Electromyography Analysis and Recognition for Human Device Interface

Engineering 090: Senior Design Project
Published May 2014

David Nahmias
dnahmias1@gmail.com

Abstract

The desire to control and manipulate computers and human device interfaces (HDI) in more natual ways has been sought after for some time. This Engineering 090 project looks use hand movements and motions to control an HDI. This is done through processing, analysis and recognition of signals from surface electromyography (EMG) sensors and an accelerometer and gyroscopes to control an HDI. The HDI controlled in this project is a computer mouse and arrow keys on a keyboard via the Makey-Makey. This project implements electronic circuit design to filter the raw EMG signals to meaningful EMG signals, digital signal processing methods to characterize the EMG signals and machine learning techniques to classify the EMG signals into hand gestures using Artificial Neural Networks (ANN). Afterwards, based on the evaluated hand gesture or movement, signals are sent to a Makey-Makey to control the mouse and designated keys on the keyboard. The system, after implementation, is able to recognize with nearly no error four hand gestures within approximately an eight of a second. This result allows for real time accurate control of the desired HDI. This system in the future can be implemented to control and manipulate any HDI that can be driven by General Purpose In/Out (GPIO) signals. Finally, electromyography driven HDIs have the potential to control more complicated systems such as machines in hazardous areas or prosthetics, the possibilities are endless.

Department of Engineering, Swarthmore College

Contents

List of Figures

List of Tables

1. Introduction

The user interface of a computer has stayed mostly constant since the inception of the keyboard and mouse. Some improvements to these systems have come through more ergonomic designs of these devices and more interactive graphical user interfaces. However, despite all these advances, using a mouse or keyboard do not come as natural human movements. The desire to control computers more naturally through hand motions and gestures has been sought after for some time. Ideally certain natural and intuitive motions and gestures could control features and commands of the computer. Technologies such as touch screen devices, the Xbox Kinect and the Leap Motion have come along as possible solutions to this problem. These technologies utilize capacitive touch sensing, computer vision and infrared sensing, respectively, to recognize gestures and positioning which can then be translated to computer commands such as a mouse click or keyboard inputs.

This project looks at an alternative method to determine motion and gesture recognition. This project attempts to utilize electromyography (EMG) signals from the hand and arm to recognize and interpret different hand and finger movements. EMG signals, after processing, have been shown to be effective in differentiating between different finger movements and hand gestures. The principle of EMG sensing is to measure voltage differences across various muscles and based on these differences recognize hand gestures being executed. The difficultly in this task is acquiring clear easily interpretable EMG signals to be used to recognize hand and figure movements. An added challenge is that this acquisition and recognition of EMG signals must be done in real-time in order to create an effective system for controlling computer computer commands. Furthermore, in addition to the EMG electrodes placed on the arm, an accelerometer and gyroscope is integrated to allow for recognition of more dynamic movements and hand gestures which allows for more control. The human device interface (HDI) which will be integrated and controlled by this system is the mouse and designated keys on a keyboard.

This project integrates many components together to obtain the overall desired goal. The center piece of equipment, the BeagleBone Black board communicates with all the peripheral data acquisition devices and HDI. Furthermore, it is also where the EMG signal analysis and recognition software is run. Good communication between the AD7689 analog front end board, the MPU6050 accelerometer and gyroscope and the Makey-Makey are all essential. The raw EMG signals, acquired via surface electromyography, are also processed using an AC couple, instrumentation amplifiers and 60Hz notch filters in order to obtain clear EMG signals for the analog front end to transfer to the BeagleBone Black board. In addition to the physical integration of the many components, the integration of the EMG recognition software which utilizes Artificial Neural Networks (ANN) to characterize the EMG signals, also integrates with the hardware and acquisition of data via a multi-thread software architecture controlled by mutex locks. This software is driving force, hosted on the BeagleBone Black board, that acquires the post-processed EMG input signals and ultimately controls the desired HDI. All processing is designed to be run in real time because of the necessary inter-device communications and desired real time control of the HDI.

2. Materials and Methods

2.1 General System Diagram

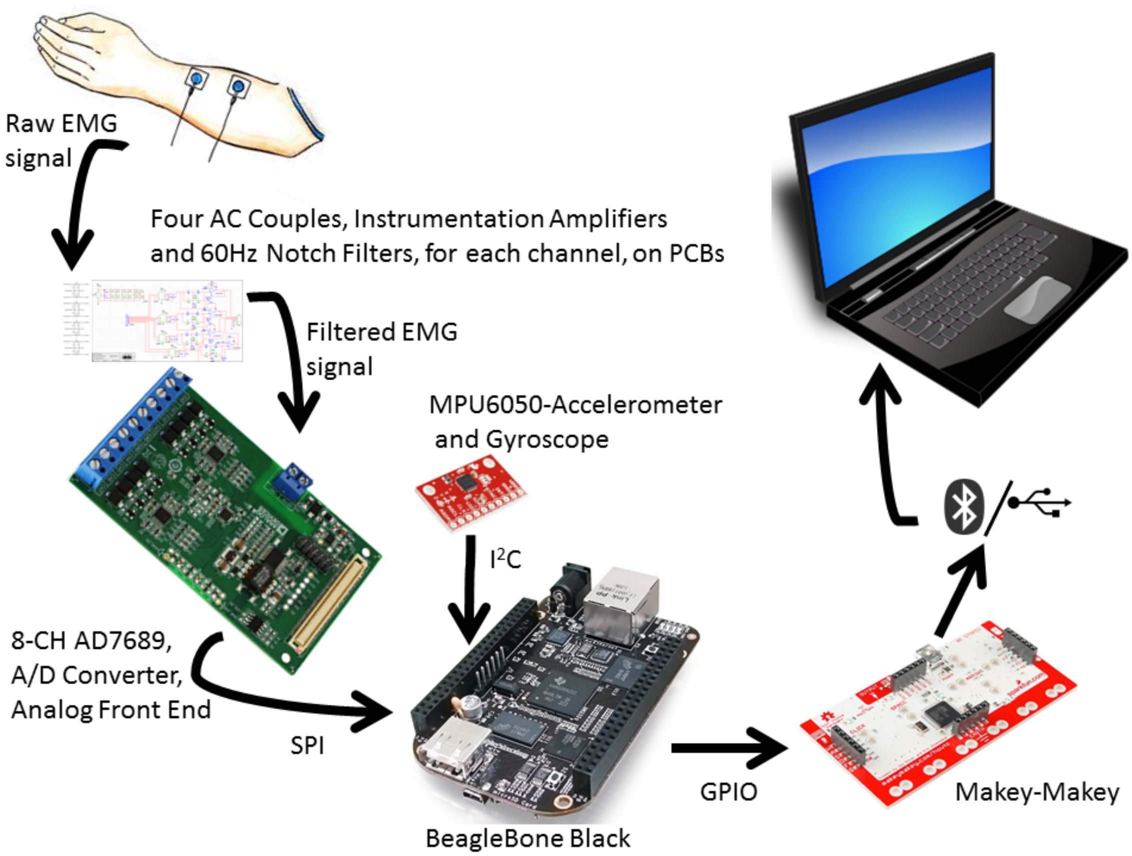

Figure 1: *General system diagram of hardware used*

2.2 Acquisition of EMG Signal

The analysis and processing of the EMG signals is done on Texas Instrument's BeagleBone Black Board. This board uses an AM335x 1GHz ARM Cortex-A8 processor running the Angstrom with eMMC flasher release date 2013-09-04 distribution of Linux.[15] Importantly for this project the BeagleBone Black boasts two 46 pin headers, labeled P8 and P9, which are used for Serial Port Interfacing (SPI), Inter-Integrated Circuit (I^2C) and General-Purpose Input/Output (GPIO). The pins on the BeagleBone Black are labeled using the notation P9_1, which would refer to pin 1 on the P9 header. More technical information about the BeagleBone Black can be found online [11] or through the Bad to the Bone manual [1].

The analog front end device used is Analog Device's AD7689.[7] This is a low cost, 16-bit 250 kSPS, 8-channel, isolated data acquisition system.[7] This allows us to acquire at maximum eight channels of EMG signal data. From previous studies four has sometimes been enough and very rarely have more than eight channels have been used thus this Analog to Digital (A/D) converter is chosen.

These devices are used in conjunction to obtain and analyze the EMG signals.
EMG signals vary at a maximum of about 500Hz [2]. Thus, as per Shannon's sampling theorem, in order to ensure no loss of information in the signal, the EMG signals are sampled at a rate of

$$500\text{Hz} \times 2 = 1000\text{Hz}.$$

This sampling rate is achieved by implementing a 'sleep' in the data acquisition code.
The exact function used is `usleep(850);`.
The value of 850, which equates to a sleep time of $850\mu s$, is used since this was experimentally found to produce a sampling rate very close to 1000Hz. Thus, in an attempt to have the system analyze the EMG signals in real time, the EMG signals are analyzed in packets of an eight of second of data. Since data packets of length of a power of 2 are desired for digital signal processing purposes, 128 long packets of data are used to analyze a single hand gesture since $\frac{128}{1000} \approx \frac{1}{8}$.

2.2.1 Hardware Acquisition of EMG Signals

The EMG data is physically acquired through placing gold platted Grass electrodes filled with Ten20 conductive paste on the arm. Eight electrodes are placed in pairs, two along the lateral side of the forearm, two on the medial side of the forearm, two on the anterior side of the forearm and two on the posterior side of the forearm. They are placed along the length of the arm so that they measure the electric potential across the same muscle fibers. A ninth electrode is placed on the elbow, where there is little muscle, as a ground reference.

An image of the arm band, which houses the electrodes, can be seen below:

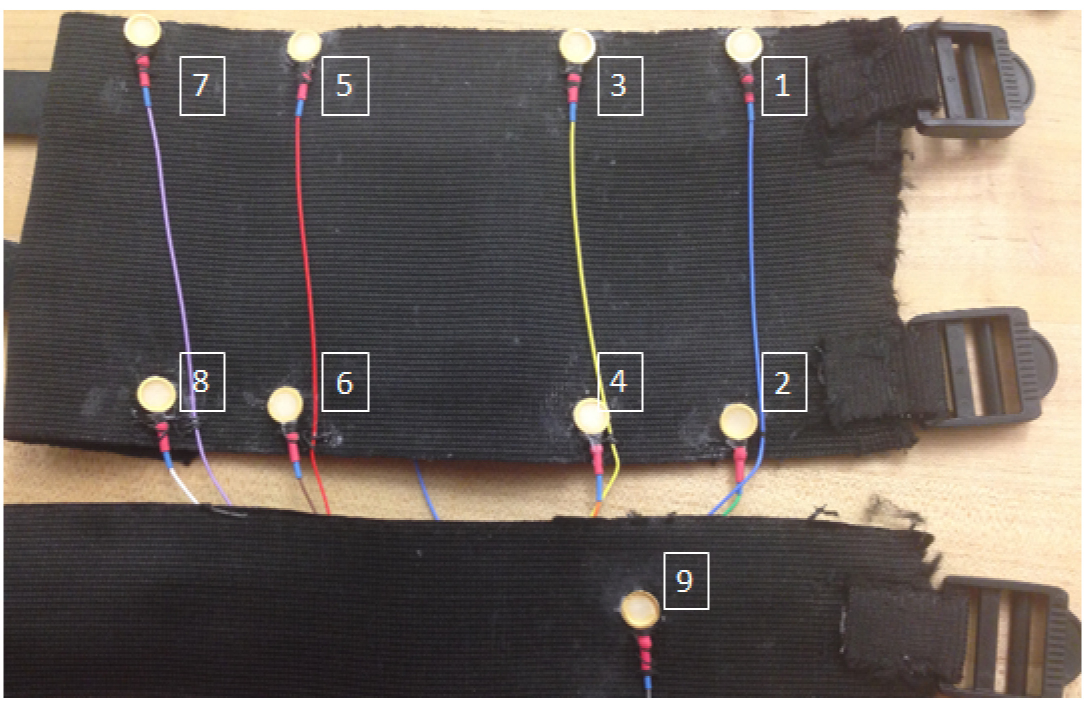

Figure 2: *Surface electrodes used for electromyography signal acquisition*

When placing the eight electrodes on the arm, the third and fourth electrodes, labeled above, are placed on the lateral side of the forearm. This should then place the buckle of the arm band on the medial side of the arm and all other electrodes spread around the arm. The arm band is then tightened until snug.
The ground electrode is then placed on the elbow.

