A Closed-Loop System to Monitor and Reduce Parkinson's Tremors

Anthony Calvo, Linda Gong, Jake Miller, and Mike Sander
Faculty Advisor: Dr. Gary H. Bernstein
Final Report
9 May 2018

Table of Contents

I. Introduction ……... .2
II. Detailed System Requirements ……...5
III. Detailed Project Description
A. System Theory of Operation ……...6
B. System Block Diagram …….. 8
C. Accelerometer ……8
D. Bluetooth ……...10
E. Cell Phone Application ……16
F. Signal Processing ……..18
G. EMS ……..20
H. Power System ……...……22
I. Interfaces ……...23
IV. System Integration Testing ……...25
V. User Manual …… 28
VI. To-Market Design Changes …… 31
VII. Conclusions ……...33
VIII. Appendices ……....34

I. Introduction

Background and Motivation:

Parkinson's is a neurodegenerative disease that affects dopamine production in an individual's brain. The disorder progresses over time, and specifically attacks neurons in the substantia nigra area within the brain. One of the major symptoms associated with Parkinson's is the tremoring of the hands and other extremities. Currently, there is no cure for the disease, but treatments can help to improve the lives of those who have been diagnosed with Parkinson's. 10 million individuals worldwide live with the disease, and it is the 14th leading cause of death in the United States.

Right now, the most common treatments for Parkinson's tremors include taking medication to increase dopamine production in the brain and surgical procedures such as deep brain stimulation. Levodopa is a widely used medication use for the treatment of Parkinson's, but also has many side effects associated with it. Other drugs associated with Parkinson's also have a large amount of side effects, and are not effective for everyone. Surgical options like deep brain stimulation are quite risky and could lead to complications such as seizures, brain hemorrhages, or infections. There is a dearth of non-invasive treatments for the symptoms associated with Parkinson's disease, so there is a need for treatment options that do not subject patients to risky drugs and procedures.

In addition, many treatments are not capable of continually adjusting themselves to fit the needs of an individual patient, or do absolutely nothing other than to collect data on the patient. Drug dosages cannot be readily changed in response to the state of the patient. Deep brain stimulation parameters can only be adjusted by a physician, making patients less responsive to

improvements or degradation in the disease. There are commercial devices designed to track Parkinson's tremors and provide this data to doctors; however, most are not integrated with any kind of treatment. Furthermore, these devices do not provide useful data to improve on the treatment process. Therefore, it is necessary to provide solutions that allow for the monitoring of patient tremors to feedback to the treatment process.

Solution:

One potential non-invasive treatment for Parkinson's tremors is the use of an Electrical Muscle Stimulation (EMS) machine. An EMS machine is a device that is used to deliver electrical impulses via electrodes on the skin in order to contract specific muscles. Medical researchers have demonstrated that the application of an EMS signal to affected muscles significantly decreased the magnitude of a Parkinson's patient's tremors, since Parkinson's affects the peripheral reflex system of the human nervous system, and EMS pulses help to manage these reflex signals. However, no one has thought to integrate an EMS machine into a single device that monitors and mitigates tremors, or to modify EMS pulse parameters in order to combat a specific patient's tremors at any given time. Thus, our solution for helping Parkinson's patients with their tremors is a closed-loop system that both monitors and reduces tremors. The device uses an accelerometer to detect when a patient is experiencing tremors. The microcontroller on the wristband communicates via Bluetooth to a cell phone application that uses signal processing to determine if tremors are occurring. If tremors are occurring, an EMS machine is used to apply an electric pulse to the patient's wrist to mitigate the tremors. Based on the presence/non-presence of a tremor, the app sends commands back to the microcontroller. The microcontroller then implements logic to increase or decrease the amplitude of the EMS pulses

that are being delivered to the patient. If tremors are not occurring for a duration of time, then the microcontroller will turn off the output of the EMS to prevent overstimulation of the muscles.

Our design met the goals and expectations that were initially set forth. Our final design is able to collect X/Y/Z acceleration data via an accelerometer chip, which interfaces to a Bluetooth-enabled microcontroller. The microcontroller sends data to the cell phone for processing, which then analyzes the information to detect tremors. The app and the microcontroller then control the EMS output based on the magnitude of the tremors. We were able to demonstrate that the EMS would increase its output when the wearer was shaking and decrease when the user was not.

However, there were certain features that we had to give up in order to make the system safer and more user friendly. Initially, we wanted to build our own EMS machine so that our system could better control the output pulses. However, this was deemed potentially unsafe since we would be testing our device on actual humans. Therefore, we chose an FDA-approved EMS device and interfaced it with our system. In addition, to make the device more controllable, we purposely added in delays in the data collection process. This meant that the control logic was delayed a few seconds between the detection of tremors and implementation of the EMS pulses; as a result, the EMS was not as responsive as we initially hoped. Lastly, we never tested our system on a Parkinson's patient. In the future, this testing would help better tailor the system design to fit patient needs.

Ultimately, our system is able to monitor the user's state and provide targeted treatment, all in a non-invasive way. In addition, the system can be manually controlled by the user to allow for greater flexibility in its operation. The cell phone application is straightforward to operate

and the EMS is safe for daily use. The bracelet that monitors the tremors is lightweight and does not interfere with normal arm, wrist, or hand motions. This is an improvement on existing solutions for the treatment of the tremors that are associated with Parkinson's disease.

II. Detailed System Requirements

To meet our goal, our system will integrate a wearable bracelet, cell phone application, and EMS machine. The bracelet and cell phone application will communicate via Bluetooth.

Focusing first on the wearable bracelet, the bracelet needs to be small and light enough to fit comfortably on a human wrist. The bracelet will house our RSL10 microprocessor and accelerometer to allow for proper data collection. The accelerometer will measure X, Y, and Z acceleration based on the movements and tremors of the user.

The RSL10 will collect this data and will relay this information via Bluetooth to a cell phone application. In order to control which device has control over Bluetooth the RSL10 will only send accelerometer data when it is requested by the cell phone app. The bracelet components (RSL10 and accelerometer) will be powered by a battery. The battery should have at least a 48 hour lifetime (based on the battery life of a Fitbit Blaze). For troubleshooting and testing purposes, the board will also contain headers that allow for a separate DC-to-DC converter to power the board.

The cell phone application will communicate via Bluetooth with the bracelet module and perform signal processing on the data to determine if tremors are occurring. The application will request accelerometer data from the RSL10 and then will perform the FFT on the data. The app will then decide whether or not a tremor is occuring. The cell phone will change the EMS machine's parameters to mitigate the tremors based upon the magnitude of the tremors detected

between 4-6 Hz. The app will have a graphical user interface (GUI) that displays whether tremors are being detected and their magnitude. In addition, the app will save tremor data to memory to allow the patient and his doctor to review his tremor history. Finally, the GUI will also allow the user to interface to the EMS and change parameters manually.

The EMS machine will be controlled with GPIO pins on the RSL10. When tremors are detected by the bracelet, the EMS machine will apply an electric signal to mitigate the tremors. The EMS can be powered by either battery or a wall power supply. Adjustments will be made to the output of the EMS based on feedback to allow for variations in the magnitude of muscle stimulation signals. The cell phone will tell the EMS to increase the channel amplitude if tremors are occurring and decrease amplitude if they are not.

III. Detailed project description

A. System theory of operation

The system design comprises of three main sections: The control board, the EMS machine, and the cell phone application. The control board contains the accelerometer, RSL10 microcontroller, and a coin cell battery (an Energizer CR2032 coin cell). The board collects accelerometer data in the X, Y, and Z directions; this is communicated via I2C to the RSL10. The RSL10 then combines the 2 bytes of data for each direction into one `int` and transmits it via Bluetooth to a cell phone app using the Bluetooth GATT protocol. One second of data is collected, corresponding to a sampling rate of 50 Hz. The cell phone app combines the X/Y/Z data into a single magnitude by taking the Euclidean distance ($|A| = \sqrt{A_x^2 + A_y^2 + A_z^2}$), and takes a Fast Fourier Transform (FFT) of the magnitudes using the Swift vDSP API. The cell phone app

then sums the magnitudes of the FFT within the range of 4 to 6 Hz and compares the result to a preset threshold; this threshold was determined experimentally by the group.

