism can also be improved to help assure that the correct pressure is read by the sensor. The first major improvement to the board itself is changing the sensor from the HSFPAR303A to the much smaller HSFPAR003A. Section 3.3 discusses the differences between the two devices in detail. The key difference between the two devices is the size. Since the 003A is much smaller and lower profile, the PCB (see Fig. 3.9) can be designed much smaller and thus the sensing devices will be more comfortable for the patient. In addition, the 303A needs to be mounted to the PCB using an adhesive, as it cannot be soldered. Its signals are routed out via a flex PCB meant to be used in conjunction with a connector, but the sensor itself cannot be soldered. Its smaller alternative, the 003A, is a surface mount device (SMD) and can be soldered using reflow soldering and is therefore far more convenient for manufacturing purposes. The 003A is also about half the price of the 303A, which is obviously desirable.

Another key component that contributes to the size of the sensor/amplifier board is the instrumentation amplifier. If the improvement described in the previous paragraph is made such that the new force sensor is the smaller HSFPAR003A, the new limiting factor in how small the PCB can be designed is the MAX4208 instrumentation amplifier. Though smaller amplifiers exist, many of them have a large input offset voltage. The offset voltage adds to the input signal so that the amplifier then sees the input voltage plus the offset voltage and amplifies the two of them together, resulting in a garbage output signal. The offset voltage of the amplifier needs to be much smaller (at least 2 orders of magnitude) than the input signal level for the amplifier to be effective. As a last resort, the PCB could be designed as a two-sided board (components on two sides instead of just one) so that components can be distributed evenly on top and bottom shrinking the overall area of the board. The drawback of this technique is that components will then be directly contacting the patient’s leg which is likely to be slightly uncomfortable, and certainly less comfortable than would be the flat side of a one-sided board. Having components on two sides of the board would also add to the already-excessive thickness of the sensing mechanisms, which is undesirable.

Improvements to the sensing mechanism include reduction of the spring constant in the pogo pins, height reduction of the entire apparatus, and turret head size for applying a force to the sensor actuator. First, the spring constant of the pogo pins is not terribly high, but a lower spring constant would allow the device to measure pressure more accurately at lower pressures. As of now, the spring force of the pogo pins needs to be overcome before the device can accurately sense pressure. The measurements are still reasonable at lower pressures, but the sensor is not feeling the full applied force since the pogo pin springs are absorbing a portion of it at low pressures. Additionally, the height of the entire apparatus can very easily be reduced by simply ordering 0.6 mm thickness PCBs instead of 1.6 mm PCBs. The entire apparatus height decreases by 2 mm by making this one simple change, and cost does not increase at all. Lastly, the force sensor datasheets [8], [9] inform the user that the force should be applied by a body that is larger in surface area than the actuator for the best results. Currently, the turret being used is smaller than the actuator, which could result in poor force measurements, and in turn, poor pressure measurements. The issue is temporarily fixed in the prototype build by soldering a small copper square (whose area is larger than that of the actuator) to the end of the turret, but this is not a permanent fix. Instead, a different turret with a larger head should be used in future revisions.

7.1.3 SOFTWARE

The software component of the system is split into two main parts. The first part is the C code written in the MPLAB IDE and uploaded to the microcontroller to govern its operation, and the second part is the smartphone application. The C code and the smartphone application can both be improved upon to make the system more efficient and more powerful. First, the C code is written as a sequential program (see Appendix C). A sequential program is one which runs from start to finish in the same order each time through, and cycles back to the top and keeps running on a loop. The result is a program which is running constantly and cranking out data which is not necessarily needed so frequently. A better alternative to sequential programming is event-driven programming. Rather than running on a loop, an event-driven program waits for “events” in order to execute code segments or instructions. These “events” can be the push of a button by a user, the measurement of a value above or below some threshold, or even just the reception of a character. In order to sense “events” in sequential programs, interrupts must be used, but they are far less robust and versatile than event-driven designs. The entire system could be improved and enhanced with better functionality if redesigned in an event-driven format.

The smartphone application is also primed for improvements, including an enhanced graphical user interface (GUI), better customizability, and compatibility with iOS. The current smartphone application was designed using the MIT App Inventor, discussed in detail in Ch. 5. The MIT App Inventor is useful but does not allow much freedom in terms of interface design. Also, as of now, the application is only compatible with Android smartphones. An alternative to the MIT App Inventor is a similar web-based program called Thunkable, and applications designed in Thunkable are compatible with both Androids and iPhones, and web applications can also be designed. Another alternative is to design the application from scratch independent of any online app inventor. This method is the most difficult and code intensive but provides the most customizability and freedom with the design of the application. For the current application, the GUI is a bit dry and not much is going on. Future improvements would include the addition of a battery status bar graphic, along with a graphic of the human leg and the four sensor levels. The sensed pressure would be displayed on the graphic and the graphic would change colors at different levels based on the current applied pressure value and whether or not it is correct within some range. For example, if the pressure at the upper calf is supposed to be 10 mmHg, the application can light up the upper calf region green when the pressure is between 8 mmHg and 12 mmHg, and yellow when the pressure is outside of this range. Perhaps it can light up red if the pressure exceeds 25 mmHg, indicating an overpressure warning.

7.1.4 POWER AND BATTERY LIFE

One major downside to the battery and circuitry used to power the system (shown in Fig. 6.1) is that there is no overvoltage, overcurrent, reverse-polarity, or over-discharge protection circuitry. Many rechargeable batteries come equipped with overvoltage, overcurrent, and over-discharge protection to keep the battery safe from harmful charging voltages/currents, and from harmful discharge levels. For example, the Texas Instruments BQ24312 Overvoltage and Overcurrent Protection IC [14] is “designed to provide protection to Li-ion batteries from failures of the charging circuit. The device continuously monitors the input voltage, the input current, and the battery voltage. In case of an input overvoltage condition, the device immediately removes power from the charging circuit…” Other ICs and protection circuits like the BQ24312 exist for Lithium-Polymer batteries as well. A surefire improvement to the system would be to equip the battery with protection circuitry. While protection circuitry would also increase cost, such a cost increase may be worthwhile.

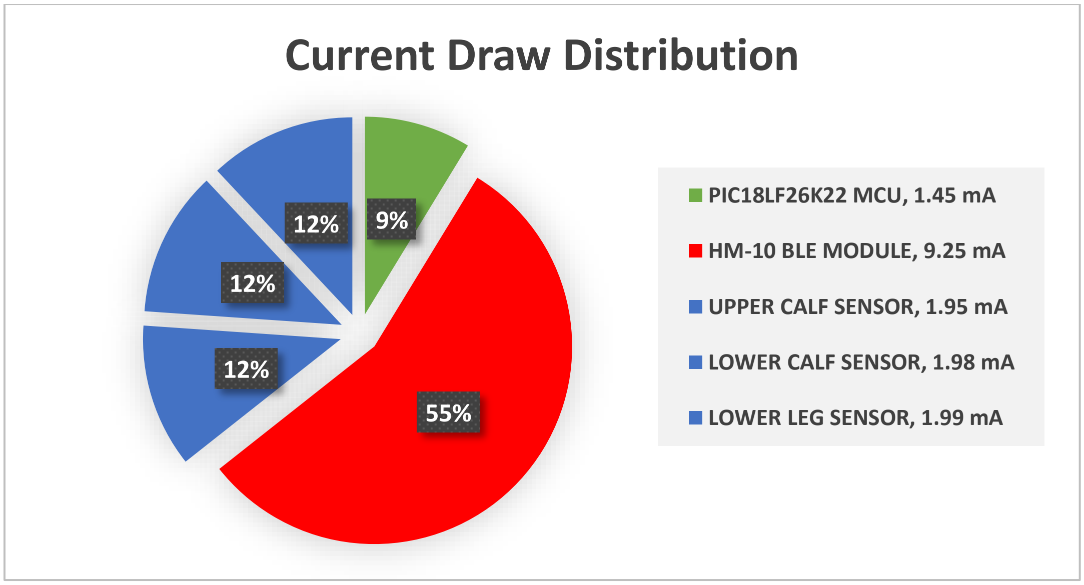


Figure 7.1 *Pie Chart of Current Draw Distribution by System Component*

Another concern is presented by the power consumption of the Bluetooth module. The HM-10 is a BLE or Bluetooth Low Energy module which nominally draws 8.5 mA [6] in active mode, whether connected or disconnected. However, when the module is connected to another Bluetooth device, the Bluetooth status LED is on, and draws an extra 0.75 mA of current, resulting in a maximum current draw of 9.25 mA, as seen in Fig. 7.1 above. With the microcontroller circuit drawing less than 1.5 mA and each of the sensor boards drawing just under 2 mA, the Bluetooth module and status LED pull **over half** (55%) of the total current for the entire system. The total current drawn by the entire system is around 16.6 mA at maximum current draw when the status LED is ON and the system is connected to a smartphone. The BLE module current draw is fixed at 8.5 mA in active mode and cannot be reduced any further. Other devices, such as the RN4870/71 and the BM70/71 BLE modules by Microchip, were explored and draw currents of between 10 –