A image of the arm band when placed on the arm can be seen below:

Figure 3: *Armband being worn for electromyography signal acquisition*

The eight electrodes are treated as four pairs and are paired as they go across the different muscle fibers. The four electrode pairs are: 1 & 2, 3 & 4, 5 & 6 and 7 & 8.

The names used to label these electrodes can be seen in the following table:

Electrode Number	Electrode Name
1	Pair 1-A
2	Pair 1-B
3	Pair 2-A
4	Pair 2-B
5	Pair 3-A
6	Pair 3-B
7	Pair 4-A
8	Pair 4-B
9	GND

Table 1: *Arm band electrode numbers to name mapping*

The wires of the nine electrodes are connected to male DB9 connector via screw terminals as shown below:

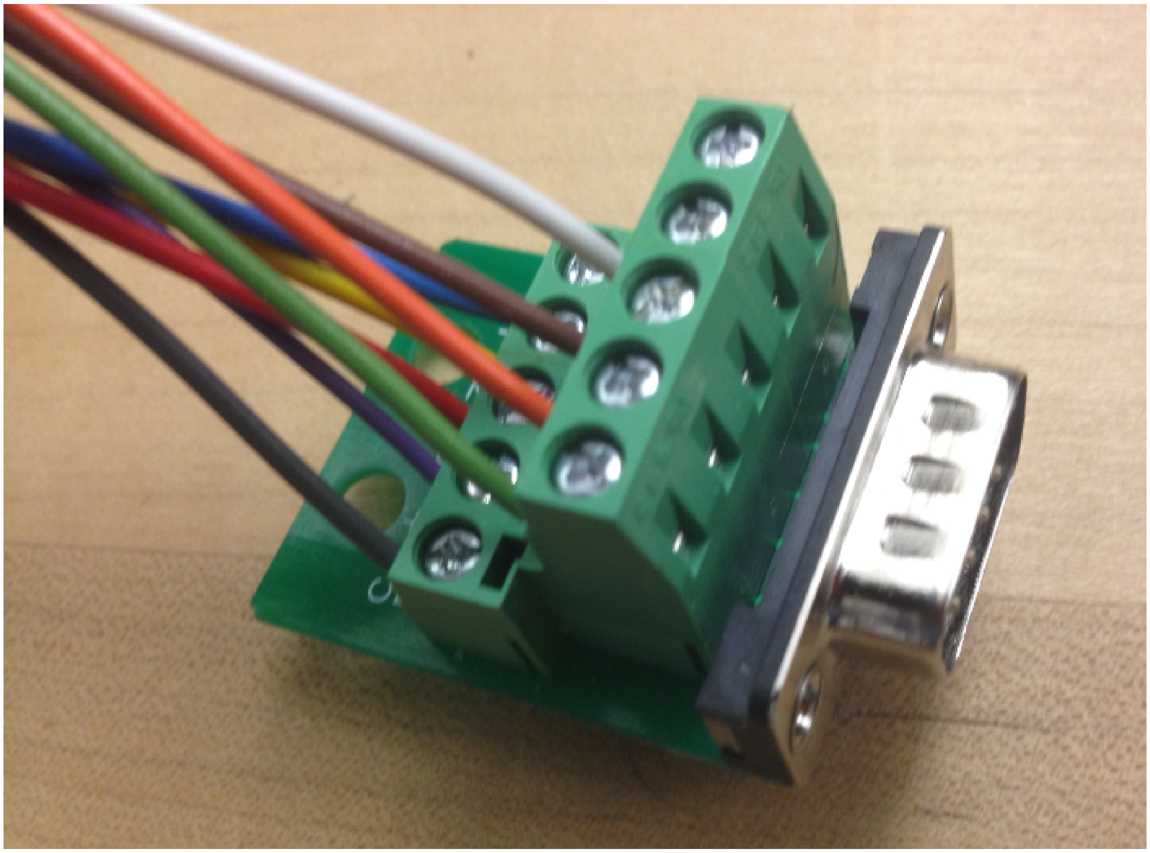

Figure 4: *Male DB9 connector used for EMG electrodes from arm band*

The electrodes from the arm band wire through the DB9 connector following the mapping in the table below:

Electrode Name	Electrode Number	DB9 Terminal Number
Pair 1-A	1	1
Pair 1-B	2	6
Pair 2-A	3	2
Pair 2-B	4	7
Pair 3-A	5	3
Pair 3-B	6	8
Pair 4-A	7	4
Pair 4-B	8	9
GND	9	5

Table 2: *Arm band electrodes to male DB9 connector map*

Each pair of the eight electrodes around the forearm is connected to the inputs of the four AC couple circuits.

The circuit diagram for one of the AC couple circuits used is shown below:

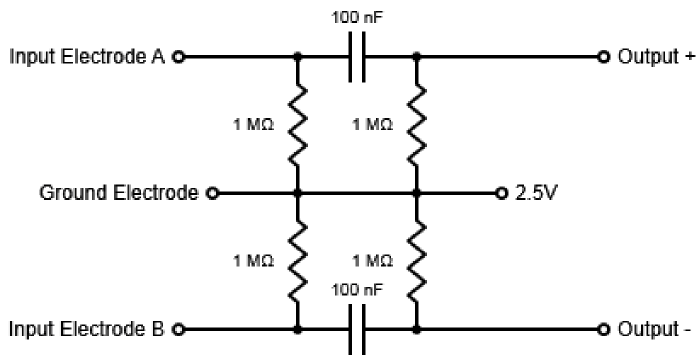

Figure 5: *AC couple circuit diagram*

This is necessary since there is are constant differences in voltage across the electrode pairs that are greater than the difference in voltage of the EMG signals. Along with the gain applied by the instrumentation amplifiers the signals become saturated and the variation across the muscle is no longer detectable. Because of this the AC couple circuit is used to eliminate any constant difference across the electrode pairs so that only more meaningful differences are amplified and clearer EMG signals can be detected. The EMG signals are AC coupled around the signal ground, which is 2.5V, and the ground electrode on the elbow is connected to each of the four AC couple circuits as shown in the diagram above.
These AC couple circuits are implemented and soldered onto a breadboard along with DB9 adapters. A female DB9 connector is used for input signal and the male DB9 connector is used for the output. Both DB9 connectors utilize ribbon wire to connect to the AC couple circuits.

The implementation of this AC couple adapter can be seen below:

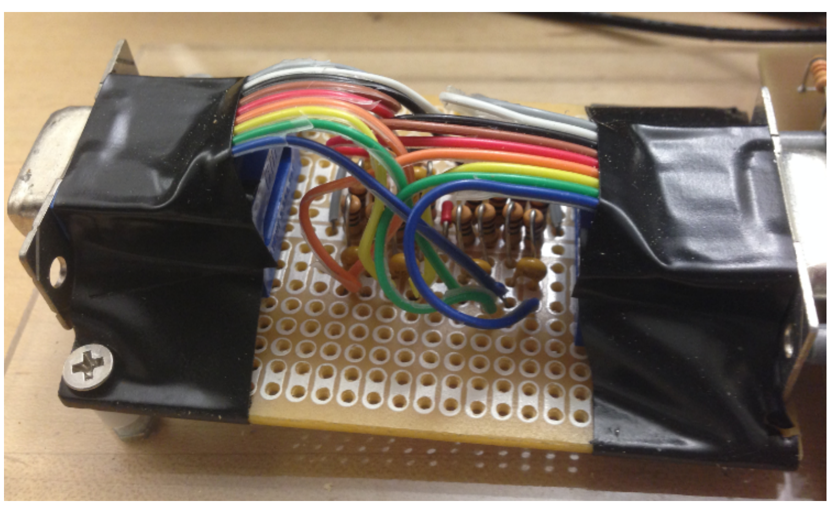

Figure 6: *AC couple circuit adapter implementation*

Note that the electrode wire colors and the ribbon wire colors are matched together. That is, the EMG signals from electrode 1 is carried by a blue wire, as seen seen in Figure 2, is also carried through the blue wires on the ribbon wires. The only exception to this is electrode 7, which uses a purple wire while the ribbon wire carrying that EMG signal is black.
The output of the AC couple adapter circuit is connected to a female DB9 mounted on to a printed circuit board (PCB) designed for this project. This PCB implements four high gain instrumentation amplifiers and four 60Hz, the details of which are described in detail in this section.

The circuit diagram for the PCB can be seen below:

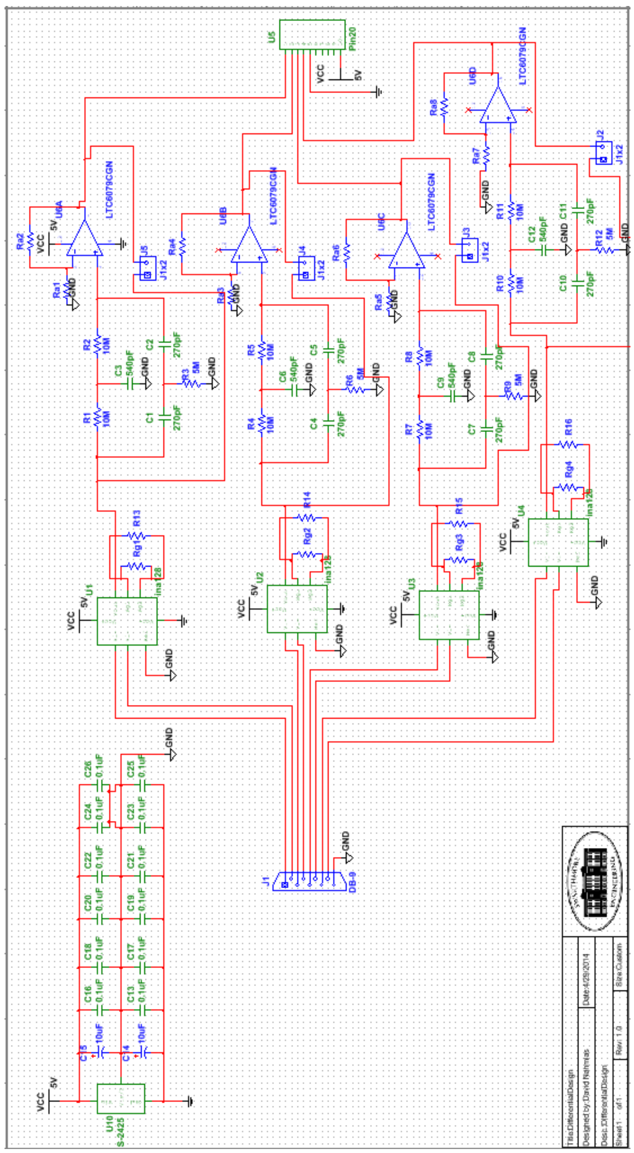

Figure 7: *PCB circuit diagram implementing four instrumentation amplifiers and 60Hz Notch filters*