If the magnitude of the FFT is higher than the threshold value, the cell phone app determines that a tremor has been detected. If tremors are occurring, the app sends commands (specific ASCII characters) to the RSL10 via Bluetooth to increase the amplitude of the EMS output. If tremors are not occurring, then commands are sent to decrease EMS output. Using digital input/output pins on the RSL10, the system is able to electronically press the buttons on the EMS. At its default state, the EMS amplitude is set to zero, and it does not output any waveform. Electrodes are connected to the EMS, which allow the electronic pulses to contact human skin safely. In auto mode, the RSL10 continuously gathers data and sends the data over Bluetooth, which is then processed by the app; the app sends over ASCII characters which control the EMS's button presses.

In addition to this automatic control mode, the EMS can be controlled manually. The cell phone app allows the user to toggle between automatic and manual control modes with a slider switch. In manual mode, the user can push buttons on the cell phone app that correspond to a button press on the EMS, and a new second's worth of data can be asked for by pressing the "ask for new data" button. Bluetooth is still used in manual mode to transmit data about button presses on the app to the RSL10.

A counter in the app is used to determine if the EMS is currently on or off. A "Zero EMS" option is included in case the current output is uncomfortable to the user; when pressed, the amplitude of the EMS is brought down to zero. The app saves a timestamp and magnitude for

each detected tremor, which can be accessed by connecting the phone with the app on it to a computer.

B. System Block diagram

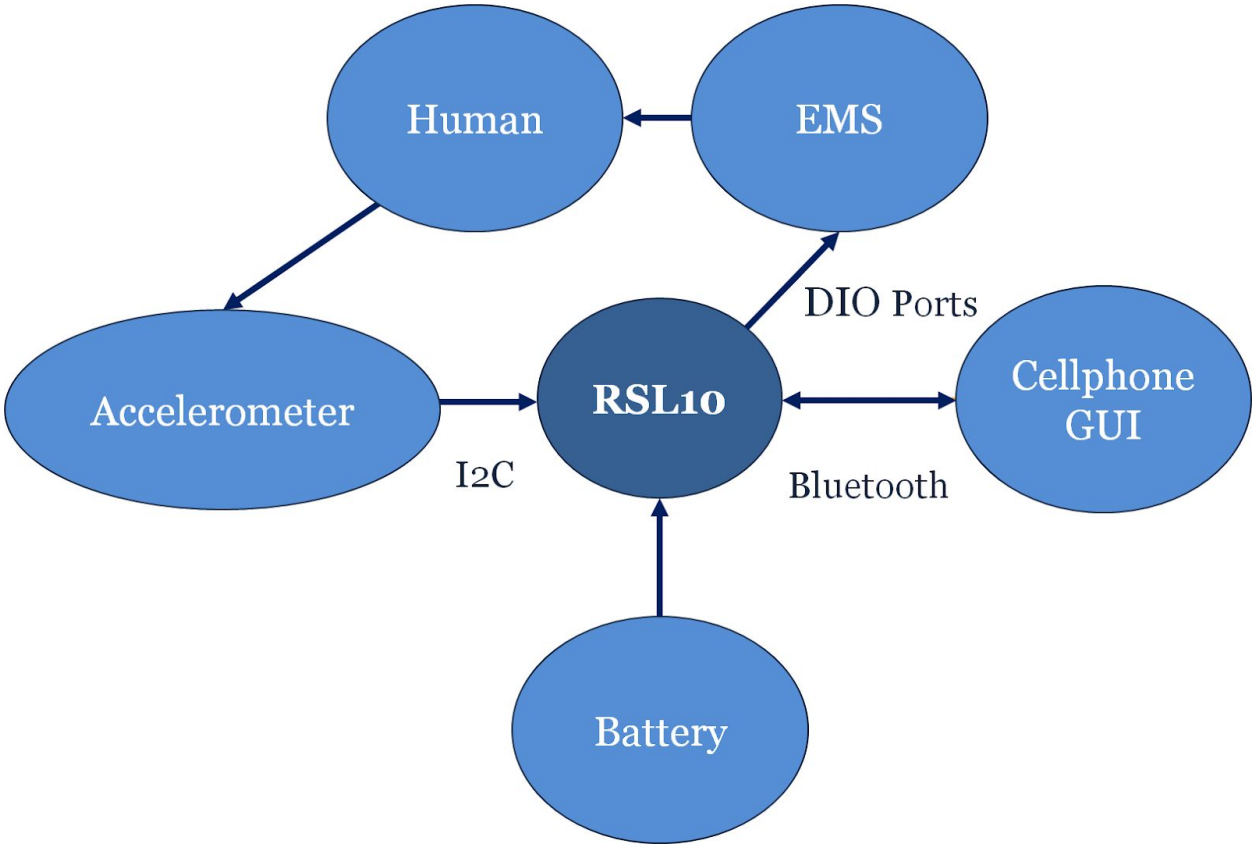

Figure 1: System Block Diagram

C. Accelerometer

An ADXL345 chip is used to collect 3-axis accelerometer data, which is transmitted to the RSL10 via I2C communication. The ADXL345 was chosen because it had two digital communication interfaces (I2C and SPI), as well as a small board footprint. In addition, the ADXL345 has low power consumption. I2C was chosen as the communication method over SPI because it required fewer data lines between the RSL10 and the accelerometer (two for I2C

compared to three or four for SPI). The slave address for the ADXL345 in the configuration we used is 0b1010011, which is used to initiate both read and write commands.

When initially powered, the device is in standby mode, and remains there until it is put into measurement mode with the POWER_CTL register. First, a number of registers need to be set to establish the desired conditions for the operation of the accelerometer. This is accomplished with an I2C write command, which involves starting a write command to the slave address using the RSL10 function Sys_I2C_StartWrite, sending the desired register address, and then sending the desired register value. The following registers are set upon startup, and their corresponding values are described: BW_Rate (address 0x2C, data rate and power mode control) with value 0x0A (output data rate of 100 Hz, bandwidth of 50 Hz; equates to a current draw of 145 μA); DATA_FORMAT (address 0x31, data format control) with value 0x02 (10 bit two's complement, right-justified with sign extension, +/- 8g range); FIFO_CTL (address 0x38, FIFO control register) with value 0x81 (Stream mode, one sample); and POWER_CTL (address 0x2D, power-saving features control) with value 0x08 (puts the device into measurement mode).

Once in measurement mode, data can be read using I2C. The RSL10 function Sys_I2C_StartRead utilizes the slave address to do this, first writing to the device with the desired register, then reading the desired number of bytes. Each direction has two registers associated with it, corresponding to 10-bit two's complement with sign extension. To combine the two 8-bit register contents into one useful acceleration value, the two 8-bit register values must be cast as `int`s; the MSB (from register X1/Y1/Z1) is bit-shifted to the left by 8 and bitwise &-ed with the LSB (from register X0/Y0/Z0) to give a full 16-bit `int` corresponding to the acceleration data. To get the complete X/Y/Z data for a certain time, all six registers must be

read in one burst, or else data for the unread registers will be lost. Figure 2, taken from the ADXL345 datasheet (linked in the appendix), shows how I2C reads and writes are achieved, with the RSL10 as the master and the ADXL as the slave.