13 mA during normal operation, which is even higher than the HM-10 module. However, other Bluetooth modules that use Bluetooth v5.0 and higher have much lower on-currents (average) are shown in Table 7.3 below. These other devices should be considered for future revisions of the device to limit current draw and increase battery life.

MODULE	MANUFACTURER	$I_{ON,AVG}$	VERSION	COST
DA14531MOD	Dialog Semiconductor	3.0 mA	Bluetooth v5.1	$4.48
BM70/71	Microchip	11.5 mA	Bluetooth v5.0	$7.26
RN4870/71	Microchip	11.5 mA	Bluetooth v5.0	$7.71
HM-10/HM-11	JNHuaMao Technology	8.5 mA	Bluetooth v4.0	$8.99
NORA-B100	U-Blox	5.1 mA	Bluetooth v5.2	$13.36
BDE-BLEM301	BDE Technology	3.0 mA	Bluetooth v5.1	$20.00

Table 7.3 *List of BLE Modules, Typical On-Currents, and Cost*

7.1.5 OTHER IMPROVEMENTS

Some other improvements include the obvious redesign and reconsideration of the garment used, along with the addition of temperature sensors for the purpose of detecting infection, and a compliance algorithm to check patient compliance with the device. The first of these improvements is the reconsideration of the garment used. The compression sensing device detailed in this paper is meant to go beneath a compression wrap and sense the pressure applied by the wrap. Thus, for the purpose of comfort and promotion of compliance, the garment should be reasonably thin and soft. The garment should also cover the lower leg and foot area, rather than just the lower leg. Though it needs to cover the foot and lower leg, any sock-like donning of the garment will be bothersome and tedious for the patient and therefore should be avoided. A Velcro-strapping method or any sort of strapping method is ideal because it allows for size adjustability for different legs. Since patients have all different sizes of legs, adjustability is desirable, and cannot be achieved by a sock. Also, the garment should not apply any compression of its own to the leg in order for the pressure readings to be accurate. For example, if the compression device is applying 40 mmHg to the leg, the device will read 40 mmHg. If the garment is then applying an extra 20 mmHg to the leg, that 20 mmHg will not be sensed. The reading on the smartphone display will be 40 mmHg, though the leg will actually be feeling 60 mmHg.

Another very important improvement that should be made to the device is the addition of temperature sensors at the force sensor levels for local temperature monitoring. According to an advanced wound care study published in the National Library of Medicine [21], "increased local temperature is a classic sign of wound infection, and its quantitative measurement has the potential to assist with assessment and diagnosis of chronic deep wound and surrounding skin infection at the bedside." Results of the study showed a "statistically significant relationship between increased skin temperature and wound infection… [demonstrating] that incorporating quantitative skin temperature measurement into routine wound assessment provides a timely and reliable method for a wound care practitioner to quantify the heat associated with deep and surrounding skin infection and to monitor ongoing wound status." The addition of temperature sensors to the current force sensor/amplifier boards would be seamless. Since the PCB is only one-sided, the addition of a small temperature sensor like those in the TMP23x family by Texas Instruments [22] would be quick and easy. Each sensor board would then have four signals routed back to the main board: VDD, ground, the force sensor output, and the temperature sensor output. The TMP23x family of temperature sensors are high-accuracy analog output temperature sensors. Figure 7.2 shows each unused ADC input channel on the current configuration of the PIC18LF26K22 MCU in the system design (red text on J1). The addition of four analog output temperature sensors would take up four more ADC input channels, leaving 6 more for the addition of extra sensors in the future.

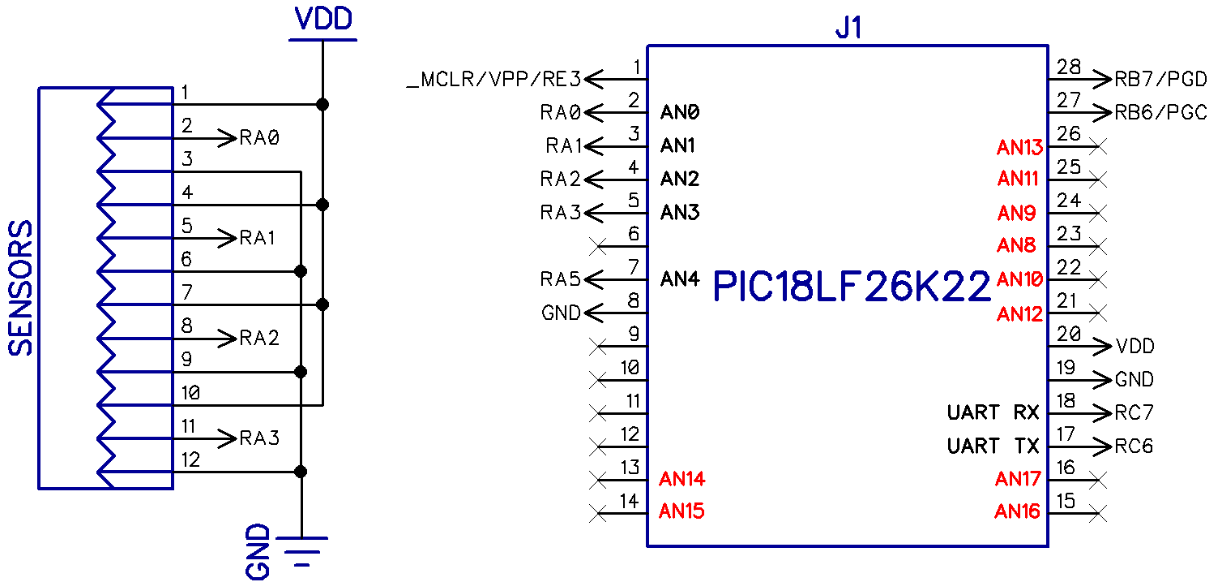


Figure 7.2 *Schematic Snippet Showing Unused ADC Input Channel Pins on MCU*

One last addition to the design is the implementation of some form of compliance monitoring algorithm. Patient compliance is a necessity if patients wish to heal properly or in a timely manner. Typically, patients will get fed up with medical devices if they are uncomfortable for the patient. While making the device as comfortable as possible is a necessity, not every patient will be pleased. One possible method for implementing a compliance algorithm would deal with the monitoring of pressure readings over time intervals when the patient is compliant and other intervals when the patient is not compliant. These patterns can be analyzed and can be used to determine based on past sensor readings whether or not a patient is compliant, and how often. For example, when a patient has the device on and it is being used, pressure values will be greater than 0 mmHg. The device can check every 20 or 30 minutes whether or not the pressure readings are greater than 0 mmHg. Each time the pressure readings are greater than, say 2 mmHg (to account for any startup offset), the patient is theoretically complying, and this would be noted. If the pressure readings are less than 2 mmHg, the patient is theoretically not complying. A running average can be taken of the number of readings where the patient is theoretically complying and not complying to form a metric for compliance in terms of percentage.