The above circuit diagram is implemented on a PCB, for which layout can be seen below:

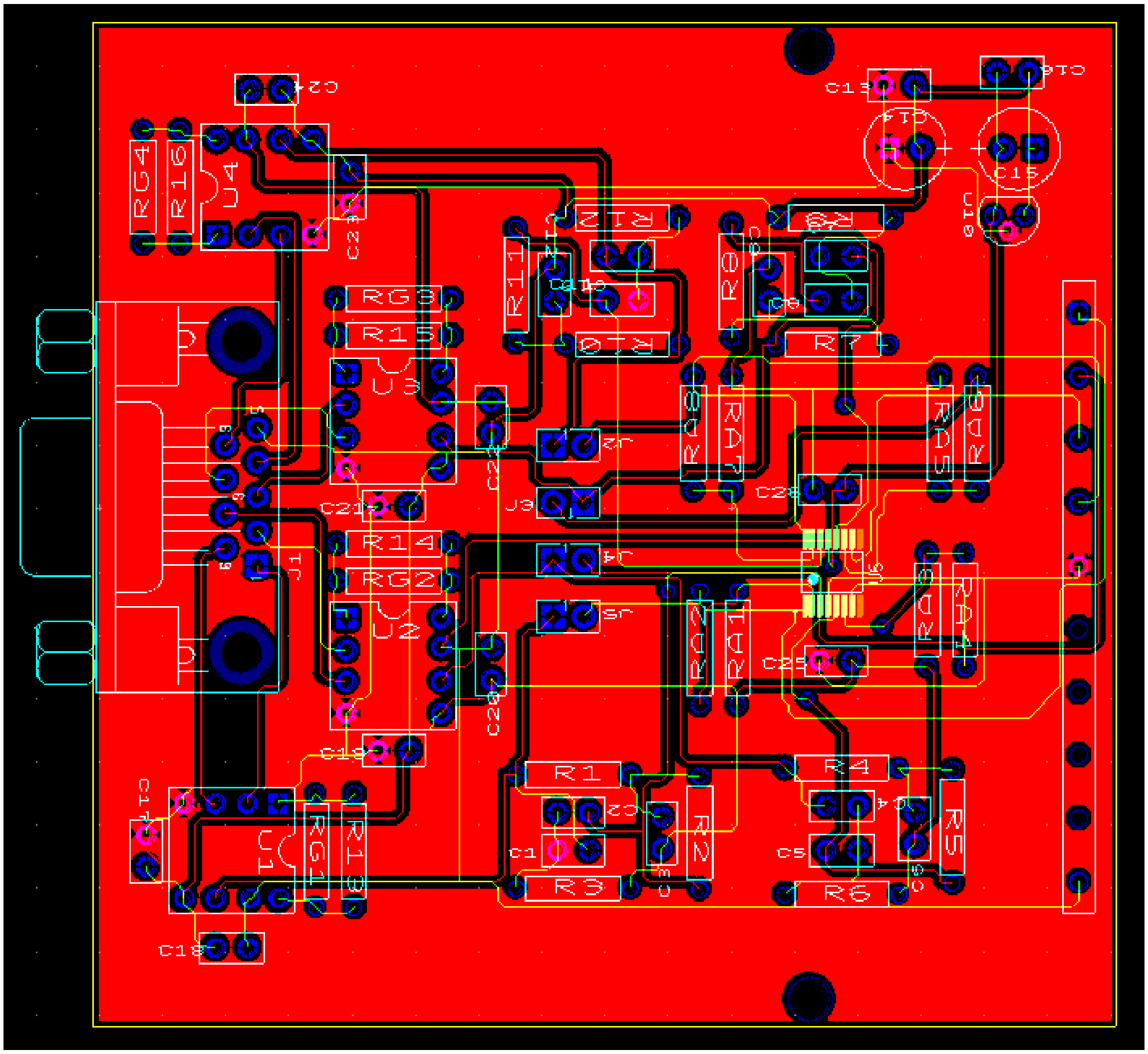

Figure 8: *PCB layout implementing four instrumentation amplifiers and 60Hz Notch filters*

Finally, we can see the PCB populated with all the circuit elements below:

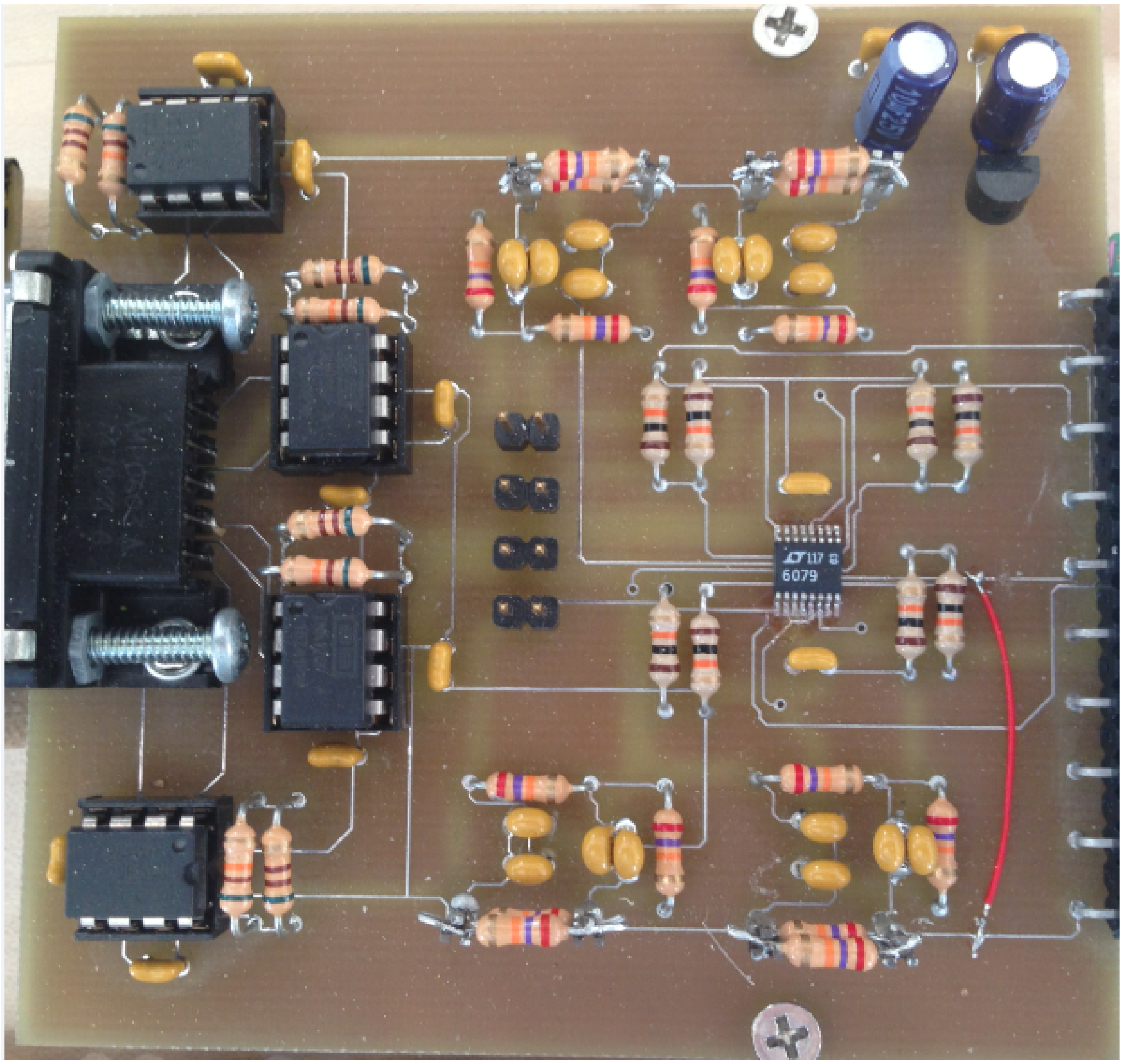

Figure 9: *Populated PCB board populated implementing four instrumentation amplifiers and 60Hz Notch filters*

On the right of the image is the DB9 input from the AC couple circuit and on the is left the filtered EMG signal connected to the screw terminals of the AD7689. The 5V source from the AD7689 is jumped to CH_7 terminal of the AD7689 which powers the PCB. The first GND terminal is used as the 0V on the PCB.

The first part of this PCB implements four INA128-high gain instrumentation amplifiers.
Each of the pairs of outputs of the AC couple circuits are connected to the inputs of the four instrumentation amplifiers.[3] These instrumentation amplifiers have an adjustable gain, G, where the gain equation is given by $G = 1 + \frac{50k\Omega}{R_g}$.

The circuit diagram implemented by the instrumentation amplifier can be seen below:

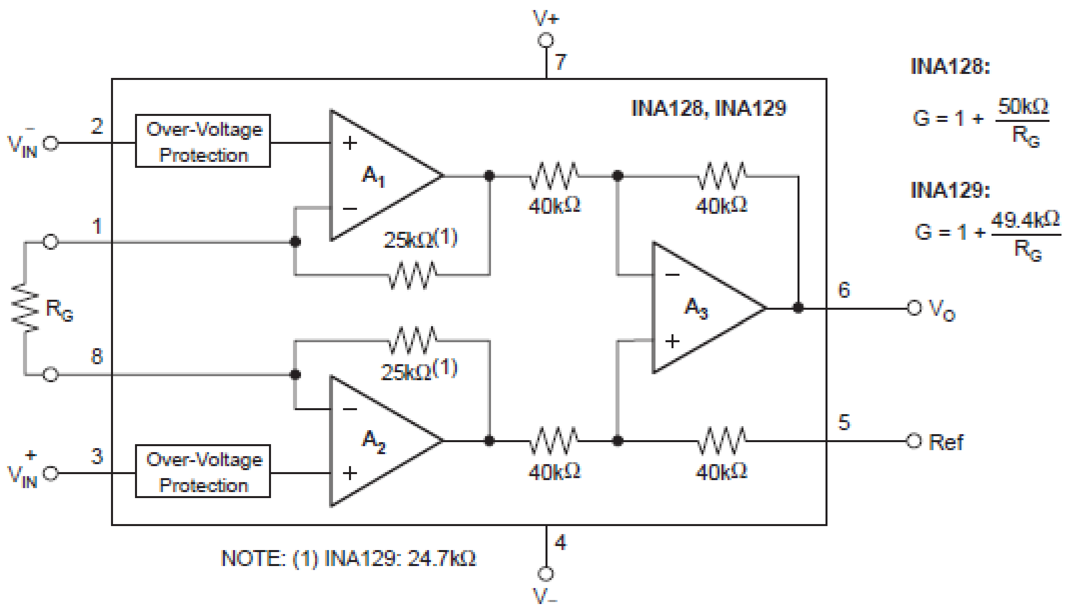

Figure 10: *INA128-Instrumentation amplifier circuit diagram*

The INA128 high gain instrumentation amplifier, with the circuit diagram shown above, follows the pin out schematic shown below:

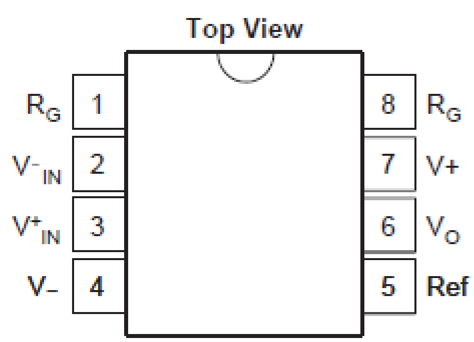

Figure 11: *INA128-Instrumentation amplifier pin schematic*

An R_g of two resistors, 510Ω and 51kΩ in parallel, for an equivalent $R_g = \frac{1}{\frac{1}{510}+\frac{1}{51000}} \approx 504.95$ is used for a $G = 1 + \frac{50k\Omega}{504.95\Omega} \approx 100$.