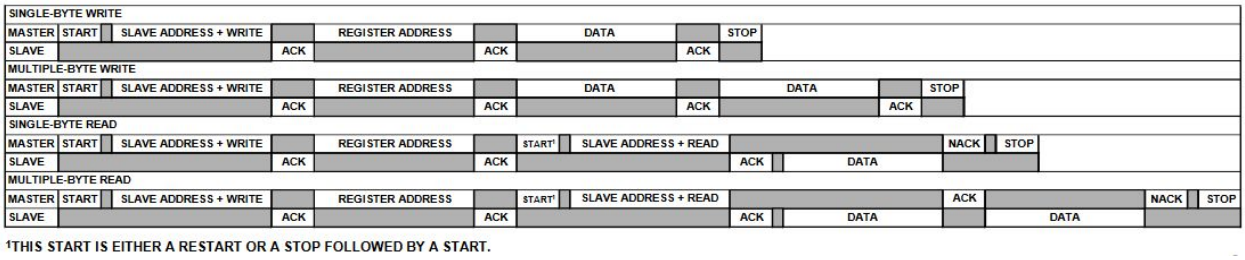

Figure 2: I2C Communication between Master (RSL10) and Slave (ADXL345)

D. Bluetooth

Figure 3 shows the flow for Bluetooth, on both the RSL10 and cell phone app ends of the communication.

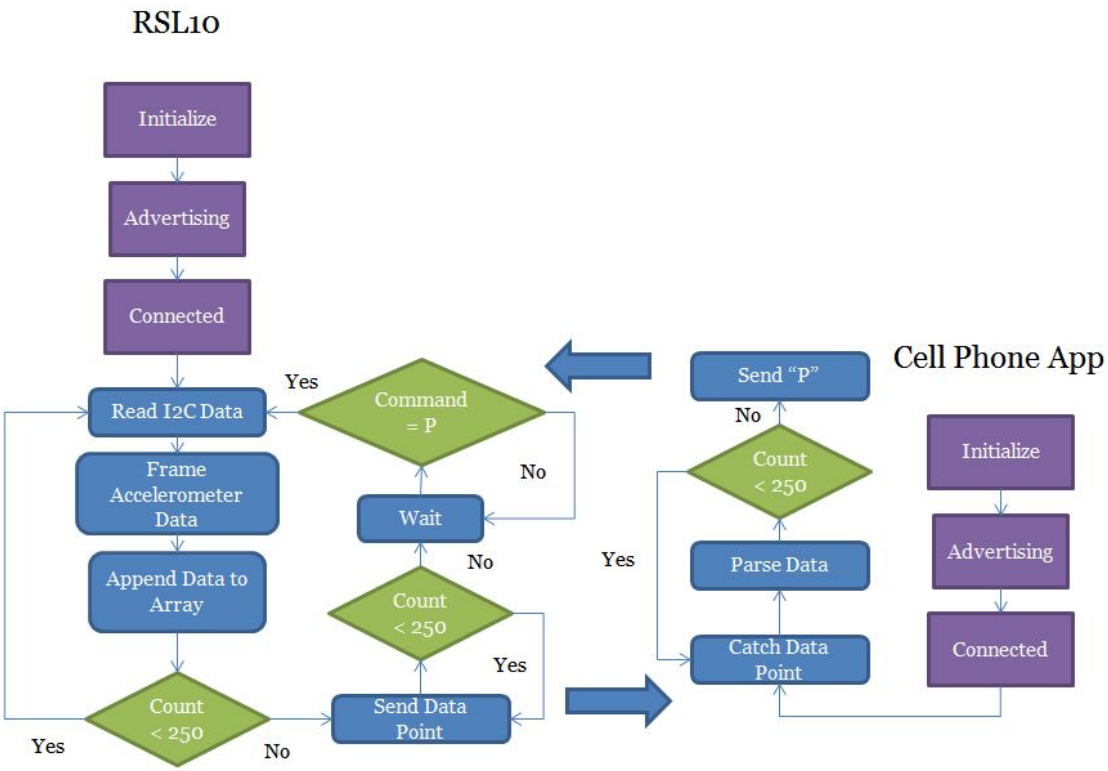

Figure 3: Bluetooth Block Diagram

Requirements:

The Bluetooth subsystem was required to be able to send data between the RSL10 and the cell phone app. Both devices needed to have the ability pair with another device, remain connected to each other, and to receive and transmit data. The Bluetooth subsystem needed to be able to send multiple bytes of data continuously from the RSL10 to the app. The system needed to broadcast at a range of 4 meters and be stable enough to successfully send data and receive a command from the app.

Overview:

The RSL10 microcontroller is enabled with Bluetooth Low Energy capabilities, which allowed us to utilize Bluetooth as the main form of communication between the board and the cell phone application. The peripheral_UART Bluetooth program (provided as sample code by On Semiconductor) was used as the basis for our Bluetooth communication code.

Our Bluetooth protocol uses the DMA buffer to transmit data through UART receive and transmit pins 4 and 5 on the RSL10. The GATT protocol is used throughout our code to transmit data, and the RSL10 is setup as a peripheral device. The GAPM protocol is used in the advertisement and connection stages of Bluetooth. Prior to connection, the RSL10 is in advertising mode and is able to connect to a central device. The GATT protocol is used for the transmission of accelerometer data.

After a central device connects to the RSL10, the microcontroller switches to connected mode. Initially, the RSL10 will sample for one second of acceleration data, and transmit this data via Bluetooth to the central device. Afterwards, the RSL10 remains connected but does not sample or transmit data until a “start” command is received from the central device. The RSL10

is still capable of receiving data via Bluetooth from the central device, which it uses to control the EMS data input/output pins. When a “start” command is received, the RSL10 will once again sample for a second of data and then transmit the collected info. The data is transmitted in Little Endian format, which means that the least significant bit is sent first. If the central device disconnects, then the RSL10 will return to advertising mode.

RSL10 Bluetooth Initialization:

The program begins by initializing the 48 MHz external crystal oscillator, which allows the RSL10 to operate at a higher frequency. In addition, the initialize function will set the VDD-RF supply to provide power to the RF and Bluetooth components of the device. Next, the RF power, RF isolation, and RF switches are set to enable a proper connection. The RSL10 waits until all RF supplies and the crystal oscillator are properly initialized and powered before proceeding with the rest of the program. The UART pins are then initialized and pull-up resistors are enabled for both the RX and TX UART pins. The UART interrupts are also initialized and configured to trigger if a keystroke is detected or data is passed through the DMA buffers.

In order to setup and control the advertising and connection process for Bluetooth, a GAPM protocol is used by the BLEInitialize function. All Bluetooth relevant interrupts are enabled, the kernel is reset, and the private device address is loaded for advertising. The Bluetooth GAPM stack is then initialized and configured. The device address, TX/RX lengths, transmit rates, and keys are set in this initialize function. The advertising protocol is then run, which fills the command message with the device information. This information is then transmitted via the kernel repeatedly until a central device connects to the RSL10. Other functions such as the GAPC_ConnectionReqInd and GAPC_GetDevInfoReqInd function are

used to transmit further information about the characteristics and connection status. The microcontroller advertises with a private address key, which we changed from the default that On Semiconductor had set in their code. Before changing the private address key, any device that ran On Semiconductor's sample code would interfere which each other (such as another RSL10 evaluation board). Therefore, we adjusted our code to avoid this issue.

Data Transmission:

The Bluetooth runs a GATT protocol for accelerometer data transmission, and adds a custom service to the protocol in order to send and receive data. The custom service also adds a specific TX and RX characteristic and defines custom character UUIDs for each. The rest of the service notification is filled with specific task IDs that are set in order to define the size and type of the data that is being transmitted. The device does not enter advertising mode until the services have been successfully added into the GATT protocol. A protocol for sending the battery level of the RSL10 is also added to the protocol, but it is not utilized or called in our code. The GATTRead service was defined but not utilized by our code. The GATTWrite service was used to catch and parse the button control commands sent by the cell phone app. This service contains a case statement that switches based on the CS_IDX_RX_VALUE_VAL, which is the ASCII character that is transmitted by the cell phone app. The service sends a command to a digital input/output based on the ASCII character and then waits for another character to be transmitted.