7.2 CONCLUDING REMARKS

The prototype build is successful by a wide variety of standards. It is capable of wirelessly transmitting accurate pressure readings to a user-friendly smartphone application via Bluetooth, and while on continuously, it lasts approximately 8 hours on a single charge. Pressure readings at each level never have an error greater than 5 mmHg, and the entire device costs less than $100 to build one unit during mass production (see Appendix C). Despite the success of the prototype, there are a plethora of improvements that need to be made for the device to be market-ready and patient-friendly. Improvements should be made to minimize the size of all components resulting in the minimum possible PCB sizes. The smartphone application should be made more aesthetically pleasing for the user. The battery life can be extended by using a lower-power BLE module and/or a higher-capacity battery. Deeper analysis of the tradeoffs associated with each improvement in terms of size, patient comfort, and cost should be considered in the design of a second revision of the system. The certain next steps and essential improvements for the design of a second revision include:

- The replacement of the HM-10 with the HM-11 Bluetooth module
- The replacement of the HSFPAR303A with the HSFPAR003A force sensor
- The replacement of the buck-boost regulator with an efficient linear regulator
- The redesign of both PCBs for area and thickness minimization
- The inclusion of temperature sensors for the detection of skin infection

- The redesign of the smartphone application for both Android and iOS compatibility
- The redesign of a garment which covers the foot and is much thinner than the calf sleeve
- The use of ribbon cable and low-profile connectors from the main board to sensor boards

With each of these improvements made, the second revision of the device will be taken to clinical trials at UMC for testing on real patients. These clinical trials will be used to collect data on the healing time of VSUs for patients with and without the device beneath their compression wraps. If the device shows promising results in terms of healing time reduction and patient compliance in the clinical trial stage, a final revision will be designed, and the device will be ready to bring to market.