The second part of this PCB implements four 60Hz Notch filters via an integrated circuit and precision resistors and capacitors.

Each of the outputs, V_O of these four instrumentation amplifiers, is wired through a 60Hz notch filter, which can be seen below:

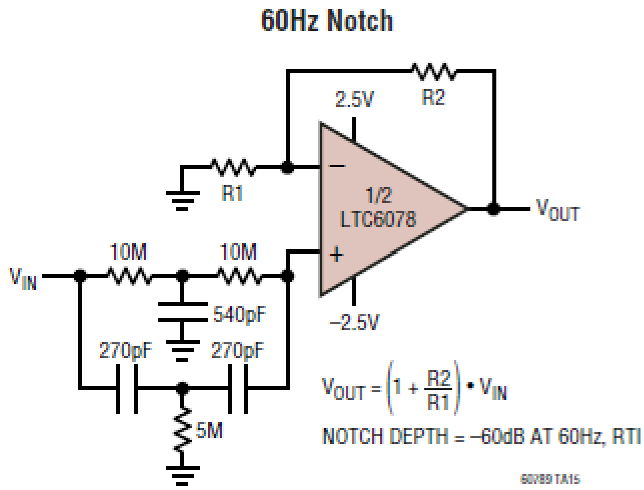

Figure 12: *60Hz Notch filter circuit diagram*

Note that though the capacitor and resistor values labeled above would work in theory, for this project:

- The $10M\Omega$ resistors are replaced with $27k\Omega$ resistors
- The $5M\Omega$ resistores are replaced with two $27k\Omega$ resistors in parallel for an equivalent resistance of $13.5k\Omega$
- The $270pF$ capacitors are replaced with $0.1\mu F$
- The $540pF$ capacitors are replaced with two $0.1\mu F$ in parallel for an equivalent capacitance of $0.2\mu F$

Note that the elements in parallel were unable to be soldered directly onto the printed circuit board and so they were soldered to each other before placed onto the printed circuit board, which can be seen in Figure 9.
Given these values we see that the frequency, f_0, we would notch is:

$$f_0 = \frac{1}{2\pi \times 27k\Omega \times 0.1\mu F} = 58.944Hz,$$

which is equivalent to the values used in the schematic above,

$$f_0 = \frac{1}{2\pi \times 10M\Omega \times 27pF} = 58.944Hz.$$

Finally, a $10k\Omega$ resistor is used for R1 and R2 which results in

$$V_{\text{OUT}} = \left(1 + \frac{10k\Omega}{10k\Omega}\right) \cdot V_{\text{IN}} = 2 \cdot V_{\text{IN}},$$

which produces a gain of 2 in the system.

The four notch filters are implemented on the printed circuit board via the LTC6079CGN, an integrated circuit [6], which can be seen below:

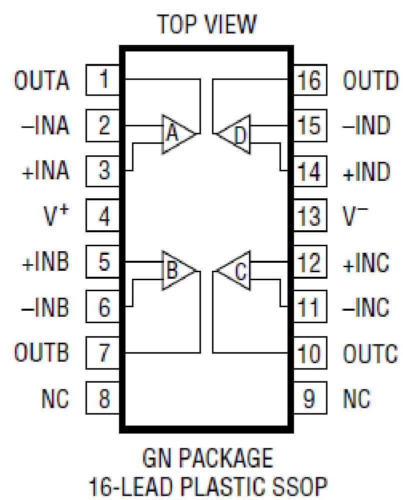

Figure 13: *LTC6079CGN pin Schematic used for 60Hz Notch filter*

Note that on the printed circuit board, the original wiring for the $0V$ and $5V$, both coming from the AD7689 analog front end, were switched for the inputs to the LTC6079CGN. This is corrected by manually altering the printed circuit board, which can be seen in Figure 9 with the red wire jumped across the PCB.

From implementing this 60Hz Notch filter and the previously analyzed instrumentation amplifier, each of the four EMG signals are being fed through a gain of $100 \times 2 = 200$ while being filtered before they are connected to the AD7689. Each of the four V_0 from the notch filters is then wired to CH_0 through CH_3 of the AD7689.

The mapping of the PCB to AD7689 screw terminal can be seen in the following table:

Signal Name	PCB Pin Number	AD7689
EMG Pair 1	1	CH_0
EMG Pair 2	2	CH_1
EMG Pair 3	3	CH_2
EMG Pair 4	4	CH_3
GND (0V)	5	Top/First GND terminal
Not Connected	6	Bottom/Second GND terminal
Not Connected	7	CH_4
Not Connected	8	CH_5
Not Connected	9	CH_6
Power (5V)	10	CH_7

Table 3: *Pin mapping for PCB to AD7689 screw terminals*

The filtered EMG signals are then read from the AD7689 by the BeagleBone black through Serial Port Interface (SPI) protocol.

To implement SPI between the BeagleBone Black and the AD7689 the following pin mapping is used:

Connection Name	BeagleBone	AD7689
GND	P9_1	Pin 5
CLK	P9_22	Pin 4
MOSI	P9_18	Pin 3
MISO	P9_21	Pin 2
CNV	P9_17	Pin 1

Table 4: *Pin out mapping to AD7689 for SPI.*

This mapping is determined by the SPI0 pin out on the BeagleBone [1] and the SPI pin out on the AD7689 in the schematic file. [7]

A complete list of all materials used in this section, including all circuit elements can be seen in Appendix 5.1.

2.2.2 Software Acquisition of EMG Signals

To connect to the BeagleBone Black via terminal the command:

```
ssh -X root@192.168.7.2
```

.
In order to use SPI with the BeagleBone Black instructions online were followed in order to obtain the correct files on the BeagleBone which do not come pre-installed.[12]
For this project SPI0, which uses the file spidev2.0, is used. The bash script, which can be found in Appendix 5.8, is run on start-up, via crontab:

```
@reboot sh startup.sh
```

to re-enable the spidev2.0 file every time.
Once this is enabled it can be accessed in C using the command

```
int fd2 = open("/dev/spidev2.0",O_RDWR);
```

after which the correct SPI procedure is followed in order to communicate with the AD7689.
The read/write after conversion (RAC) mode without busy indicator [7] is used for this project.
The 16-bit configuration register is determined by the following desired modes:

- 1-Overwrite contents of register
- 111-Unipolar, INx reference to GND
- Channels:
  - 000-Channel IN0
  - 001-Channel IN1
  - 010-Channel IN2
  - 011-Channel IN3
- 0-1/4 of bandwidth

- 001-Internal reference, REF=4.096V output, temperature enabled; 00-Disable sequencer
- 1-Do not read back contents of configuration
- 00-The two least significant bits are automatically set to be 0 since only 14 bits are needed for configuration but 16 bits need to be sent.

Thus all together we see:

- 111 0000 0010 0100-Channel IN0
- 111 0010 0010 0100-Channel IN1
- 111 0100 0010 0100-Channel IN2
- 111 0110 0010 0100-Channel IN3

Thus, as seen in the main code the hex codes transmitted for SPI are:

Channel Name on PCB	Channel Name for SPI	Variable Name	Hex Code Transmitted
CH_0	IN0	tx0[1]	0xF024
CH_1	IN1	tx1[1]	0xF224
CH_2	IN2	tx2[1]	0xF424
CH_3	IN3	tx3[1]	0xF264

Table 5: *Hex codes for EMG signal channels for SPI mapping*

Finally, note that the data is sent back after two clock cycles when using RAC mode without busy indicator. Thus in order to obtain the data in the correct order the channels are called in the following order: CH2, CH3, CH0, CH1.
The SPI communication is then executed via standard C SPI protocol using the 'struct spi_ioc_transfer' and 'ioctl' function, which for the first channel is:

```
struct spi_ioc_transfer tr0 = {
.tx_buf = (unsigned long)tx0,
.rx_buf = (signed long)rx0,
.len = 2*ARRAY_SIZE(tx0),
.delay_usecs = delay,
.speed_hz = speed,
.bits_per_word = bit,
};
```

and

```
ret = ioctl(fd2, SPI_IOC_MESSAGE(1), &tr0);
```

The full details for using SPI with the AD7689 can be found in the AD7689 datasheet.[7]
Since the SPI for each of the four channels is done serially, in order to acquire the data from four channels 128 times $4 \times 128 = 512$ SPI transfer calls are done for every hand gesture. This data is stored in a 4x128 array until this all 512 SPI transfers are completed after which the data is ready for processing.
The implementation of this code can be seen in the data acquisition thread named 'SPIdata_thread' of the main code in Appendix 5.2 and training code in Appendix 5.3.

2.3 Analysis of EMG Signals

2.3.1 EMG Software Architecture

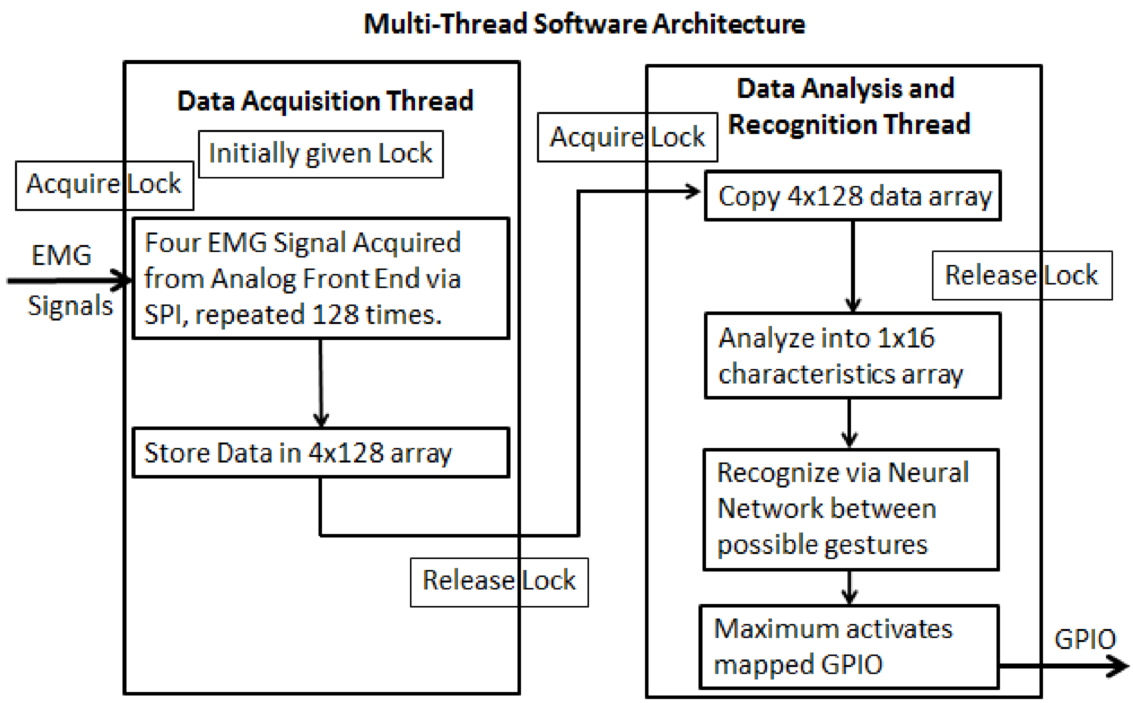

Figure 14: *EMG software architecture diagram*

In order for there to be very little to no lapse in EMG data a multi-threaded program is implemented. When the program is first run, the data acquisition thread acquires the first 128 readings of the four channels and then transfers them over to the main thread using a mutex lock to ensure the data will not be corrupted as the threads share this resource. Once the transfer is complete, main thread begins processing the first set of data while the data acquisition thread collects the second set of 128 readings. Thus, the multi-thread process allows for the analysis of the previously collected data while data is collected for the future analysis. The mutex lock is released by C the command

```
pthread_mutex_unlock(&(p->lock1));
```

and it can then be requested and then acquired, when available and released by the other thread, by the C command

```
pthread_mutex_lock(&(p->lock1));
```

Both the main code in Appendix 5.2 and training code in Appendix 5.3 implement this design. A third independent, not diagrammed here, thread is further implemented to acquire and process the data from the accelerometer and gyroscope independently from the EMG analysis and is described in Section 2.5.