The CustomSendServiceNotification (CSSN) function is used to send data from the RSL10 to the cell phone. This function is called within the main code every time accelerometer data is ready to be transmitted. The CSSN begins by allotting a space in the message kernel for

the transmission. Then, the CSSN sets a pointer to the location of the accelerometer data array. A variable “t” is passed into the CSSN from the main code, which defines which value in the array is to be transmitted from the RSL10 to the cell phone app. Then, a message is sent across containing the specified data point. The size of the data point transmitted is defined to be 16 bits in order to send both the most and least significant bits of accelerometer data.

The main UART code is built around two functions: UARTEmptyRXBuffer and UARTFillTXBuffer. UARTEmptyRXBuffer is run when data is received by the RSL10 and the DMA interrupt is triggered. UARTFillTXBuffer is run when data is sent by the RSL10 and the DMA interrupt is triggered.

Cell Phone Bluetooth:

The Bluetooth code implemented on the cell phone app was sourced from an online example. The cell phone application begins by allowing the user to select which device to utilize as the data collection board. The app scans for all devices, but filters them based on UUID. The UUID for the RSL10 is stored in the code in addition to the UUIDs for the RX and TX characteristics. All devices that are not the RSL10 show up as “nil” in the device selection window. The RSL10 will show up either with the default name “Peripheral_server_UART” or with the user-defined name, “Team_2_Tremors.” The RSL10 must be powered on and in advertising mode in order for the cell phone app to detect it.

Testing:

When testing the Bluetooth operation, a TI packet sniffer and TI software were used to capture Bluetooth packets. This allowed us to determine that the RSL10 was successfully sending advertising packets, since a cell phone’s default Bluetooth cannot interface with

Bluetooth Low Energy devices. Initially, the program would blink an LED off DIO6 when the RSL10 was in advertisement mode, and leave the LED constantly on when the RSL10 was connected to a central device. This allowed us to determine easily whether our cell phones were successfully pairing to the device through the Nordic app. The Nordic RF cell phone application also allows the user to view the UUID's, RX/TX characteristics, and other information about the RSL10 Bluetooth protocol. In addition, the Nordic app allowed us to see the data being sent by the RSL10.

For initial testing of the cell phone code, an Arduino board was used to transmit data to and from the device. The online sample code had also come with Arduino code, which is why it was used to test the connection. After the cell phone application successfully interacted with the Arduino, we hardcoded the UUID of the RSL10 into the cell phone app. When the code was implemented on our final board, the LED on DIO6 was disabled to prevent it from interfering with the DIO pins that controlled the EMS buttons. This made debugging the final board slightly difficult when it was first constructed, but was not an issue once we were able to successfully program the board. The Bluetooth subsystem operated better than we initially expected for the design. Our design was operational up to a distance of 30 feet, which was much farther than our set goal of 4 feet. The phone was able to scan for devices and easily identify the control board, connect, and stay connected to the board. We were able to transmit data continuously throughout our demos and also control the EMS buttons. We did notice that if the number of cellular devices in the demo room increased, then connecting to the RSL10 via Bluetooth was slightly more difficult. The cell phone app needed to be reloaded a few times in order for the connection to be made between the two devices.

E. Cell Phone Application

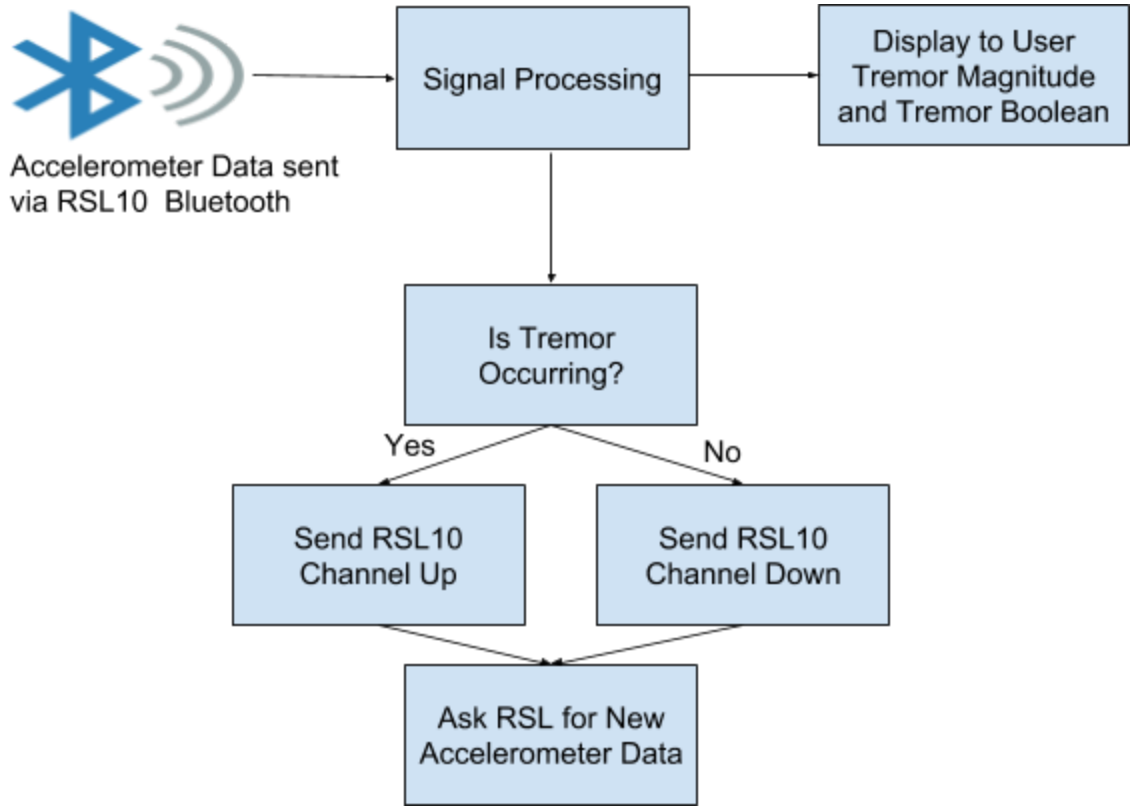

Figure 4: Cell Phone in Auto function

After selecting a peripheral device, the cell phone app automatically connects to the device and the EMS view controller opens up. The EMS view controller allows for the user to toggle between manual and automatic control of the EMS. In manual control mode, all buttons are enabled and the user is able to increase and decrease the amplitude and frequency of the EMS output. In automatic control mode, the cell phone app receives accelerometer data from the RSL10, and processes the data to determine if tremors are occurring.

The app is written in the Swift programming language for XCode, which is the software used to create cell phone apps specifically for iPhone and other iOS operating systems. In Xcode, different swift codes control various sections of the app. The UUID key file is used to scan

peripheral devices and determine which ones have UUIDs that match the targeted characteristic. The data is collected in the EMS view controller, which takes the X/Y/Z data sent by the RSL10 and combines it into a single decimal value. The accelerometer data is received in Little Endian format, so the app reverses the order of the bits in order to place the most significant bit first. The data that is sent by the RSL10 is parsed based on the start/stop framing bits, which is discussed further in the interface section. This allows the app to only store the X/Y/Z values that are passed from the board. The app then combines all three axes of data into a single value and performs an FFT on this combined data.

In automatic mode, the signal processing algorithm decides whether tremors are detected. If tremors are detected, then the app sends ASCII character “J”, to alert the RSL10 to increase the EMS output amplitude. If tremors are not detected, then the app sends ASCII character “K”, to alert the RSL10 to decrease the EMS output amplitude. After each command is sent, the app then sends ASCII character “P” to command the RSL10 to start sampling accelerometer data again. The app then prepares to catch and parse the accelerometer data once more.

In manual mode, the buttons on the cell phone app are enabled. Based on which button is pressed, the app sends an ASCII character associated with the button via Bluetooth to the RSL10. If the “Ask for New Data” button is pressed, the ASCII character “P” is sent to trigger a new data collection. An emergency stop button is also located on the EMS view controller. If pressed, the app sends ASCII character “Z”, which sets the EMS output to zero. Figure 5 shows the layout of the app.