APPENDIX A: SCHEMATICS

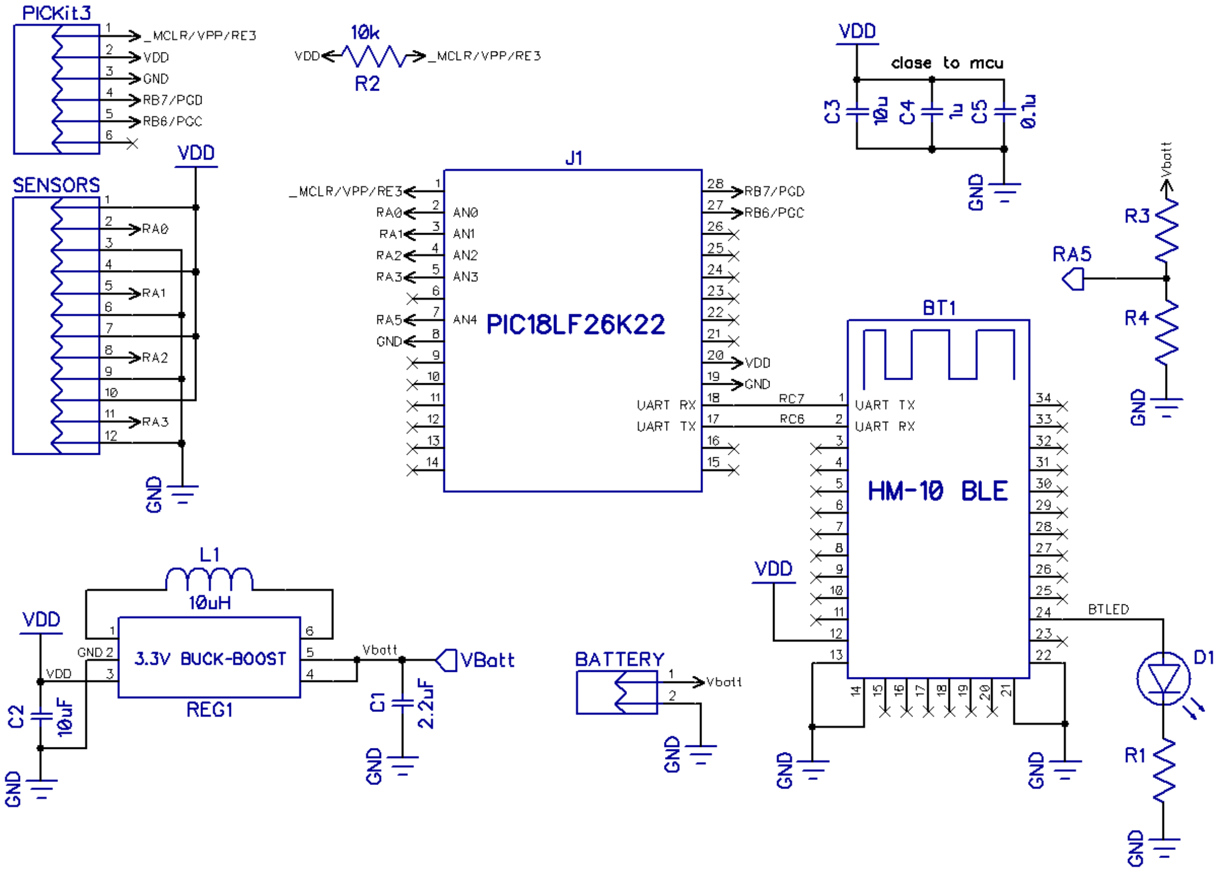


Figure A.1 *Main MCU/BLE PCB Schematic*

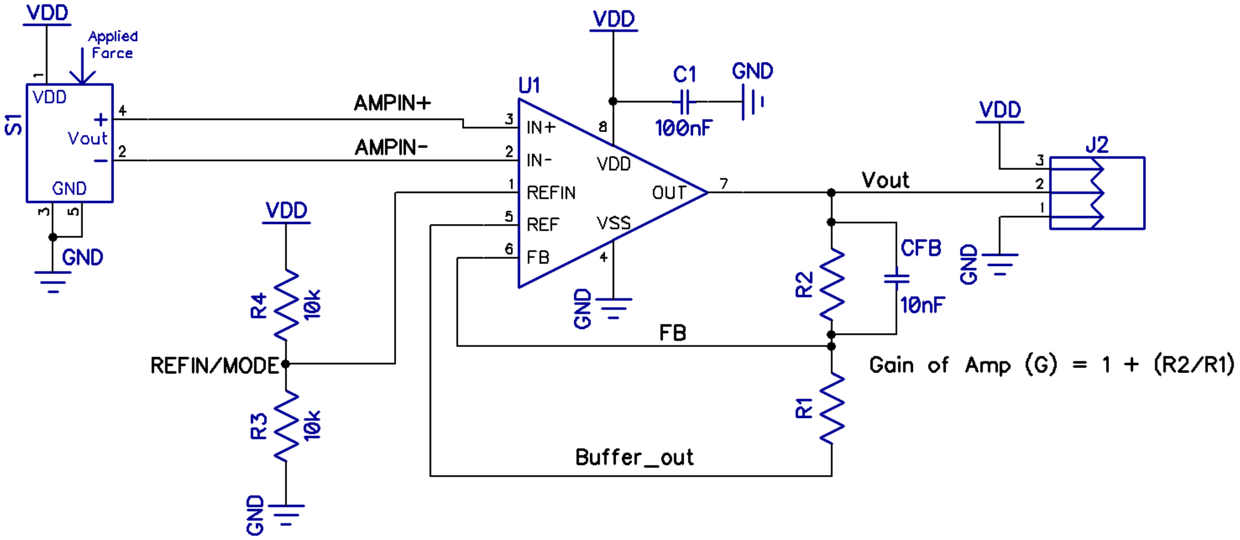


Figure A.2 *Force Sensor/Amplifier PCB Schematic*

APPENDIX B: SINGLE UNIT COST BREAKDOWN

PART DESCRIPTION	UNIT PRICE	QUANTITY	TOTAL
MAIN MCU/BLE PCB	$5.00	1	$5.00
HM-10 BLUETOOTH LOW ENERGY MODULE	$8.99	1	$8.99
BLUETOOTH STATUS LED 0805 (BLUE)	$0.18	1	$0.18
LTC3531-3.3 BUCK-BOOST REGULATOR IC	$5.39	1	$5.39
LFT4022T-100M-D 10µH INDUCTOR	$1.73	1	$1.73
PIC18LF26K22 MICROCONTROLLER	$3.44	1	$3.44
PASSIVES: CAPACITOR 0603, VALUES VARY	$0.22	7	$1.54
PASSIVES: RESISTOR 0402, VALUES VARY	$0.13	8	$1.04
TENERGY 3.7V 380MAH LIPO BATTERY	$6.99	1	$6.99
2 X 100 MIL RIGHT ANGLE MALE HEADER	$0.03	1	$0.03
COPPER WIRE, RED, STRANDED	$0.30	3	$0.90
COPPER WIRE, BLACK, STRANDED	$0.30	3	$0.90
COPPER WIRE, YELLOW, STRANDED	$0.30	3	$0.90
SENSOR/AMPLIFIER PCB	$5.00	3	$15.00
SENSOR MECHANISM HOUSING PCB	$5.00	3	$15.00
TERM TURRET SINGLE L=1.79MM	$0.72	3	$2.16
HSFPAR303A FORCE SENSOR	$11.60	3	$34.80
MAX4208 INSTRUMENTATION AMPLIFIER	$4.16	3	$12.48
GOLD-PLATED COPPER POGO PIN CONN.	$0.13	9	$1.17
VIVE LOWER LEG COMPRESSION WRAP	$13.99	1	$13.99
		TOTAL COST:	**$131.63**

Table B.1 *Cost Breakdown by Part for One Single Unit*

APPENDIX C: MASS PRODUCTION (1000+) UNIT COST BREAKDOWN

PART DESCRIPTION	UNIT PRICE	QUANTITY	TOTAL
MAIN MCU/BLE PCB	$0.75	1	$0.75
HM-10 BLUETOOTH LOW ENERGY MODULE	$8.99	1	$8.99
BLUETOOTH STATUS LED 0805 (BLUE)	$0.12	1	$0.12
LTC3531-3.3 BUCK-BOOST REGULATOR IC	$3.38	1	$3.38
LFT4022T-100M-D 10µH INDUCTOR	$0.79	1	$0.79
PIC18LF26K22 MICROCONTROLLER	$2.85	1	$2.85
PASSIVES: CAPACITOR 0603, VALUES VARY	$0.05	7	$0.35
PASSIVES: RESISTOR 0402, VALUES VARY	$0.02	8	$0.16
TENERGY 3.7V 380MAH LIPO BATTERY	$3.50	1	$3.50
2 X 100 MIL RIGHT ANGLE MALE HEADER	$0.03	1	$0.03
COPPER WIRE, RED, STRANDED	$0.30	3	$0.90
COPPER WIRE, BLACK, STRANDED	$0.30	3	$0.90
COPPER WIRE, YELLOW, STRANDED	$0.30	3	$0.90
SENSOR/AMPLIFIER PCB	$0.32	3	$0.96
SENSOR MECHANISM HOUSING PCB	$0.34	3	$1.02
TERM TURRET SINGLE L=1.79MM	$0.40	3	$1.20
HSFPAR303A FORCE SENSOR	$11.60	3	$34.80
MAX4208 INSTRUMENTATION AMPLIFIER	$2.36	3	$7.08
GOLD-PLATED COPPER POGO PIN CONN.	$0.13	9	$1.17
VIVE LOWER LEG COMPRESSION WRAP	$13.99	1	$13.99
		TOTAL COST:	**$83.84**