Three external files are used in addition to the main, and are included via `#include`,

which are used for GPIO memory mapping, in Appendix 5.7, a library of auxiliary and characteristic evaluation functions, in Appendix 5.4 and the neural network evaluation function, in Appendix 5.5.
Finally, this code is written with processing speed and simplicity in mind. Thus, the fewest possible inclusion of external libraries are used. Only standard C libraries, already pre-installed on the linux distribution, are used in the code with no external third party libraries downloaded.

2.3.2 EMG Signal Characteristics

The EMG signals are analyzed one channel, or 1x128 long array of data, at a time. Each channel is attributed four characteristics show below. These calculations together produce an array of 16 values which represent four characteristics from the four channels thus reducing the dimensionality of the data from 512 to 16.
The mean of the EMG signal subtracted from the signal so that its overall amplitude is not considered but rather the amplitude of its changes around the mean. Thus the vectors being analyzed are $\vec{X}_i = \vec{X}_i - mean(\vec{X})$ with $i = 0, 1, \ldots, N$.
Thus from this point on $\vec{X}$ refers to the data vector with mean zero. Furthermore, N refers to the number of elements in the vector which in this case is 128.

- Variance of High and Low Frequencies.
  A moving average Finite Impulse Response filter designed with the Parks-McClellan algorithm is used.
  - Using in MATLAB firpm(6,[0 0.4 0.6 1],[0 0 1 1]),
    $Filter_{high} = [0.1195, -0.0001, -0.3133, 0.4998, -0.3133, -0.0001, 0.1195]$
  - Using in MATLAB firpm(6,[0 0.4 0.6 1],[1 1 0 0]),
    $Filter_{low} = [-0.1195, 0.0001, 0.3133, 0.5002, 0.3133, 0.0001, -0.1195]$

  Variance of High and Low Frequencies: $Var(\vec{X}_{Filtered})$ where $\vec{X}_{Filtered_i} = \sum_{j=0}^{6} x_{i+j} \times Filter_{high/low_j}$ with $i = 0, 1, \ldots, N-7$ and $Var(\vec{X}) = \sum_{i=0}^{N-1} (x_i - \bar{x})^2$, where $\bar{x} = mean(\vec{X})$
- Number of Slope Changes: Number of times $|diff(sign(diff(\vec{X})))| = 2$, where $diff(\vec{X}) = x_i - x_{i+1}$ with $i = 0, 1, \ldots, N-2$
- Number of Zero (or Mean) Crossings: Number of times $\vec{X}_i \times \vec{X}_{i+1} \leq 0$ (only counting once for series of zeros) with $i = 0, 1, \ldots, N-1$.

These operations are called in the main function once the data is transferred over but are implemented in an auxiliary code, classificationFunctions.c, which can be seen in Appendix 5.4

These characteristics are useful for looking at the EMG signals because the visible variation that occurs between different hand gestures such as an open versus a closed hand can be seen below:

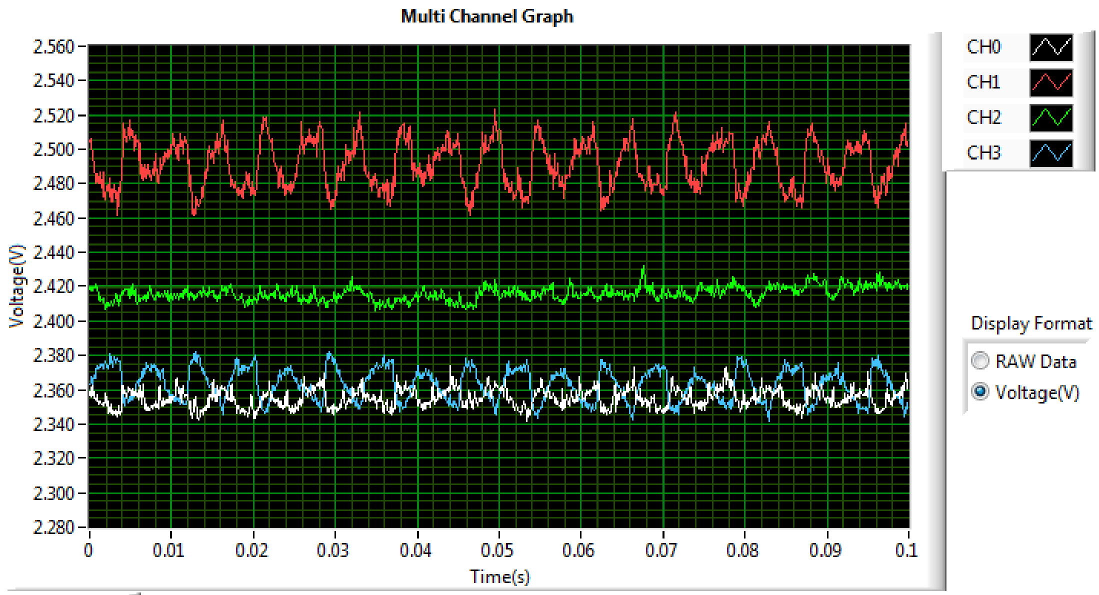

Figure 15: *Four Channel signal when hand is open and relaxed*

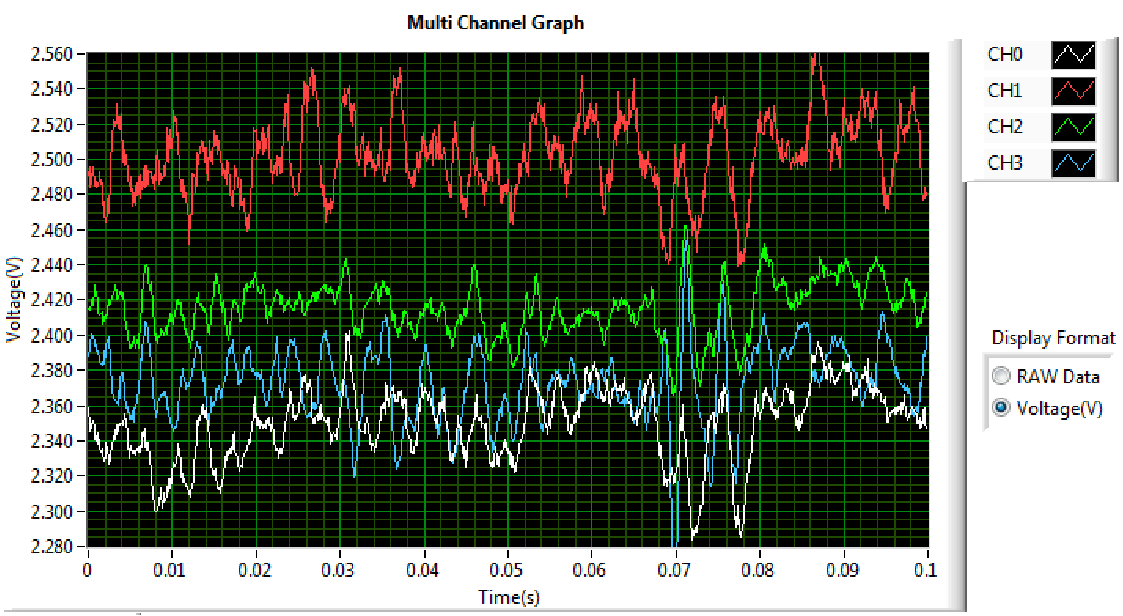

Figure 16: *Four Channel signal when hand is closed and muscles are contracted*

2.4 Training and Application of Artificial Neural Networks

Because of the need to evaluate the data through an Artificial Neural Network (ANN) in C several option were considered and the Multiple Back Propagation software was decided upon. Multiple Back Propagation is used because of its ability to produce a C function which evaluates a new input on the trained ANN using only the math.h library which is already available on the BeagleBone Black board. Currently Multiple Back Propagation is supported for Windows OS and more information about the software can be found online. [17]

2.4.1 Motivation and Theory of Artificial Neural Networks

ANNs are a machine learning technique which are commonly used as a method of pattern recognition. Specifically, they are trained using a supervised learning algorithm. This means that the training set used to initialize the ANN has the corresponding outputs known for the ANN to compare its outputs to. The ANN trains until it has minimized the root mean squared error (RMSE) between its outputs and the known outputs of the training set.

A single neuron in an ANN follows the structure shown below:

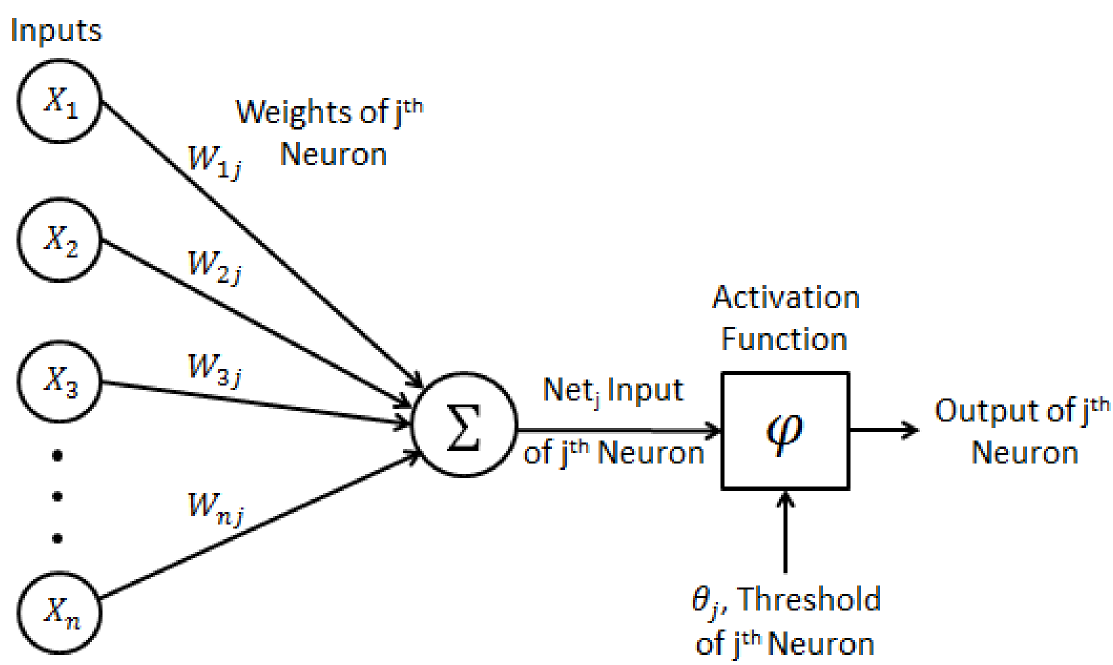

Figure 17: *Diagram of neurons used in applied ANN*

The inputs to the activation function, Net_j is defined as

$$Net_j = \sum_{i=1}^{n} Input_i \times w_{ij}$$

for the j^{th} neuron.