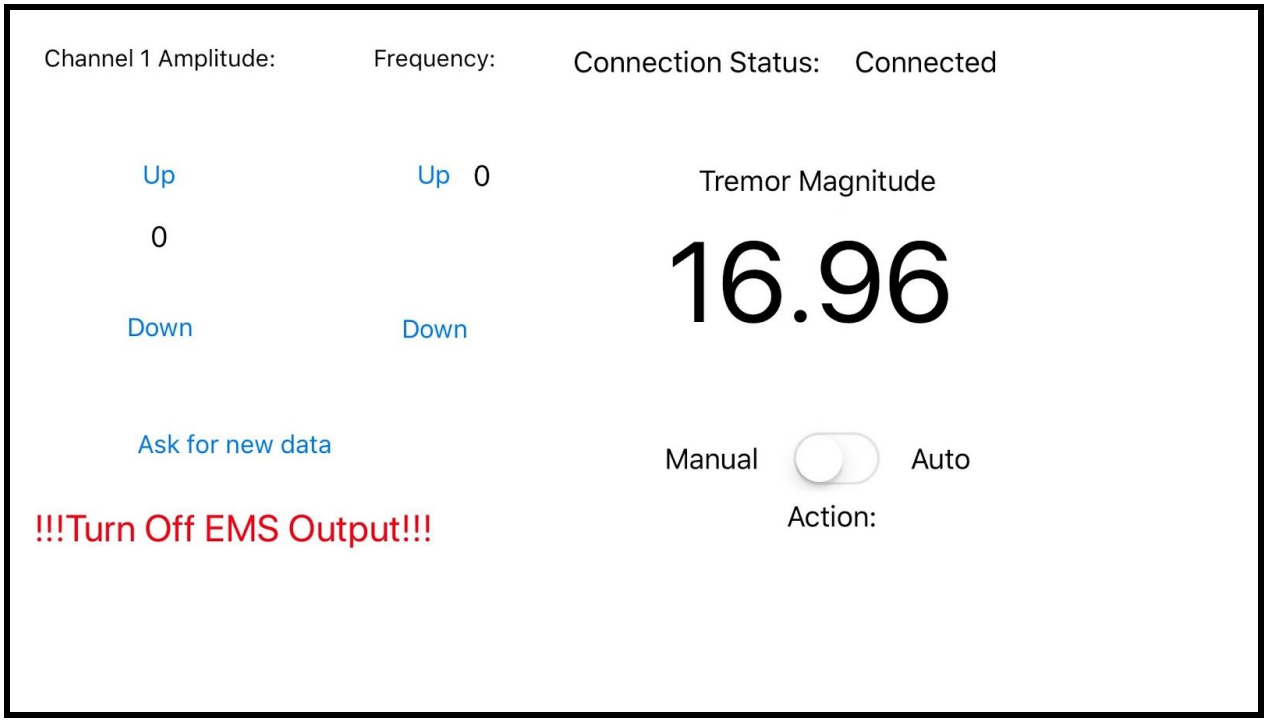

Figure 5: App in Action

F. Signal Processing

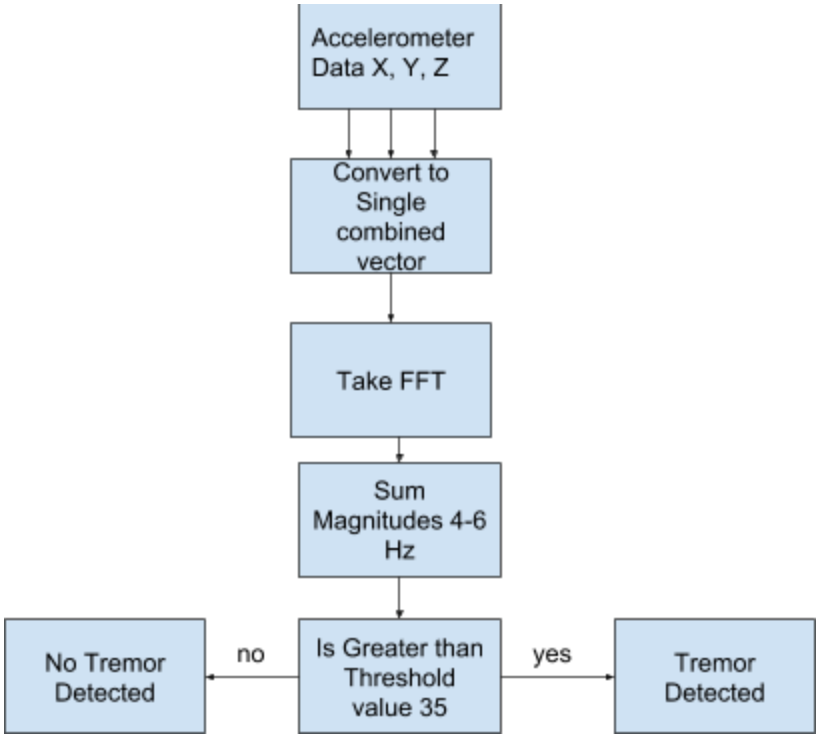

Figure 6: Signal Processing Flow Chart

Parkinson's tremors occur at very low frequencies, with rest tremors causing oscillation in the limbs at a rate of 4-6 Hz [9]. Because the frequency content of interest is so low-frequency, the requirements on the signal processing subsystem were not particularly stringent. The above flow chart shows the general implementation of the signal processing in the cell phone app. The accelerometer data needed to be sampled fast enough to avoid aliasing inside the band of interest; a sampling rate of 50 Hz (achieved by interspersing reads of the accelerometer with delays of approximately 18 ms, since a read of the six data registers took approximately 2 ms -- this timing was confirmed via the use of the Saleae logic analyzer) was sufficient to avoid aliasing in this band.

One second of data, corresponding to 50 samples (once the three X/Y/Z data points were combined into one value to give a magnitude of acceleration) with a sampling frequency of 50 Hz, was collected for processing. A time length of one second corresponds to a frequency resolution of 1 Hz in the FFT. Since there were 50 points in the time-data vector which was transformed, the resulting frequency-domain vector also had 50 points; therefore, the frequency domain result of the FFT showed frequency content at integer values, from 0 to 49 Hz. Swift contains an API for signal processing called vDSP; an example found online of a basic FFT program was used as a starting point for the Tremors cell phone application [11]. The frequency domain results were normalized in order to allow for consistent analysis between different second-length blocks of data.

After the FFT was performed on the vector of acceleration magnitude data, the frequency band of interest was investigated to determine if a tremor occurred during the period of time during which the data was collected. To do this, the normalized magnitudes corresponding to

frequencies 4, 5, and 6 Hz are summed together and compared to a threshold value. The threshold value was determined experimentally to be 35.0. If the sum of the magnitudes contained in the band of interest is above this threshold, a tremor is determined to be occurring; if the sum is below the threshold, a tremor is not considered to be occurring.