Table C.1 *Cost Breakdown by Part for One Single Unit in Mass Production*

APPENDIX D: MICROCONTROLLER (C LANGUAGE) SOURCE CODE

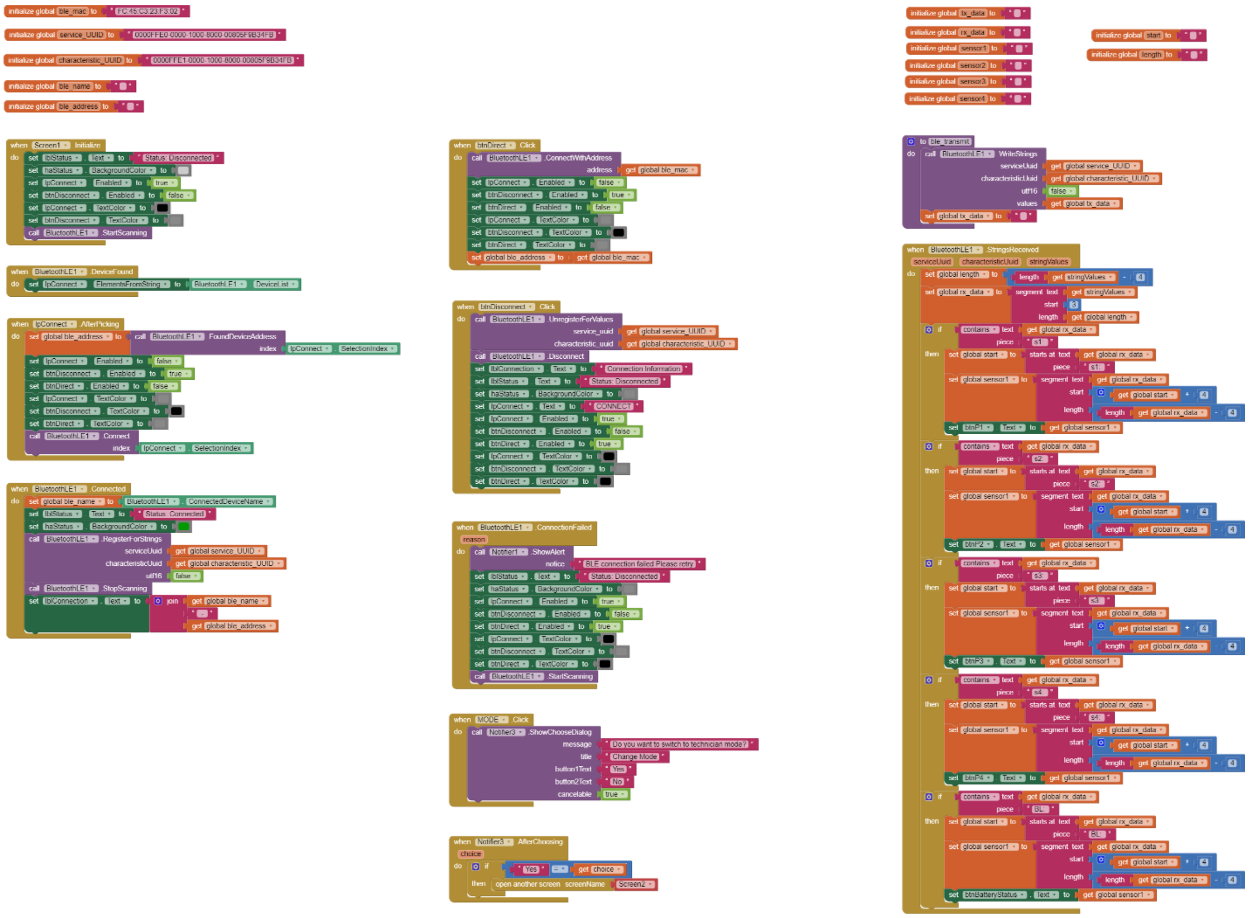
```c
/*
 *
 * Last Updated: 08/29/2021
 * Updated By: James Skelly
 *
 */

#include <xc.h>
#include "config_bits_PIC16LF26K22.h"
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

#define _XTAL_FREQ  4000000              // set delay clk, system clk = 4 MHz
#define BAUDRATE    9600                 // set baudrate to 9600
#define VDD         3.3                  // declare VDD to be 3.3V
#define VCM         1.65                 // declare VCM to be 1.65V
#define WHITE_LED   PORTBbits.RB6        // define programmer clk on RB6
#define BLUE_LED    PORTBbits.RB7        // define programmer data on RB7


///////////////////////////////////////////////////////////////////////////////
/////////////////////// ***** Function prototypes ***** ///////////////////////
///////////////////////////////////////////////////////////////////////////////

void putch(char);                        // Used to enable printf function

char UART_RxChar(void);                  // Performs reception of a character
void UARTInit(const long int);           // Initializes UART (rs232)
void INTERRUPTInit(void);                // Initializes/Enables Interrupts
void CompleteTask(char);                 // Completes task based on character
                                         // received by MCU

void ADCInit(void);                      // Initializes/configures the ADC

float GetADCResult(int);
float VoltageToForce(float, float);
float ForceToPressure(float, float);


///////////////////////////////////////////////////////////////////////////////
///////////////////////// ***** Main Code Body ***** //////////////////////////
///////////////////////////////////////////////////////////////////////////////

void main(void)
{
   ///// ***** Variable Declarations ***** /////

   char sbuf1[40], a[]="s1: "; // label to identify pressure from sensor 1
   char sbuf2[40], b[]="s2: "; // label to identify pressure from sensor 2
   char sbuf3[40], c[]="s3: "; // label to identify pressure from sensor 3
   char sbuf4[40], d[]="s4: "; // label to identify pressure from sensor 4

   char sbuf5[40], e[]="f1: "; // label to identify force value from sensor 1
char sbuf6[40], f[]="f2: "; // label to identify force value from sensor 2
char sbuf7[40], g[]="f3: "; // label to identify force value from sensor 3
char sbuf8[40], h[]="f4: "; // label to identify force value from sensor 4

char sbuf9[40], i[]="v1: ";  // label to identify voltage from sensor 1
char sbuf10[40], j[]="v2: "; // label to identify voltage from sensor 2
char sbuf11[40], k[]="v3: "; // label to identify voltage from sensor 3
char sbuf12[40], l[]="v4: "; // label to identify voltage from sensor 4
char sbuf13[40], m[]="BL: "; // label to identify battery life
char ch;                     // character received by MCU UART Rx pin

int AN = 0;
int cycleCounter = 0;
_Bool isFirstCycle = 1;

int currentChannel = 0;
int resultOfConversion[5] = {0,0,0,0,0};
float resultingVoltage[5] = {0.0,0.0,0.0,0.0,0.0};
float appliedForce[4] = {0.0,0.0,0.0,0.0};
float appliedPressure[4] = {0.0,0.0,0.0,0.0};
float BIAS[4] = {0.0,0.0,0.0,0.0};

const float AREA_PCB = 346.36;       // units of mm squared, actual PCB area
const float AREA_CALC = 665.35;      // units of mm squared, calculated area
const float AREA_FINAL = 594.06;     // units of mm squared, edited area

///// ***** Initialization ***** /////

OSCCON = 0x56;       // set internal oscillator to 4MHz
UARTInit(BAUDRATE);  // initialize the UART
INTERRUPTInit();     // initialize the interrupt
ADCInit();           // initialize the ADC
TRISA = 0xFF;        // set all port A pins as inputs to enable ADC channels
TRISB = 0x00;        // set all port B pins as outputs to enable LEDs
PORTB = 0x00;        // initially set all port B pins low


///// ***** Continuous Loop ***** /////

while (1)
{
    cycleCounter++;

    // Obtain Sensor Values
    for (int AN=0; AN<4; AN++)
    {
        if(isFirstCycle == 1)
        {
           BIAS[AN] = GetADCResult(AN);
        }
        else
        {
           resultingVoltage[AN] = GetADCResult(AN);
           appliedForce[AN] = VoltageToForce(resultingVoltage[AN], BIAS[AN]);
           appliedPressure[AN] = (ForceToPressure(appliedForce[AN], AREA_FINAL));
        }

    }

    // Obtain battery voltage
    int AN=4;
    resultingVoltage[AN] = (GetADCResult(AN))*2;
```

```
       // create strings with data to send to app
       sprintf(sbuf1, "%s%.3f", a, appliedPressure[0]);
       sprintf(sbuf2, "%s%.3f", b, appliedPressure[1]);
       sprintf(sbuf3, "%s%.3f", c, appliedPressure[2]);
       sprintf(sbuf4, "%s%.3f", d, appliedPressure[3]);

       sprintf(sbuf5, "%s%.3f", e, appliedForce[0]);
       sprintf(sbuf6, "%s%.3f", f, appliedForce[1]);
       sprintf(sbuf7, "%s%.3f", g, appliedForce[2]);
       sprintf(sbuf8, "%s%.3f", h, appliedForce[3]);

       sprintf(sbuf9, "%s%.3f", i, resultingVoltage[0]);
       sprintf(sbuf10, "%s%.3f", j, resultingVoltage[1]);
       sprintf(sbuf11, "%s%.3f", k, resultingVoltage[2]);
       sprintf(sbuf12, "%s%.3f", l, resultingVoltage[3]);
       sprintf(sbuf13, "%s%.3f", m, resultingVoltage[4]);

       // send pressures to app
       printf("%s\n", sbuf1);
       __delay_ms(20);
       printf("%s\n", sbuf2);
       __delay_ms(20);
       printf("%s\n", sbuf3);
       __delay_ms(20);
       printf("%s\n", sbuf4);
       __delay_ms(20);

       // send forces to app
       printf("%s\n", sbuf5);
       __delay_ms(20);
       printf("%s\n", sbuf6);
       __delay_ms(20);
       printf("%s\n", sbuf7);
       __delay_ms(20);
       printf("%s\n", sbuf8);
       __delay_ms(20);

       // send voltages to app
       printf("%s\n", sbuf9);
       __delay_ms(20);
       printf("%s\n", sbuf10);
       __delay_ms(20);
       printf("%s\n", sbuf11);
       __delay_ms(20);
       printf("%s\n", sbuf12);
       __delay_ms(20);

       // send battery voltage to app
       printf("%s\n", sbuf13);
       __delay_ms(20);

       if (cycleCounter >= 1)
       {
           isFirstCycle = 0;
       }
   }

    return;
}
```

```
// Functions
////////////////////////////////////////////////////////////////////////////////

// function below writes characters to USART using printf
   void putch(char data)
{
    while(!TX1IF)
        continue;
    TXREG1 = data;
}

// function to receive char
char UART_RxChar()
{
    while(RCIF == 0);
    RCIF = 0;
    return RCREG1;
}

// function to complete task based on input char
void CompleteTask(char ch)
{
    switch(ch)
    {
     case '0':
     {
         BLUE_LED = 0;
         printf(" --> BLUE LED TURNED OFF\n");
         break;
     }
     case '1':
     {
         BLUE_LED = 1;
         printf(" --> BLUE LED TURNED ON\n");
         break;
     }
     case '2':
     {
         WHITE_LED = 0;
         printf(" --> WHITE LED TURNED OFF\n");
         break;
     }
     case '3':
     {
         WHITE_LED = 1;
         printf(" --> WHITE LED TURNED ON\n");
         break;
     }
     default:
     {
         printf(" --> INVALID COMMAND\n");
     }
    }
}

   float GetADCResult(int analogChannelNum)
   {
       float Vresult = 0.0;
       int ADCresult = 0;

       switch(analogChannelNum)
       {
           case 0:
               ADCON0 = 0b00000001;
               break;
           case 1:
               ADCON0 = 0b00000101;
               break;
           case 2:
               ADCON0 = 0b00001001;
               break;
           case 3:
               ADCON0 = 0b00001101;
               break;
           case 4:
               ADCON0 = 0b00010001;
               break;
           default:
               printf("ADC Channel Not Connected");
               break;
       }
       __delay_ms(20);

       ADCON0 = ADCON0 | 0b00000010;             // start ADC conversion cycle
       __delay_ms(10);

       ADCresult = ((ADRESH<<8)+ADRESL);         // pull ADC result from ADreg

       Vresult = ((double)ADCresult/1023)*VDD;   // calculate voltage from ADC

       if (analogChannelNum == 4)
        {
           ADCON0 = 0b00000001;
        }

       return Vresult;
   }

   void ADCInit(void)
   {
     // ADC control registers, page 304-306 in datasheet
     // RA0: AN0, 00000
     // RA1: AN1, 00001
     // RA2: AN2, 00010
     // RA3: AN3, 00011
     // RA5: AN4, 00100
     ADCON0 = 0b00000001; // set ADC channel input RA0(AN0), enable ADC(bit0)
     ADCON1 = 0b00000011; // connect Voltage Reference to analog VDD and VSS
     ADCON2 = 0b10001011; // Right justify result, set A/D acq. time, conv. clk
   }

/* The function below calculates the force being applied based on a voltage
       reading from the the ADC coming from the sensor. The (v) variable is
       the current voltage reading and the (b) variable is the biased voltage
       reading set at the beginning when the program starts running and the
       counter is at 0. */
float VoltageToForce(float voltage, float bias)
{
    float force = (voltage - bias)/0.2442; //(1.221 for gain of 100)
    return force;
}

float ForceToPressure(float force, float area)
{
    float conversionFactor = 7500.62;   // convert N/mm2 to mmHg
    float pressure = (force/area)*conversionFactor;
    return pressure;
}

void UARTInit(const long int baudrate)
{
   float x = ((_XTAL_FREQ/16)/BAUDRATE)-1; // baudrate formula
   int sp = x;                             // SPBRG1 integer value

   TRISCbits.RC6 = 1;    // TX pin set as output
   TRISCbits.RC7 = 1;    // RX pin set as input

   SPBRG1 = x;           // baudrate generator value
   TXSTA1bits.BRGH = 1;  // high baud rate select bit

   TXSTA1bits.SYNC = 0;  // setting for Asynchronous
   RCSTA1bits.SPEN = 1;  // enable serial pins

   TXSTA1bits.TXEN = 1;  // enable transmission
   RCSTA1bits.CREN = 1;  // enable receiver

   ANSELCbits.ANSC6 = 0; // disable ADC channel functionality of RC6
   ANSELCbits.ANSC7 = 0; // disable ADC channel functionality of RC7
   __delay_ms(2000);
}

void INTERRUPTInit(void)
{
    RCONbits.IPEN = 1;      // enable interrupt priority specification
    INTCONbits.GIEH = 1;    // enable all high priority interrupts

    PIE1bits.RC1IE = 1;     // enable interrupt for UART Rx 1
    IPR1bits.RC1IP = 1;     // set as high priority interrupt
    __delay_ms(200);
}

void __interrupt(high_priority) TechMode(void)
{
    char ch = UART_RxChar();    // wait for a character
    CompleteTask(ch);           // complete task according to button pressed

    return;
}
```