As an example, for simplicity, the following single input neuron is considered below:

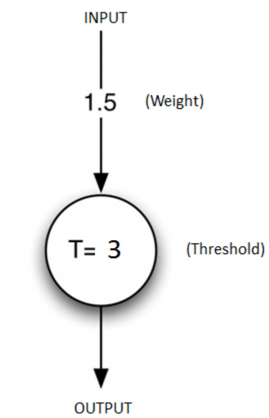

Figure 18: *Single neuron example with weight=1.5 and threshold=3 from an ANN*

In this case, as shown in Figure 17, the input is multiplied by the weights of the neuron and then all inputs$\times$weights of the single neuron are summed together. If that value exceeds the value of the threshold then the neuron is activated and the output is 1. Otherwise, if the sum of all the inputs$\times$weights is less than the threshold then the neuron is not activated and the output is 0. The threshold used here is 3 and thus an input>2 would be needed in order to activate this neuron.

The activation function used in this example is a step function, which produces the binary 0 or 1 outputs and can be seen below:

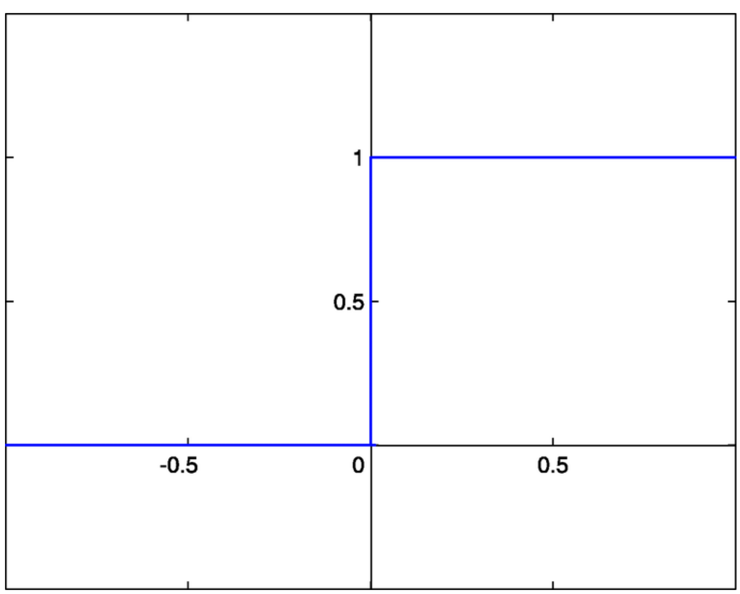

Figure 19: *Step activation function used in simple neuron*

The step function is defined by:

$$\varphi_{step}(t) = \begin{cases} 0 & : t < 0 \\ 1 & : t \geq 0 \end{cases}$$

with t the Net_j input.

ANNs using these types of neurons can be applied to simple logic problems as seen below:

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

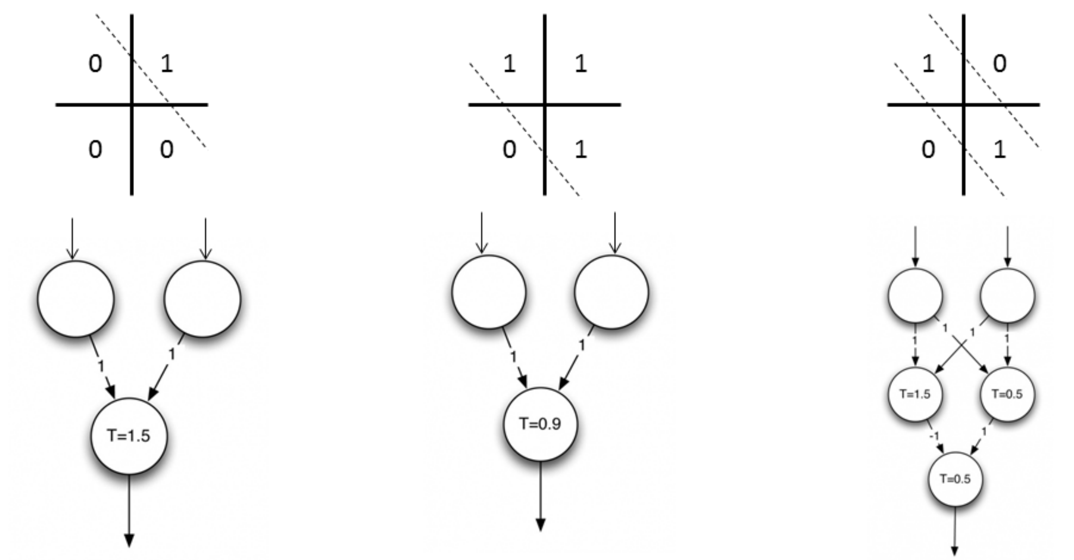

Figure 20: *Simple ANNs applied for solving logic problems, noting linear separability*

AND and OR are both simple and can be solve using a single layered ANN since the problem's solutions are linearly separable. However, XOR, exclusive OR, is not linearly separable and thus requires a hidden layer with two hidden neurons to solve the problem. For this reason the topology of the ANNs used are multi-layered and use a hidden layer with many hidden neurons.

Furthermore, unlike the neurons used in these examples that use a step function as their activation functions, the neurons used in the ANNs applied in this project utilize the Sigmoid function as their activation functions, shown below:

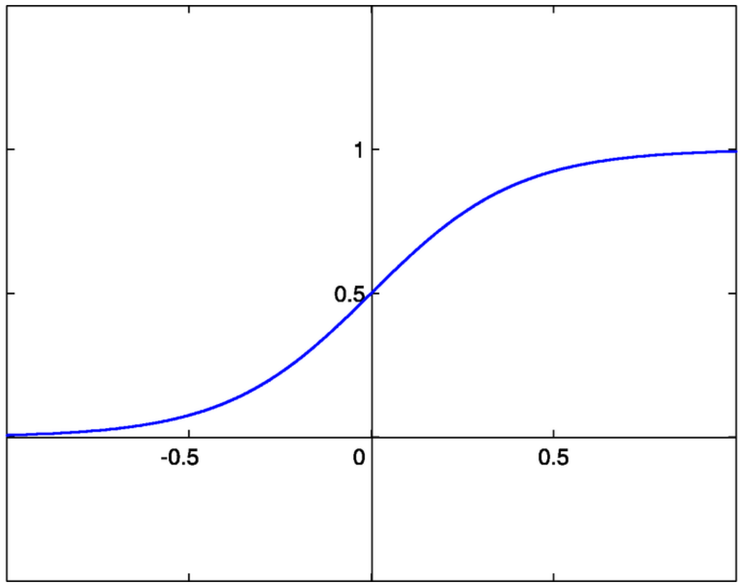

Figure 21: *Sigmoid activation function used*

This function is defined by:

$$\varphi_{Sigmoid}(\beta, t) = \frac{1}{1 + e^{-\beta t}}$$

with β as the slope parameter and t the Net_j input.
This function is widely used when implementing the back propagation algorithm because it is easy to differentiate since
when $\beta \neq 1$,

$$\varphi(\beta, t) = \frac{1}{1 + e^{-\beta t}}$$

and so

$$\frac{\mathrm{d}}{\mathrm{d}t}\varphi(\beta, t) = \beta[\varphi(\beta, t)[1 - \varphi(\beta, t)]]$$

but when $\beta = 1$, which is more commonly used,

$$\varphi(t) = \frac{1}{1 + e^{-t}}$$

and so

$$\frac{\mathrm{d}}{\mathrm{d}t}\varphi(t) = \varphi(t)[1 - \varphi(t)]$$

This activation function produces outputs ranging from 0 to 1, showing the confidence of the ANNs evaluation of the inputs with 1 implying that the ANN strongly believes that output to be the correct one. Finally, with these easily computable derivatives, using the Sigmoid function dramatically reduces the computation involved while training since the derivative of the activation is used in the back propagation training algorithm.

The training process for ANNs is rather complicated, however, the main method of training used in this project is an algorithm called back propagation, where the name of the software comes from. This algorithm initially randomizes the weights and thresholds of the ANN. The algorithm then updates the weights and thresholds of the ANN depending on the error between the output of the ANN and the known outputs of the training set. The goal is to weigh and threshold the inputs of the training set to evaluate the correct outputs, as compared to the training set. This process is done iteratively, with each iteration known as Epochs, until the desired minimum RMSE level or maximum allowable Epochs is reached.

A flow of the Back Propagation algorithm is diagrammed below:

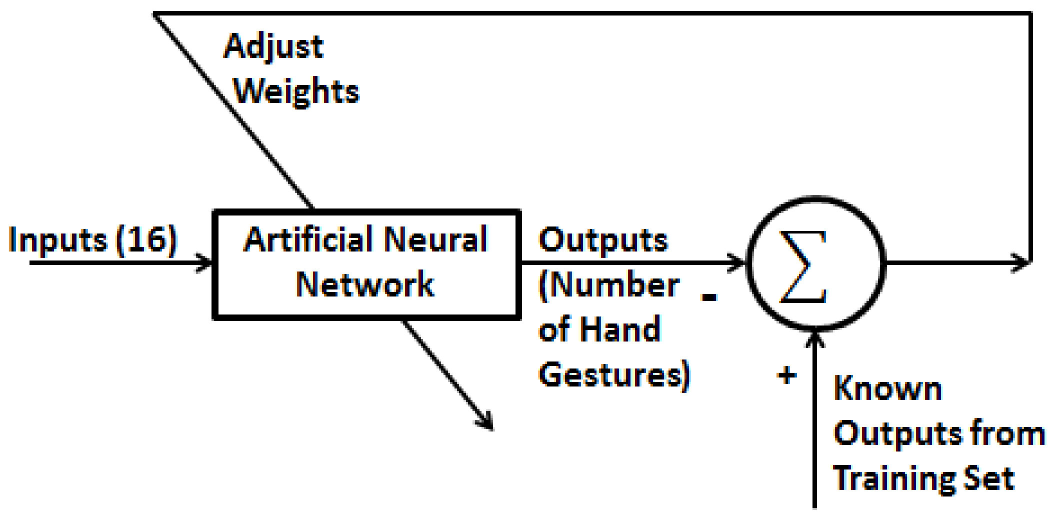

Figure 22: *Multiple Back Propagation algorithm diagram*

The properties and theory of the ANNs application and training algorithms are further described by the Multiple Back Propagation program's underlying papers and tutorial.[17]

2.4.2 Training Artificial Neural Networks for EMG Recognition

Before the ANN can be trained the training set data is acquired. Through the 'trainNNGesture.c' code, the number of desired hand gestures to be recognized is edited. Once this is done the code is compiled and run.
The training set is then collected by acquiring the EMG data, processing it and then storing it in .txt files in the data sub-folder, which is created prior to running this program. 500 sets of 16 characteristics for each of four hand gestures is collected and stored in .txt files. Furthermore, in the case of four hand gestures, for each set of 16 characteristics, the known output of the data is stored in the form of three 0s and a 1, with the 1 representing the correct hand gesture. For example each of the 500 sets of 16 characteristics for the first hand gesture would be followed by '1 0 0 0.' Each hand gesture is to be held for approximately 1.5 minutes as the data is accumulated after which there is a three second pause when the next hand gesture must be made and held for approximately 1.5 minutes.