G. EMS

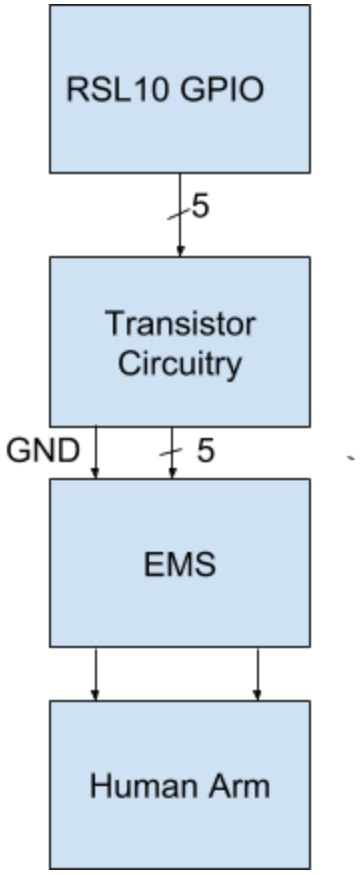

Figure 7: EMS Flowchart

The board interfaces to the EMS machine via a Molex connector. The flow chart above shows the general system. The Molex is the interface between the transistor circuitry on the monitoring board and EMS. As mentioned in the Cell Phone subsystem, ASCII characters are sent to the RSL10 via Bluetooth that tell the RSL10 to write given pins high, wait a small delay, and then write them low. The delay exists because there is a limit of how fast the EMS device

can respond to voltage changes (this was determined experimentally). Each button on the EMS consist of an open between ground and the EMS's control circuitry (the commercial control circuitry, not the one on our board). Originally the device had rubber buttons with a conductive material beneath the rubber that would short out the open and perform the function of the button. The team's transistor circuit is designed to "short" the buttons electronically.

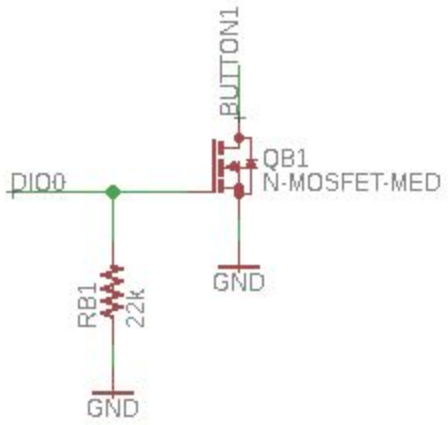

Figure 8: EMS Transistor Circuit

The Figure above shows a general schematic for the transistor circuit. An NMOS transistor is placed across the open circuit of the EMS device. The drain is connected to the high side of the open circuit via a Molex connector and the source is connected to the grounded portion of the open circuit on the EMS. The output of the RSL10 (a DIO pin) is connected to the gate of the NMOS, and the grounds of the EMS and the circuit are connected. A resistor is connected from the gate of the transistor to ground to ensure the gate discharges after being turned on. This ensures writing a high voltage to the transistor does not persist. There are five transistors in the circuit in order to accommodate pressing five buttons. The output of the circuit is sent over the Molex connector along with the ground wire. The board can "press" the following buttons:

channel one up, channel one down, set, left, and right. These buttons give full control of channel one's current output (0-100 mA according to the manual), pulse width duration, and frequency. For the purpose of the project it is assumed the device is always running in the constant mode of the EMS machine (MPO 85000 Combo).

H. Power System

The power system is responsible for powering the RSL10 microcontroller, the ADXL345 accelerometer, and the Bluetooth antenna on the control board. The RSL10 also must power the digital input/output pins such that it can control the transistors that control the EMS machine. The battery chosen to power the RSL10 must be capable of providing the correct voltage to the chip while also providing enough current to power the board for the uses listed above. The battery must also have enough capacity to power the RSL10 for at least 48 hours. Due to the wearable nature of the control board, the battery must also have a small profile such that it is not uncomfortable to wear on the wrist along with the board. For this reason, a coin cell was chosen for the power system.

The RSL10 has a voltage regulator on chip, which means that an external voltage regulator is not necessary for the battery. The RSL10 supports a supply voltage range of 1.1 to 3.3V. We chose to run at the ~3V range so that digital output pins run at a higher voltage, making our transistors for controlling the EMS machine run as we expect.

At 3V, the Bluetooth max receiving current is 3mA, and the transmitting max current is 4.6mA. This means our battery had to be able to provide at least a 7.6 mA current temporarily. Current from the transistor circuits could contribute, but the current they draw is several factors below 7.6 mA, so it was ignored for our max current requirement.

The spec sheet says audio streaming with the RSL10 uses 1.8 mA continuously (at 3V); we used this estimate for our capacity requirement calculations. Once again, our transistor circuits take a minimal amount of current to turn on, and are only turned on for fractions of a second; therefore, we ignored their current draw for our calculations. Thus, 1.8 mA is our expected average current draw. For 24 hours, our battery capacity requirement is 86.4 mA-hr.

The 2032 coin cell from Energizer has a capacity of 240 mA with a cutoff voltage of 2V. We measured our minimum turn-on voltage for our transistors as ~2.7 V; this cuts the 2032's capacity down to about half (~120 mA-hr) based on current discharge characteristics provided by Energizer. Thus, the battery provides more than enough capacity to last 48 hours. Energizer doesn't provide a max current, but lists pulse discharge characteristics for the 2032 at 6.8 mA. Although this is .8 mA below our max current requirement, we found that some other CR2032 manufacturers run their pulse drain tests at currents above our requirement. Therefore, we assumed that the 2032 could safely provide our needed max current of 7.6 mA.

The 2032 takes up an area of 1 cm^3 and has a weight of 3g; we determined that this would satisfy our requirement for a power supply with a small impact on board size and weight.

I. Interfaces

Data Transmission:

The accelerometer data was sent via Bluetooth from the RSL10 to the cell phone app. The data is sent as a stream of X/Y/Z points with start/stop framing bits to facilitate the parsing of it in the cell phone app. The RSL10 initiates a read of the accelerometer, and then stores the data in an array. A start bit of 0x1122 is inserted at the beginning of the data, and a stop bit of 0x8899 is appended to the end of the accelerometer data. These data points are then sent via

Bluetooth to the cell phone app. The app then samples each data point, but will drop the data if it does not get a start bit. Once the start bit has been received, the cell phone app then begins to store the X/Y/Z data points for signal processing.

Antenna:

The antenna for the Bluetooth subsystem increases the range of the device and the power of the Bluetooth signals. A 2.4Ghz SMD antenna is connected to pin 1 on the RSL10 via a coplanar waveguide. A coplanar waveguide was chosen for the matching network because it was both space efficient and effective in providing a 50 Ohm match. LineCalc, which is a tool within Advanced Design System (an RF design software), was used to calculate the line width for the board. The length of the coplanar waveguide did not matter, but we chose to keep the length as short as possible to avoid loss. The area on the board surrounding the antenna was kept free of components and ground planes to allow for maximum transmission from the antenna. Figure 9 shows the line calculations used to determine the coplanar waveguide parameters for the board.

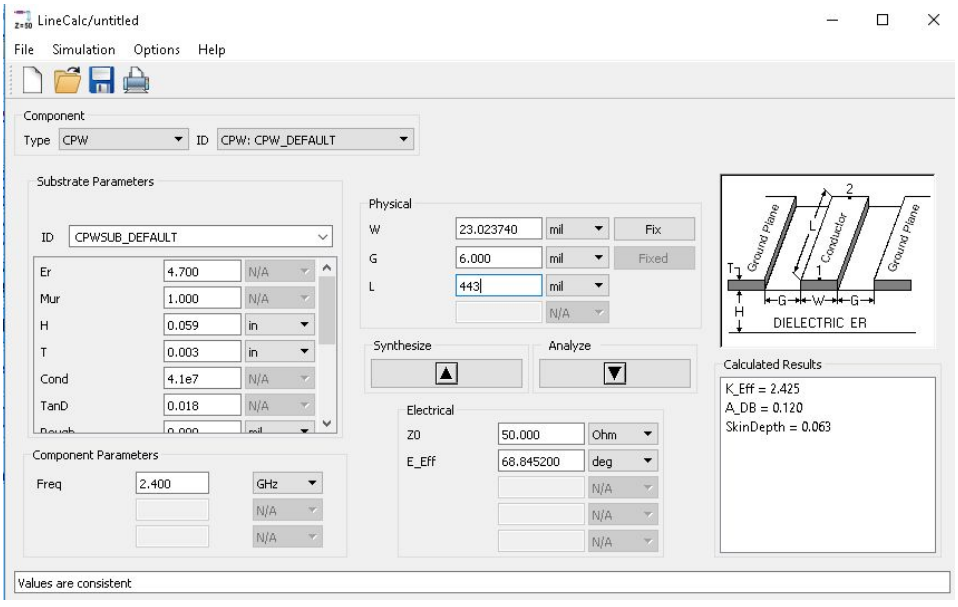

Figure 9: Line Calculation for Coplanar Waveguide

Figure 10 shows the Eagle layout for the antenna and its coplanar waveguide; note that the thickness of the trace and the distance of the ground planes from the trace correspond to the parameters determined by the line calculations previously shown.