Figure D.1 *MCU C Code for Data Reception, Processing, Transmission*

APPENDIX E: MIT APP INVENTOR BLOCKS VIEW



```
initialize global ble_mac to "FC:45:C3:23:F3:02"
initialize global service_UUID to "0000FFE0-0000-1000-8000-00805F9B34FB"
initialize global characteristic_UUID to "0000FFE1-0000-1000-8000-00805F9B34FB"
initialize global ble_name to ""
initialize global ble_address to ""

when Screen1.Initialize
do  set lblStatus.Text to "Status: Disconnected"
    set haStatus.BackgroundColor to
    set lpConnect.Enabled to true
    set btnDisconnect.Enabled to false
    set lpConnect.TextColor to
    set btnDisconnect.TextColor to
    call BluetoothLE1.StartScanning

when BluetoothLE1.DeviceFound
do  set lpConnect.ElementsFromString to BluetoothLE1.DeviceList

when lpConnect.AfterPicking
do  set global ble_address to call BluetoothLE1.FoundDeviceAddress
                                  index lpConnect.SelectionIndex
    set lpConnect.Enabled to false
    set btnDisconnect.Enabled to true
    set btnDirect.Enabled to false
    set lpConnect.TextColor to
    set btnDisconnect.TextColor to
    set btnDirect.TextColor to
    call BluetoothLE1.Connect
         index lpConnect.SelectionIndex

when BluetoothLE1.Connected
do  set global ble_name to BluetoothLE1.ConnectedDeviceName
    set lblStatus.Text to "Status: Connected"
    set haStatus.BackgroundColor to
    call BluetoothLE1.RegisterForStrings
         serviceUuid get global service_UUID
         characteristicUuid get global characteristic_UUID
         utf16 false
    call BluetoothLE1.StopScanning
    set lblConnection.Text to join get global ble_name
                                   "-"
                                   get global ble_address

when btnDirect.Click
do  call BluetoothLE1.ConnectWithAddress
         address get global ble_mac
    set lpConnect.Enabled to false
    set btnDisconnect.Enabled to true
    set btnDirect.Enabled to false
    set lpConnect.TextColor to
    set btnDisconnect.TextColor to
    set btnDirect.TextColor to
    set global ble_address to get global ble_mac

when btnDisconnect.Click
do  call BluetoothLE1.UnregisterForValues
         service_uuid get global service_UUID
         characteristic_uuid get global characteristic_UUID
    call BluetoothLE1.Disconnect
    set lblConnection.Text to "Connection Information"
    set lblStatus.Text to "Status: Disconnected"
    set haStatus.BackgroundColor to
    set lpConnect.Text to "CONNECT"
    set lpConnect.Enabled to true
    set btnDisconnect.Enabled to false
    set btnDirect.Enabled to true
    set lpConnect.TextColor to
    set btnDisconnect.TextColor to
    set btnDirect.TextColor to

when BluetoothLE1.ConnectionFailed
  reason
do  call Notifier1.ShowAlert
         notice "BLE connection failed Please retry"
    set lblStatus.Text to "Status: Disconnected"
    set haStatus.BackgroundColor to
    set lpConnect.Enabled to true
    set btnDisconnect.Enabled to false
    set btnDirect.Enabled to true
    set lpConnect.TextColor to
    set btnDisconnect.TextColor to
    set btnDirect.TextColor to
    call BluetoothLE1.StartScanning

when MODE.Click
do  call Notifier3.ShowChooseDialog
         message "Do you want to switch to technician mode?"
         title "Change Mode"
         button1Text "Yes"
         button2Text "No"
         cancelable true

when Notifier3.AfterChoosing
  choice
do  if "Yes" = get choice
    then open another screen screenName "Screen2"

initialize global tx_data to ""
initialize global rx_data to ""
initialize global sensor1 to ""
initialize global sensor2 to ""
initialize global sensor3 to ""
initialize global sensor4 to ""

initialize global start to ""
initialize global length to ""

to ble_transmit
do  call BluetoothLE1.WriteStrings
         serviceUuid get global service_UUID
         characteristicUuid get global characteristic_UUID
         utf16 false
         values get global tx_data
    set global tx_data to ""

when BluetoothLE1.StringsReceived
  serviceUuid characteristicUuid stringValues
do  set global length to length get stringValues - 4
    set global rx_data to segment text get stringValues
                                  start 3
                                  length get global length
    if contains text get global rx_data
                piece "s1:"
    then set global start to starts at text get global rx_data
                                       piece "s1:"
         set global sensor1 to segment text get global rx_data
                                       start get global start + 4
                                       length length get global rx_data - 4
         set btnP1.Text to get global sensor1
    if contains text get global rx_data
                piece "s2:"
    then set global start to starts at text get global rx_data
                                       piece "s2:"
         set global sensor1 to segment text get global rx_data
                                       start get global start + 4
                                       length length get global rx_data - 4
         set btnP2.Text to get global sensor1
    if contains text get global rx_data
                piece "s3:"
    then set global start to starts at text get global rx_data
                                       piece "s3:"
         set global sensor1 to segment text get global rx_data
                                       start get global start + 4
                                       length length get global rx_data - 4
         set btnP3.Text to get global sensor1
    if contains text get global rx_data
                piece "s4:"
    then set global start to starts at text get global rx_data
                                       piece "s4:"
         set global sensor1 to segment text get global rx_data
                                       start get global start + 4
                                       length length get global rx_data - 4
         set btnP4.Text to get global sensor1
    if contains text get global rx_data
                piece "BL:"
    then set global start to starts at text get global rx_data
                                       piece "BL:"
         set global sensor1 to segment text get global rx_data
                                       start get global start + 4
                                       length length get global rx_data - 4
         set btnBatteryStatus.Text to get global sensor1
```

Figure E.1 *Blocks View for Home Screen Design, MIT App Inventor*

REFERENCES

[1] D. Joseph, OD. “Chronic Venous Insufficiency (CVI).” My.clevelandclinic.org. https://my.clevelandclinic.org/health/diseases/16872-chronic-venous-insufficiency-cvi (accessed Oct. 1, 2021).