Once the program is done, a single .dat file, of all hand gesture data is created by running in the command line:

```
cat *.txt > allData.dat
```

within the data sub-folder.
The hand gesture data is initially stored in different files for debugging and analysis purposes.
This .dat file can then be loaded into the Multiple Back Propagation program for training.
The topology of the Artificial Neural Network is given by 16-HN-4, where HN is the number of hidden neurons in the hidden layer and is open for testing. The first number is the number of input neurons, which are the four characteristics for each of the four channels, 16 and the last number is the number of hands gestures being recognized,4. The HN value must make it so that the ANN is able to train and later recognize new data. An HN value that is too low risks not being able to reach the required accuracy of the ANN output while an HN value too high may over-train the ANN and may only recognize accurately data from the training set and not new data which will vary slightly from the training set. For this project, through testing, it is determined that $HN = 20$ is the fewest number of hidden neurons required to reliably produce the desired accuracy.

The topology of such an ANN is shown below:

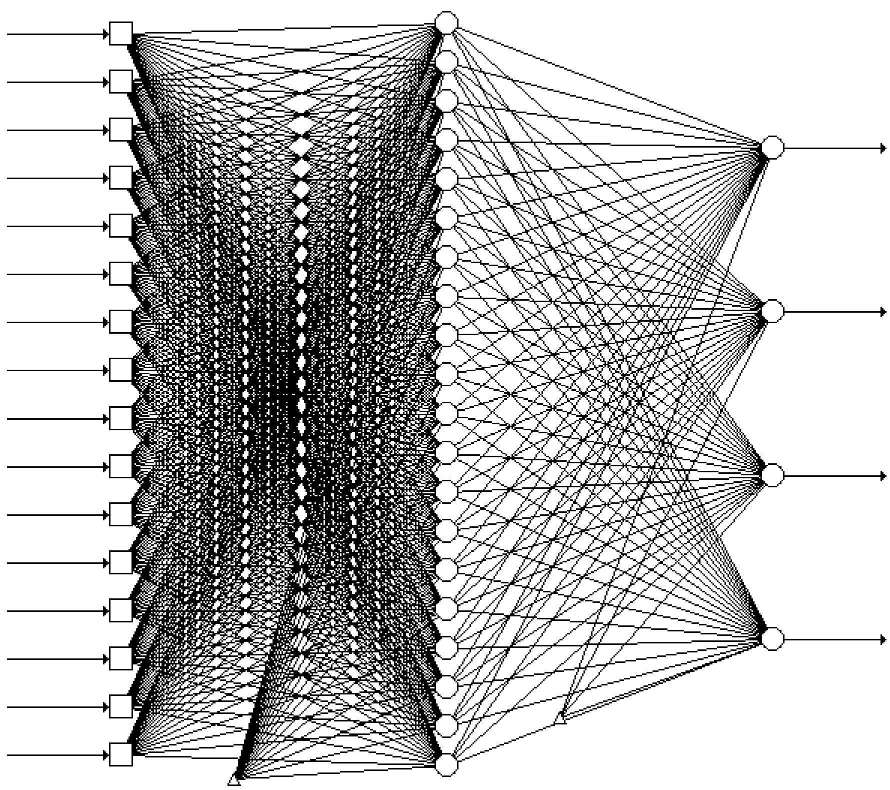

Figure 23: *ANN topology for 16 inputs, 20 hidden neurons, and 4 outputs*

In the figure above, the square nodes represent the inputs, the circle nodes each represents the single neuron diagram seen in Figure 17 and the small triangles are biases that are used when updating and evaluating the ANN to vary the input values randomly.

Once the data is collected and placed into the .dat file, it is loaded into the Multiple Back Propagation software and trained after which the C code can be downloaded as 'MBPnnEval.c,' which can be seen in Appendix 5.5. This file is then copied into the working directory of the main run file.

2.4.3 Applying Artificial Neural Networks for EMG signal Recognition

Data of similar size and characteristics that is used to initially train the ANN must be applied when using the ANN in order to produce accurate results. When implementing the trained ANN the same features will be extracted from the 'live' signals and these features will be run through the ANN which will identify which motion the features came from.
In applying the ANN code from Multiple Back Propagation, the only things necessary are an input vector of the same type of 16 classifiers used to train the Artificial Neural Network and a pre-allocated array with length depending on number of hand gestures being recognized. Multiple Back Propagation's code takes care of the rest and applies the weights of each neuron in the network thus computing probabilities for each hand gesture, with values ranging from 0 to 1. The highest valued element in the array is the hand gesture believed to be executed.

The topology and sample output of the ANN used to recognize the four hand gestures can be seen below:

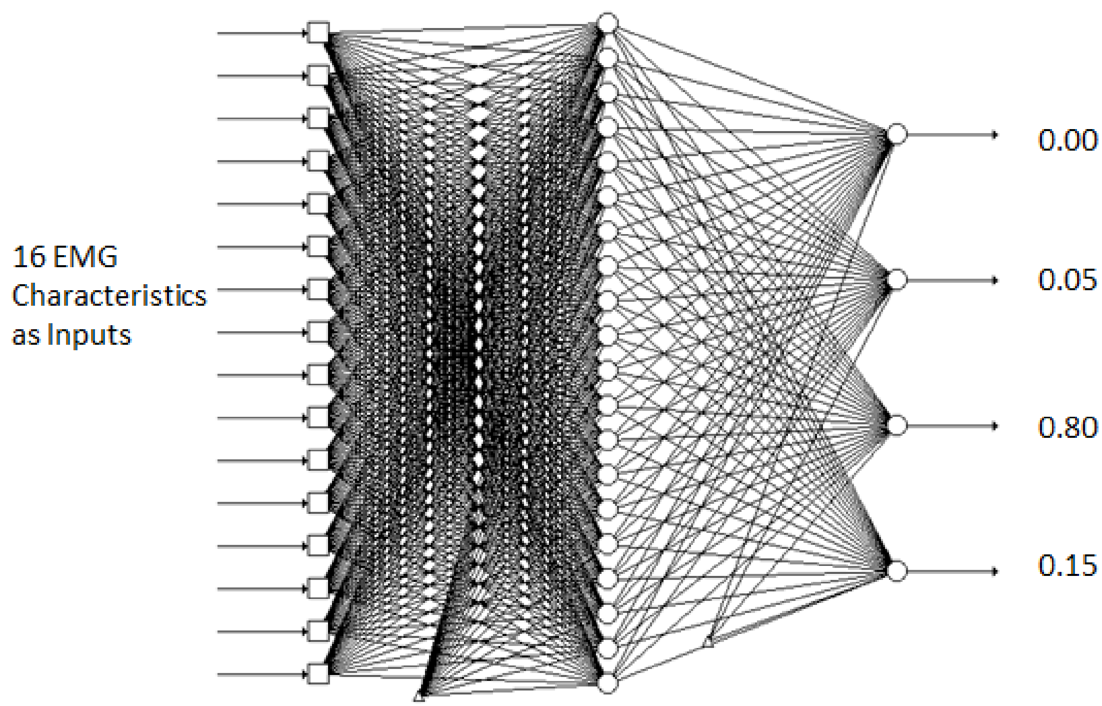

Figure 24: *ANN topology applied for initial four gestures with sample outputs*

With these sample outputs, the ANN is indicating that the third hand gesture is being executed.

2.5 Integration of Accelerometer and Gyroscope

Given the application of this project, only the accelerometer data is acquired and analyzed. To integrate the accelerometer and gyroscope through I^2C a third independent thread within the main code is used.

The software architecture diagram with the additional thread can be seen below:

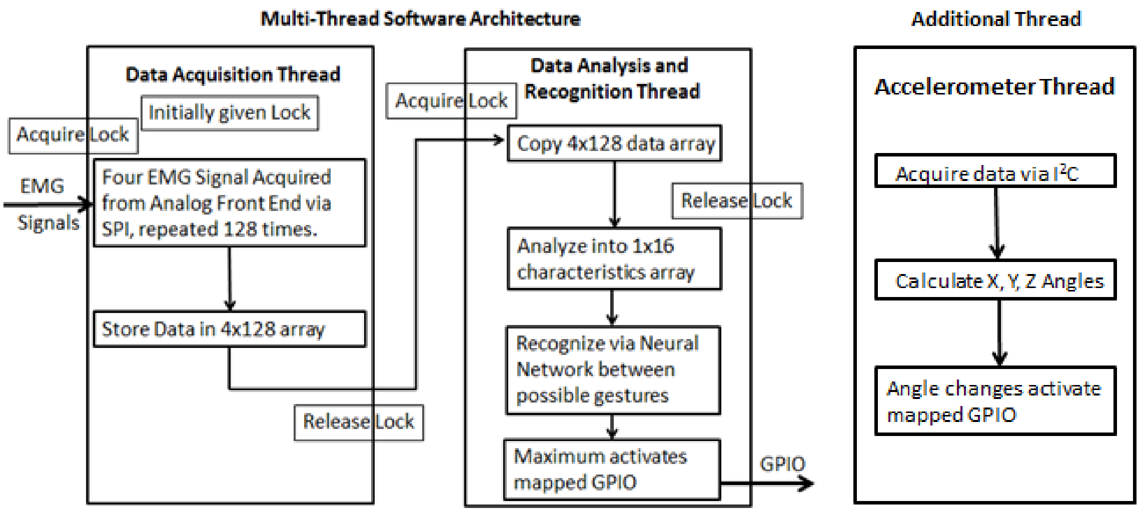

Figure 25: *Software integration diagram of integration for accelerometer*

This third independent thread is used so to not interfere with the already existing EMG data acquisition and analysis software timing. The i2c-1 port is used on the beaglebone for this I^2C protocol which is done in C by using the commands:

```
char *filename = "/dev/i2c-1";
file = open(filename, O_RDWR);
```

Furthermore, on the beaglebone, the MPU6050 when connected via I^2C is at the address 0x68, determined via the terminal shell command:

`i2cdetect` [13]

which is subsequently set in C by the command:

```
unsigned char addr = 0x68;
```

To implement I^2C between the BeagleBone Black and the MPU6050 the following pin mapping is used:

Connection Name	BeagleBone	MPU6050
VDD(3.3V)	P9_3	VDD(Pin 1)
GND	P9_2	GND(Pin 2)
SCL	PIN19-9	SCL(Pin 5)
SDA	PIN20-9	SDA(Pin 6)
VIO(3.3V)	PIN4-9	VIO(Pin 7)

Table 6: *Pin out mapping to MPU6050 for I^2C*

This mapping is determined by the I^2C pin out on the BeagleBone [1] and the pin out labeled directly on the MPU6050 and can also be found in the datasheet [10].
From the register mapping [8] the board is set ON by setting the 0x6B bit low, to 0x00, which is done in C by the following commands:

```
ioctl(file,I2C_SLAVE, addr);
wBufOn[0] = 0x6b;
wBufOn[1] = 0x00;
write(file,wBufOn,2);
```

Other than this register, all other default setting are kept the same and no other registers are set to any values via software.
The register addresses for the accelerometer data are found on the 0x3B to 0x40 registers.