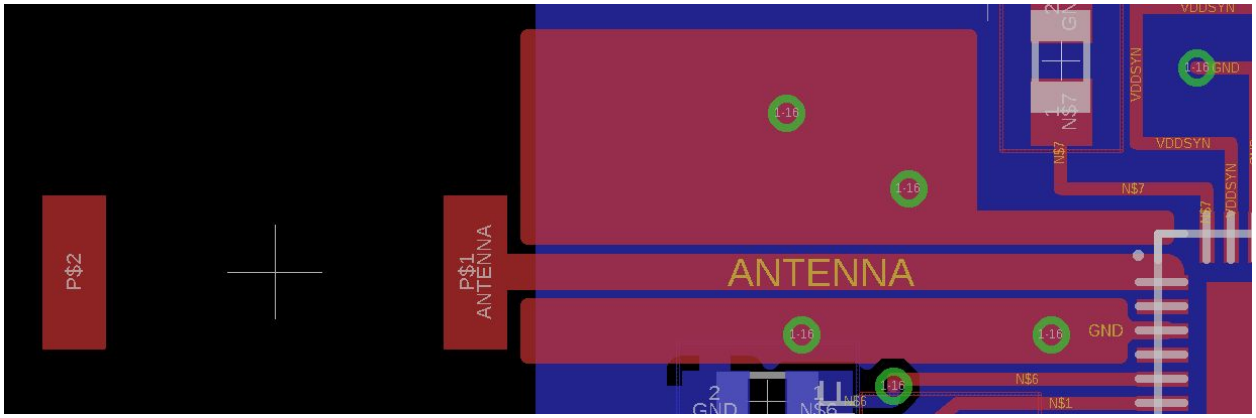

Figure 10: Eagle layout for antenna

IV. System Integration Testing

6.1 Subsystem Testing Description

Our subsystem testing consisted of multiple stages, corresponding to the order of operation of our block diagram. The first step after putting together our board was to make sure that the correct voltages were located where we expected them. Immediately, we found an issue; RSL10 pin 9, which had appeared to be an output on our original schematic of the RSL10, in reality was an input which needed a connection to VDD (confirmed by an updated datasheet released after we submitted our board). On our board, we solved this problem with a jumper wire, and updated our EAGLE board drawing accordingly. After that fix, our next task involved making sure we could program our board using the SEGGER J-Link EDU Mini, which connected from a laptop's USB port to the FTSH connector on our board. At first, this gave us an error ("Failed to Initialize RAM code"), which was due to an unconnected oscillator. After

re-soldering the oscillator, the board could be programmed with a simple blinking LED sample code provided by ON Semiconductor, showing us that our board was operational.

Once the board was up and running, the various subsystems were tested. We followed the order of our block diagram, first testing the subsystems that utilized the RSL10 and then the subsystems that utilized the cell phone app; we followed this plan because the failure of an individual subsystem would require a fix before the whole system could work. First, we confirmed that the RSL10 could read and write via I2C to the ADXL345 on our board, both with a single-byte read of the device ID register (which returns 0xE5) and then with a multi-byte read of the six X/Y/Z acceleration data registers. The Saleae logic analyzer confirmed that the communication worked as expected. Next, we confirmed that a single ASCII character could be sent via Bluetooth from the RSL10 and received by the Nordic nRF Connect app, proving that the RSL10 on our board was capable of Bluetooth communication.

After testing the RSL10 subsystems, we moved on to testing the cell phone app's subsystems. First, we confirmed that we could receive data sent by the RSL10 on the cell phone app, making sure that the bytes of the received data were in the correct order. We displayed the received character on the Xcode readout to confirm that it matched the sent character. Afterward, we made sure that the RSL10 could receive data from the iPhone app. To do this, we sent different ASCII characters, which correspond to different output pins of the RSL10 which were connected to LEDs driven by the transistor circuit meant to interface the EMS. By sending different ASCII characters, we were able to verify that the correct LED turned on, confirming that the RSL10 could receive data from the iPhone app. The code was then adapted to show that writing high then low to the transistors was able to control the EMS buttons. Finally, the saving

of tremor data was tested by checking the iPhone's internal text files and ensuring the magnitudes saved only corresponded to data that was sampled when tremors were detected. With that, our testing concluded, and the remaining steps were to finalize the entire system.

6.2 Show how the testing demonstrates that the overall system meets the design requirements

The process of powering up the board proved that the battery subsystem was functional and supplied the right voltage. Initially it did not power, meaning that something was not effective. Testing the power system by adding an LED allowed us to confirm that the device was in fact receiving power from the battery. Although the power requirement of the board lasting 48 hours was not specifically tested, the power supply experienced little to no power degradation throughout development, testing, and demonstration (taking place over a two week period). Once the board was programmed the power system was verified as successful because in order for the RSL10 to program, the voltage inputs must be correct. If we had simply done it based of the LED, it was possible that we could have been over- or under-powering the device, but still supplying the proper power for the LED to illuminate.

Once a blink LED program on the board worked, the “ability to write out to GPIO pins” requirement was met. In testing the transistor circuit, the EMS subsystem was verified since writing to the pins resulted in the pushing of buttons. Testing that the RSL10 could both read and write from the accelerometer with I2C confirmed the accelerometer subsystem requirement. The Bluetooth requirement of being able to send data was confirmed by sending ASCII characters over Bluetooth in both directions.This was then implemented along with the iPhone completing the requirement of transmitting accelerometer data over Bluetooth between the cell phone and the RSL10. Furthermore, once tested with the app it was shown it would only send data when it

was requested from the phone. The requirement to save data was confirmed by comparing a text file to the iphone output values.

V. Users Manual/Installation manual

7.1 How to install your product

Cell phone app:

1. Ensure that you are using a Mac computer running the latest iOS update.
2. Download XCode from the Apple Store and install it on the computer.
3. Download Swift files from our team website.
4. Open XCode on your computer.
5. From the "File" drop down menu, select "Import" and then select the Tremors GUI project folder.
    a. This should import the entire project into XCode.
6. Press the "Run" button at the top of the screen to download the app onto your cell phone.
    a. In order to run the app on your phone, you will need to sign into your Apple account and set it as the developer.
    b. With a free developer account, you can download the app onto three different cell phones.
7. Open your cell phone and navigate to the Settings, then click "General", then click "Verify Certificates." Verify the certificate for the Tremors app.

7.2 How to setup your product

Product setup consists of configuring the tremor reduction board properly, as well as the tremor GUI application (Note: Any inability to properly configure the device will result in this group no longer being liable for direct or indirect losses, arising from any such injury, illness, decapitation, heart palpitation, physical pain, emotional pain, damage, loss accident, or expense including consequential loss or economic loss however caused). To configure the tremor reduction board please follow the steps below:

1. First make sure that the tremor reduction board and the EMS device are powered off. Ensure there is no coin cell in the board.
2. Connect the Molex connector (the white connector) coming out of the EMS device to the Molex connector on the board.
3. Place a coin cell battery (Energizer CR2032) into the battery holder. Press the reset button. If the light blinks on briefly, the process is complete. Otherwise, flip the power selection switch to the other side and confirm that the light blinks on when the reset button is pressed.
4. Strap the board to your wrist using the velcro wrist strap that is provided. The board can be connected in either direction, depending on which arm the patient prefers to monitor. The board should be placed as close to the hand as possible to best gather data.