[2] D. Jaliman, MD. “Understanding Varicose Veins – the Basics.” Webmd.com. https://www.webmd.com/skin-problems-and-treatments/understanding-varicose-veins-basics (accessed Oct. 1, 2021).

[3] K. Holland. “Obesity Facts.” Healthline.com. https://www.healthline.com/health/obesity-facts (accessed Oct. 1, 2021).

[4] J. Menezes, MD. (2021). Optimization of Compression Therapy – Smart Stockings [PowerPoint slides].

[5] Texas Instruments, “2.4-GHz Bluetooth low energy and Proprietary System-on-Chip,” CC2541 datasheet, Jan. 2012 [Revised June 2013].

[6] JNHuaMao Technology Company, “Bluetooth 4.0 BLE module,” HM-10 datasheet [Revised Aug. 2013].

[7] Linear Technology, “LTC3531 200mA Buck-Boost Synchronous DC/DC Converters,” LTC3531-3.3 datasheet [Revised 2006].

[8] Alps Alpine, “Force Sensor HSFPAR003A Data sheet,” HSFPAR003A datasheet [Revised Mar. 2019].

[9] Alps Alpine, “Force Sensor HSFPAR303A Data sheet,” HSFPAR303A datasheet [Revised Apr. 2019].

[10] TDK, “Inductors for power circuits,” LTF4022-D datasheet [Revised Sept. 2018].

[11] Maxim Integrated, “Ultra-Low Offset/Drift, Precision Instrumentation Amplifiers with REF Buffer,” MAX4208 datasheet, May 2015 [Revised Sept. 2019].

[12] Microchip, “28/40/44-Pin, Low-Power, High-Performance Microcontrollers with XLP Technology,” PIC18LF26K22 datasheet [Revised 2012].

[13] Branch Education. How does Bluetooth Work? (May 19, 2021). Accessed: Sep. 24, 2021. [Online video]. Available: https://www.youtube.com/watch?v=1I1vxu5qIUM&t=580s

[14] Texas Instruments, “bq24312 Overvoltage and Overcurrent Protection IC and Li+ Charger Front-End Protection IC,” BQ24312 datasheet, Aug. 2009 [Revised Aug. 2015].

[15] Wikipedia, “Android version history,” (Oct. 4, 2021). Accessed: Oct. 7, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Android_version_history [Revised Oct. 4, 2021].

[16] Microchip, “Bluetooth Low Energy (BLE) Module,” BM70/71 datasheet, 2017 [Revised Aug. 2018].

[17] Microchip, “Bluetooth Low Energy Module,” RN4870/71 datasheet, 2016 [Revised Feb. 2020].

[18] Dialog Semiconductor, “SmartBond TINY Module,” DA14531MOD datasheet [Revised Aug. 2020].

[19] BDE, “Bluetooth 5.1 LE Module,” BDE-BLEM301 datasheet, July 2020 [Revised Apr. 2021].

[20] U-Blox, “Stand-alone dual-core Bluetooth 5.2 Low Energy and IEEE 802.15.4 module,” NORA-B1 series datasheet, Sep. 2020 [Revised Aug. 2021].

[21] M. Fierheller, R. G. Sibbald, “A clinical investigation into the relationship between increased periwound skin temperature and local wound infection in patients with chronic

leg ulcers." Pubmed.ncbi.nlm.nih.gov. https://pubmed.ncbi.nlm.nih.gov/20631603/ (accessed Oct. 9, 2021).

[22] Texas Instruments, "TMP23x-Q1 Automotive Grade, High-Accuracy Analog Output Temperature Sensors," TMP235AQDBZRQ1 datasheet, Apr. 2019 [Revised Nov. 2019].

[23] SunPower Electronics, "Efficiency," (2019). Accessed: Oct 8, 2021. [Online]. Available: https://www.sunpower-uk.com/glossary/what-is-power-supply-efficiency/.

[24] Bluetooth and USB 3, "Testing for the Bluetooth+USB3 problem," (2014). Accessed: Oct 9, 2021. [Online]. Available: https://www.bluetoothandusb3.com/checking-bluetooth-signal-strength.

[25] M. Thomas. "What Does DBm Mean? How Do I Know If My Signal Strength Is Optimal?" securitycoverage.com. https://www.securitycoverage.com/articles/faq/dbm-mean-know-signal-strength-optimal/ (accessed Oct. 9, 2021).

[26] T. Strassburger. "What is a Good Cell Phone Signal Strength?" accu-tech.com. https://www.accu-tech.com/accu-insider/what-is-a-good-cell-phone-signal-strength (accessed Oct. 9, 2021).

[27] Microchip, "MPLAB XC8 C Compiler User's Guide for PIC MCU," MPLAB XC8 User Guide, March 2018 [Revised: Feb. 2020].

CURRICULUM VITAE

James W. Skelly

James.W.Skelly@gmail.com http://cmosedu.com/jbaker/students/james_s/james_s.htm

Education

University of Nevada, Las Vegas, 2020-2021

Master of Science in Electrical and Computer Engineering

Graduating: December 2021

University of Nevada, Las Vegas, 2016-2020

Bachelor of Science in Electrical Engineering

Honors: Magna Cum Laude (GPA: 3.96/4.00)

Research

- **Graduate Research Assistant in an integrated circuit design/testing research group supervised by Dr. R. Jacob Baker at UNLV.**
    - **Publication 1:** Vikas Vinayaka, Sachin P. Namboodiri, Shadden Abdalla, Bryan Kerstetter, Francisco Mata-Carlos, Daniel Senda, **James Skelly,** Angsuman Roy, R. Jacob Baker. 2019. Monolithic 8x8 SiPM with 4-bit Current-Mode Flash ADC with Tunable Dynamic Range. In GLSVLSI '19: 2019 Great Lakes Symposium on VLSI, May 9-11, 2019, Tysons Corner, VA, USA. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3299874.3318005
    - **Publication 2:** S. P. Namboodiri, G. Arteaga, **J. Skelly**, F. Mata-carlos, A. Roy and R. J. Baker, "A Current-Mode Photon Counting Circuit for Long- Range LiDAR Applications," *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2020, pp. 146-149, doi: 10.1109/MWSCAS48704.2020.9184584.
    - **IC design, layout, tape out** using C5, AMS, TowerJazz processes in Cadence.
    - **PCB design** for ICs designed in the lab and other lab projects.
    - **Soldering** through-hole, SMD components by hand, as well as reflow soldering.
    - **Programming microcontrollers** for various embedded systems projects.

Other Work Experience

- **Electrical Engineering Intern** at Vorpal Research Systems, a laser and electro-optical system design and manufacturing company. (Spring 2019 – Fall 2020)
- **Electrical Engineering Intern** at Pololu Robotics and Electronics, design and test voltage regulators, motor drivers, controllers. (Fall 2021)
- **Intellectual Property Technical Consultant**
    - **Covington & Burling LLP (Palo Alto, CA and Washington, DC)**
        - Case 1 – Phenix (sic) Longhorn, LLC v. *Texas Instruments, Inc.*
            - Case Number - Texas, ED (Marshall) 2:18-cv-00020. Complaint filed on January 22, 2018.
            - Case Subject Matter – Circuit with non-volatile memory for gamma correction in a display screen.
            - Work Performed – Provided expert consulting services including reviewing schematics and other case materials.
        - Case 2 – Bell Semiconductor, LLC v. *Texas Instruments, Inc.*
            - Case Number – Texas, ED 2:20-cv-00048
            - Case Subject Matter – Package drawing files using cutouts to reduce parasitic capacitance on high-speed pins.
            - Work Performed – Reviewed package drawing files and categorized package designs.
    - **DLA Piper (East Palo Alto, CA)**
        - Case – Invensas Corporation and Tessera Advanced Technologies, Inc. v. *NVIDIA Corporation*
            - Case Number – Delaware, 1:19-cv-00861. Complaint filed on May 8, 2019.
            - Case Subject Matter – Reference voltage circuits (programmable bandgaps) having a substantially zero temperature coefficient using bipolar and MOS transistors.
            - Work Performed – Provided expert consulting services including reviewing schematics and other case materials.
- **Grader** for various electrical and computer engineering courses (Spring 2020 – Spring 2021)
- **Math Tutor –** tutored 6 high school and undergraduate level students in a variety of mathematics courses including (high school) algebra I, algebra II, geometry, (college) pre-calculus, calculus I, II, III. (Fall 2016 – Spring 2019)
- **Textbook Reviewer** for *CMOS Circuit Design, Layout, and Simulation, Fourth Edition* – R. Jacob Baker.