7.3 How the user can tell if the product is working

Press the reset button and observe if the light blinks. Then to ensure the application is working correctly, open the cell phone application entitled “Tremors GUI.” Connect to the Team _2 _Tremors device. Press “ask for new data” while shaking the device. Wait 10 seconds and if

the tremor magnitude has not changed, there is likely an issue with the Bluetooth connection. Simply kill the app, reset the board, re-open the app and attempt to connect again.

7.4 How the user can troubleshoot the product

In the event that the app and the board are not working properly (e.g. the board disconnects quickly, the EMS cannot be controlled with the buttons in the app, or data is not being sent to the phone), try these common fixes:

1. Reset the board and the iPhone app. Make sure to also remove the app from running in the background. Then reconnect.
2. It is possible that there are too many other Bluetooth devices in the area or that another Bluetooth device is attempting to broadcast with similar identifiers, such as the device address or characteristic. In this case, move the device away from the presence of other devices and reconnect. Once the device is connected, the presence of other devices should have little impact.
3. Ensure that there is plenty of power in your iPhone. If the iPhone enters low power mode, performance may decrease.
4. Remove all apps from the background of your iphone and then restart your phone. Wait about a minute for your phone to be fully booted, then run the app.
5. If a Bluetooth connection is in fact established (the presence of the return to first screen button in the upper right hand corner of the app means it is not connected), and the EMS buttons are not being pressed, there could be an issue with wiring. If the app is receiving data, but you cannot press any of the EMS buttons, use a voltmeter to measure the voltage at the digital input/outputs on the Molex. If the voltage output changes with a

button press, then the issue mostly lies within the hardware connection to the EMS. If no voltage changes with a button press, then there is an issue with the Bluetooth connection. Check the wire connecting the board to the EMS machine. It is possible the connection was severed. In this event, carefully ensure the board is powered off and re-solder the wires to the contacts. Then wrap them in heat shrink and place the ends into a Molex connector. Make sure to keep track of the wire ordering. When looking at the connector coming off the EMS board, line up the wires in this order: channel 1 down, channel 1 up, left, right, set, ground.

7.5 How the user can access their Tremor Data Log

1. Connect the iPhone to the laptop and open up Xcode.
2. In the toolbar at the top of the screen select Window, then select Devices and Simulators (Alternatively, press shift+command+2).
3. Find the device and the Tremors GUI application. Press the settings gear icon and press "download container."
4. Save the container. Right click on the file and press "show package contents."
5. The text file will be saved in the appdata folder.

VI. To-Market Design Changes

Due to constraints on our team's budget and time, the project could stand to undergo a number of improvements before being sold in the marketplace as a viable product. The first and most important change would be to move the signal processing subsystem, currently located in the cell phone app, onto the board. This change could be accomplished in a variety of ways: the RSL10 could still be used as the intelligence of the board while its LPDSP32 processor, which

we lacked access to and could not afford to purchase the necessary hardware and software for, could be used to perform the signal processing; a different microcontroller could be used, such as a dsPIC, which would require changing the board; or the RSL10 could be combined with a DSP co-processor. This change would make two key improvements possible between our prototype and the new version. First, the EMS could be controlled without needing a phone, getting rid of one restriction on the current system. Second, the app would transition from being a necessary intelligence of the system to being an additional feature for the user focused only on displaying information. These improvements would make for a more user-friendly system.

Another potential change could be switching from a commercially available EMS device to a custom EMS device constructed specifically for the project. For this year's iteration, an FDA-approved device was used, but the modifications made to the device no longer allow the device to meet FDA regulations. If a custom EMS was created, the team would have greater control of the output waveform, and could potentially add Bluetooth capabilities to remove the need for a hardwired connection between the EMS and the board. Alternatively, a more advanced (and more expensive) commercial, FDA-approved EMS device could be purchased, potentially already with Bluetooth capabilities; a device of this nature was outside the budget range of this year's project.

A number of minor changes could be made to this year's setup, as well. The size of the board, while satisfying the requirement for a small profile for this year's design, could continue to be reduced; the goal is to make the board as small as possible to allow the patient to unobtrusively wear the device on his or her wrist. Our original idea to utilize a rechargeable battery (such as the battery that powers a wearable device like a FitBit) rather than replaceable

coin cells could also be attempted; we decided that an initial prototype did not need this feature, but it would be nice to have for a product in the marketplace. The current control system is simple, so a more sophisticated control system could be developed. This would allow the system to better fine-tune its efforts to mitigate tremors, potentially incorporating more EMS parameters like frequency and pulse width in its logic, as channel amplitude is not the only EMS condition that affects tremor magnitude.

If more channels from the EMS are accessed, additional wires would be needed to connect the EMS to output ports of the RSL10. A potential way to achieve this could be if the TDO, TDI, and TRST wires are no longer connected to DIOs 13 to 15. Right now, our board design allows the RSL10 to be programmed with either a JTAG (which requires five connections) or SWD (which only requires two wires, JTAG_TMS and JTAG_TCK) interface. Eventually, we transitioned to programming the board using only SWD, meaning that those three DIOs could be used to supply simulated button presses using a transistor to the EMS. Improving the device's Bluetooth reliability is another potential improvement.

Finally, clinical trials would be useful to see how the device affects real-life Parkinson's patients. These changes represent the most prominent next steps before the product could be brought to market.

VII. Conclusions

Overall, the project was a success. The system worked as expected, as we could gather acceleration data using the ADXL345 and RSL10, transmit the data over Bluetooth to a cell phone application, process the data and determine if a tremor was occurring, and send back

commands over Bluetooth to control the output of an EMS machine meant to mitigate the tremors. The design process provided good opportunities to learn embedded systems programming, hardware analysis, wireless communication, signal processing, biomedical device operation, and app development skills, tying together a number of different EE courses taken at Notre Dame.

The purpose of this project was to be a proof of concept, showing that this type of system can achieve the desired goal of monitoring and reducing Parkinson's tremors. Since the project was a success, the system should be pursued as a viable option and further developed in the future.

VIII. Appendices

Resources:

1. http://www.parkinson.org/understanding-parkinsons/what-is-parkinsons
2. https://www.parkinsonassociation.org/facts-about-parkinsons-disease/
3. http://www.parkinson.org/Understanding-Parkinsons/Treatment
4. https://www.mayoclinic.org/diseases-conditions/parkinsons-disease/diagnosis-treatment/drc-20376062
5. https://www.rxlist.com/sinemet-side-effects-drug-center.htm
6. https://www.mayoclinic.org/tests-procedures/deep-brain-stimulation/about/pac-20384562
7. https://www.ncbi.nlm.nih.gov/pubmed/26342942
8. https://www.thegoodbody.com/electric-muscle-stimulator/
9. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3722015/
10. https://chortle.ccsu.edu/AssemblyTutorial/Chapter-15/ass15_3.html
11. https://github.com/christopherhelf/Swift-FFT-Example
12. https://learn.adafruit.com/crack-the-code/overview
13. https://www.microwaves101.com/encyclopedias/fr-4

Links to data sheets:

RSL10: http://www.onsemi.com/pub/Collateral/RSL10-D.PDF
ADXL345: https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf
48 MHz oscillator: http://www.ecsxtal.com/store/pdf/ECX_1247.pdf
32 kHz oscillator: https://support.epson.biz/td/api/doc_check.php?dl=brief_FC-135R_en.pdf

Chip antenna: https://linxtechnologies.com/wp/wp-content/uploads/ant-fff-chp-x.pdf
FTSH connector: http://suddendocs.samtec.com/catalog_english/ftsh_smt.pdf
J-Link EDU Mini: https://www.segger.com/products/debug-probes/j-link/models/j-link-edu-mini/
CR2032 battery: http://data.energizer.com/pdfs/cr2032.pdf

Complete Board Files and Code Listings can be found on the Tremors team website, located at: http://seniordesign.ee.nd.edu/2018/Design%20Teams/tremors/docs.html