Projects

Individual

- **Bluetooth Low Energy Module Breakout Board:** Designed a breakout board for the HM-10, HM-11 BLE modules with on-board buck-boost SPS. PIC18LF26K22 MCU is used to send data serially to the BLE module and to configure settings on the module. MCU programmed in C using MPLAB. System can be connected to Android apps.
- **Force Sensing Mechanism with Amplified Output:** Designed a PCB containing a small force sensor with analog output voltage and an instrumentation amplifier. Entire unit is comprised of two PCBs connected by pogo pins for spring action.
- **Manually Operable Scoreboard:** Designed a 9” by 15” fully functional scoreboard for various sports using an ATmega328P MCU programmed in C.
- **Darkness Sensor:** Designed, programmed, and built PCB containing ATMEGA328P MCU and a photoresistor divider to sense when the undergraduate lab is dark and hit the switch turning the lights back on using DC push-type solenoids. MCU programmed in C.
- **PIC Microcontroller Breakout Board:** Designed a breakout board for the QFP44 PIC18LF46K22 microcontroller including convenient PICkit3 programming pins, female header ports for each IO pin, indicator LEDs for programming and power, and a UART port breaking out the TX and RX pins.
- **CMOS Boost Switching Power Supply:** Designed, simulated, and laid out a Boost SPS IC for varying temperature (0 to 100 degrees Celsius) and power supply voltage ($3.75V \leq VDD \leq 4.75V$) with a fixed 5V DC output reference voltage.
- **555 Timer Christmas Tree Ornament:** Designed a PCB to be used as an ornament in the shape of a Christmas tree using a 555 timer and powered by a 9V battery. The ornament has flashing and solid modes, and the flashing frequency can be adjusted by the on-board easily accessible potentiometer. No programming necessary.
- **CMOS High-Speed Transimpedance Amplifier:** Designed and simulated a transimpedance amplifier using differential amplifiers to convert light from an avalanche photodiode into a voltage output.
- **CMOS Low Voltage, High Gain Op-Amp:** Designed and simulated an op-amp with Gain Bandwidth Product over 1 MHz, capable of operating over a wide power supply range ($2V \leq VDD \leq 6V$).
- **CMOS Serial-to-Parallel Data Converter:** Designed, simulated, and laid out 8-bit Serial-to-Parallel data converter in Cadence’s C5 process.
- **CMOS Low-Power Voltage Amplifier:** Designed, simulated, and tested (on breadboard) a CMOS voltage amplifier with a gain of 10 which draws less than 1mA of current from a 9V power supply.
- **CMOS Full Adder:** Designed CMOS 8-bit full adder, performed logic simulation using transient analysis of digital signals, and laid out the circuit in Cadence’s C5 process.

Group

- **Wireless Data Transmission System (Thesis):** Worked in a team of 2 to design a system (confidential) to extract data from sensors, process the data and transmit processed data wirelessly to a smartphone application for analysis. System was designed using HM-10 BLE module and PIC MCU, programmed in C using MPLAB.
- **Motor-Driven Laser Alignment Station (Senior Design):** Worked in a team of 2 to design a laser lens alignment station using programmable stepper motors and ball-screw linear actuators. GUI programmed using C# and beam modeling performed in MATLAB.
- **Alignment Station 3D Modeling:** Worked in a team of 2 to model each individual component of the laser alignment station and create a final assembly in SolidWorks.
- **Freedom Photonics IC Tape-out:** Worked in a team of 5 to tape out a 152-pin, 5mm x 5mm ASIC with on-chip current and voltage DACs, op-amps, LVDS channels, and other structures for a Freedom Photonics project. Cadence TowerJazz process was used.
- **Four Function Calculator:** Led a team of 2 in design of 8-bit four-function calculator, implemented on DE2 board. Wrote code for each function using Verilog, designed schematic.
- **Test Structures IC:** Worked in a team of 3 which designed IC containing logic gates (NAND, NOR, NOT), ring oscillator, voltage divider, MOSFETs, and boost SPS circuitry. Laid out in Cadence's C5 process and fabricated for testing.
- **CMOS Audio Amplifier:** Led a team of 2 in design, simulation, and testing of a CMOS audio amplifier using ZVN3306A and ZVP3306A transistors. Input is audio signal from iPhone audio jack, output on 22-ohm speaker.

Volunteering & Service Activities

- **Reach Our City** – Travel down to the Las Vegas Strip every other Wednesday to help give out 100 free Bibles, free waters, and pray with people walking by.
- **Calvary Downtown Outreach** – Volunteer at Calvary Downtown Outreach helping to feed homeless people in the downtown Las Vegas area.
- **F.E.A.T. (Families for Effective Autism Treatment) Picnic** – Volunteer at F.E.A.T. picnic manning game stations, giving out lunch, setup, and breakdown.
- **I.K.E.D. (Introduce a Kid to Engineering Day)** – Led different age groups of 15 or more children in creating a makeshift light spectrometer using cereal boxes and CDs, answered questions about engineering and college in general.
- **Panelist** on student panel for NSF Las Vegas Scholars' Program. (Summer 2019)

Leadership

- **Former President of Tau Beta Pi, NV Beta Chapter at UNLV:** Lead chapter (containing 845 total members) by planning of service events, delegating tasks to other officers, organizing and leading initiation and orientation ceremonies.
- **Teaching Assistant:** Lead group review and study sessions as a TA, as well as office hours for several electrical and computer engineering courses, including Digital Logic Design I, Mixed-Signal Circuit Design, Digital Electronics and Digital IC Design, Digital Electronics Lab, and Memory Circuit Design. (Spring 2020 – Spring 2021)
- **IEEE Workshop Leader:** Led PCB Design, Soldering, LTSpice workshops for students at UNLV who are pursuing degrees in electrical/computer engineering.
- **Event Manager at The Plaza, Whitney Ranch:** In charge of event setup and venue management, directing and managing caterers, bartenders, barbacks, DJs, and guests for over three years. (June 2015 – September 2018)
- **UNLV Intramural Basketball Team Captain** (Spring 2018 – Spring 2019)
- **Men's Slow-pitch Softball Team Coach/Captain** (Fall 2019, Spring 2021)

Honors/Awards

- UNLV Rebel Grad Slam 3-Minute Thesis Competition **Grand Prize Winner** (Fall 2021)
- Marjorie and Victor Kunkel Scholarship (Fall 2020 – Spring 2021)
- AEE Nevada Chapter 2020 Scholarship (Fall 2020 – Spring 2021)
- **Magna Cum Laude, Bachelor of Science in Engineering** (Spring 2020)
- Wolzinger Family Engineering Scholarship (Fall 2019 – Spring 2020)
- Gilman and Bartlett Engineering Scholarship (Fall 2018 – Spring 2019)
- Earl and Hazel Wilson Scholarship (Fall 2016 – May 2020)
- Valedictorian Scholarship (Fall 2016 – May 2020)
- Millennium Scholarship (Fall 2016 – May 2020)
- Robert Mars Principal Achievement Scholarship (Fall 2016 – Spring 2017)
- Howard R. Hughes College of Engineering Dean's Honor List (Fall 2016 – May 2020)
- Named to UNLV Intramural Basketball All-Star Team (Spring 2019)
- Back-to-back UNLV Intramural 3-Point Contest Champion (Fall 2020, Spring 2021)

Professional Associations

- **Member,** IEEE (Institute of Electrical and Electronics Engineers)
- **Member,** Tau Beta Pi (Engineering Honor Society) National Chapter