The following table from the MPU6050 register map shows the accelerometer data registers:

Term	Hex Register Map on MPU6050
ACCEL_XOUT[15:8]	0x3B
ACCEL_XOUT[7:0]	0x3C
ACCEL_YOUT[15:8]	0x3D
ACCEL_YOUT[7:0]	0x3E
ACCEL_ZOUT[15:8]	0x3F
ACCEL_ZOUT[7:0]	0x40

Table 7: *Accelerometer raw data register map in hex code on the MPU6050*

The register to be read is initially set to 0x3B, the eight most significant bits of ACCEL_XOUT. The I^2C protocol can then be run in burst mode, reading the next five sequential registers without having to reset the register to be read.
This is done through the following commands in C:

```
ioctl(file,I2C_SLAVE, addr);
wBuf[0] = 0x3b;
write(file,wBuf,1);
ioctl(file,I2C_SLAVE, addr);
```

Note that each register is only contains 8 bits though each of the accelerometer values are 16 bits long. Thus, when reading the accelerometer data, the [15:8] bits are left shifted by 8 bits and then added to the [7:0] bits of the same accelerometer axis.
This is done, for example for the x-axis acceleration, in C, through the following commands:

```
read(file,rBuf,1);
accel_x = rBuf[0];
read(file,rBuf,1);
accel_x = (accel_x«8)+rBuf[0];
```

In order to interpret the values from the MPU6050, which is returning raw data, as angles we let the raw 16 bit x, y, z data be defined as A_x, A_y, A_z respectively and calculate the angles using the following equations:

$$\rho = \arctan\left(\frac{A_x}{\sqrt{A_y^2 + A_z^2}}\right)$$

$$\phi = \arctan\left(\frac{A_y}{\sqrt{A_x^2 + A_z^2}}\right)$$

$$\theta = \arctan\left(\frac{\sqrt{A_x^2 + A_y^2}}{A_z}\right)$$

with ρ, the Pitch, is defined as the angle of the X-axis relative to ground, ϕ, the Roll, is defined as the angle of the Y-axis relative to ground, and θ, not quite the Yaw, is defined as the angle of the Z-axis relative to the ground, because of the effect of gravity.

2.6 Integration of Human Device Interface

After characterizing and classifying the EMG signal via the ANN, General Purpose In/Out (GPIO) pins are used to control the HDI. In this project the HDI being controlled is a computer mouse and arrow keys and space bar on a keyboard. This is done by activating, setting high, the appropriate GPIOs as a specific hand movement or gesture is executed. The GPIO signals are then sent to the Makey-Makey board.

The Makey-Makey board, when connected via USB to the computer, controls the computer mouse and certain set keys of the keyboard, including the arrow keys and space bar, as the appropriate pins are activated. Pins on the Makey-Makey are activated when a change in voltage, relative to a common ground, is detected. Thus, when the appropriate hand gesture is recognized or hand movement from the accelerometer and gyroscope is executed a set GPIO pin can be activated thus executing a computer command via the Makey-Makey.

The GPIO pins are mapped via memory location for speed and reliability. The starting address for Mode 1 for all pins on the BeagleBone Black's pin header is 0x4804C000. The GPIO pins used are then left shifted by different amounts as directed on the pin out mapping of the BeagleBone Black.

The GPIO mapping for the BeagleBone Black can be seen in the table below:

GPIO Location on Pin Header	GPIO Name	Left Shift Memory Location
P8_11	GPIO1_13	13
P8_12	GPIO1_12	12
P8_15	GPIO1_15	15
P8_16	GPIO1_14	14
P8_26	GPIO1_29	29
P9_12	GPIO1_28	28
P9_14	GPIO1_18	18
P9_15	GPIO1_16	16
P9_16	GPIO1_19	19
P9_23	GPIO1_23	23

Table 8: *GPIO memory mapping for the BeagleBone Black as implemented in C*

These GPIO memory locations are initialized in 'beaglebone_gpio.h', which can be seen in Appendix 5.7 and implemented in the main code which can be seen in Appendix 5.2.

Finally, note that in order to implement these GPIOs onto the Makey-Makey, P8_1, an available GND pin on the BeagleBone Black board, must be connected to the ground plane of the Makey-Makey. For more information on the Makey-Makey see the website and documentation.[16]

3. Results

The system is implemented analyzing the following four hand gestures shown below:

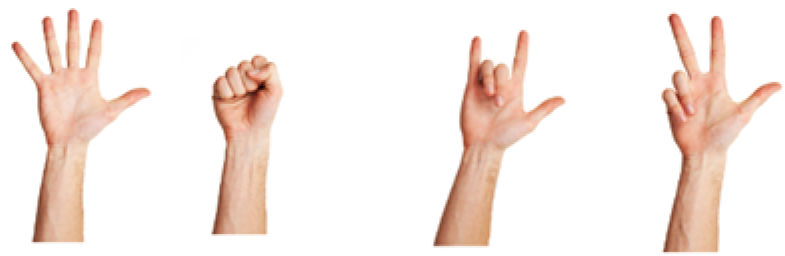

Figure 26: *Initial set of four hand gestures analyzed for recognition*

The hand gestures above, from left to right are referred to as the first hand gesture or open hand, the second hand gesture or closed hand, the third hand gesture or spiderman hand and the fourth hand gesture or three hand.

These hand gestures are chosen since they provide a good spectrum of some hand gestures being vastly different from one another, such as the open and closed hands, while some hand gestures are similar to one another, such as the spiderman hand and the three hand. Thus, it is possible to see how well the system does in recognizing and differentiating hand gestures that are vastly different versus hand gestures that are similar in their muscle activation.

3.1 Results with Four Hand Gestures

For the training set acquired via the four hand gestures show above and using the ANN topology shown in Figure 24.

The plot below shows the RMSE as it varies with the number of Epochs performed:

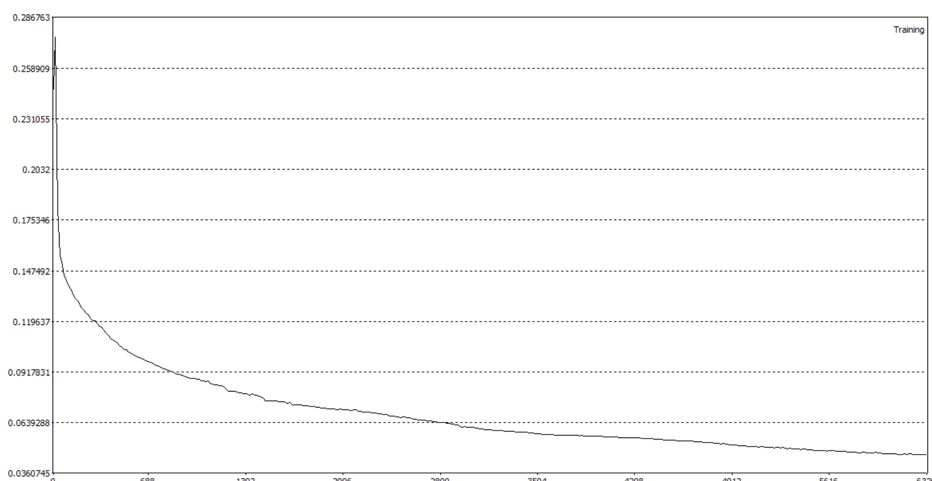

Figure 27: *RMSE of the ANN after 6335 Epochs over approximately 10 minutes*

From this test, the minimum obtained RMSE≈ 0.046, from 6335 Epochs, which takes approximately 10 minutes using the Multiple Back Propagation software. The C file produced by this test can be seen in Appendix 5.5.

For the four hand gestures, the output and accuracy of this ANN can be seen below in the four plots. These plots indicate in black the known outputs of the training set and in red the believed outputs of the ANN.

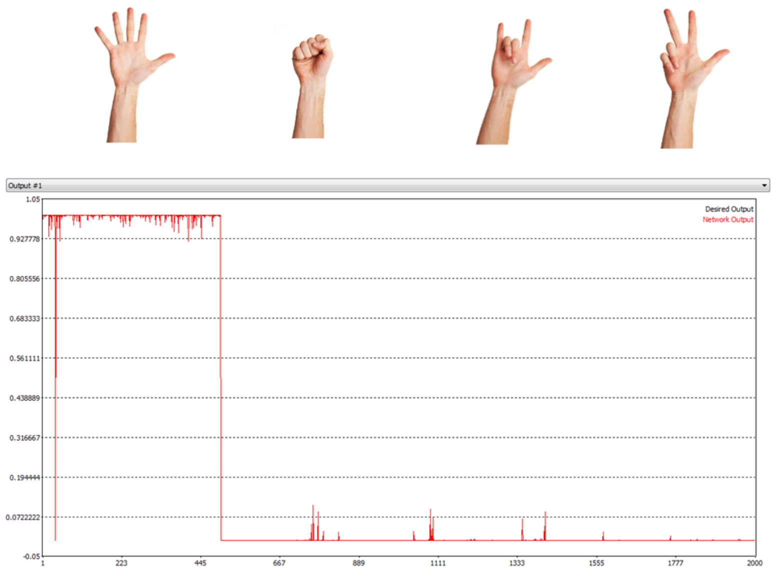

Figure 28: *ANN output for initial set of four hand gestures analyzed for recognition-First hand gesture: Open hand*

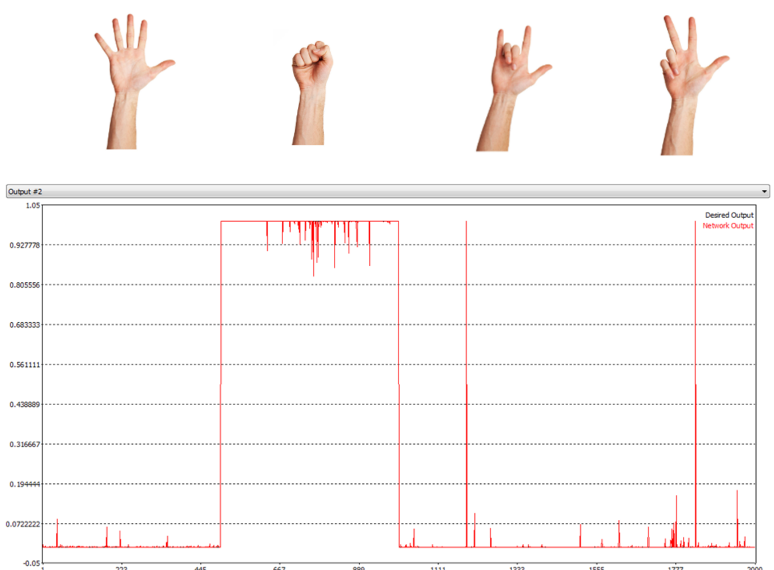

Figure 29: *ANN output for initial set of four hand gestures analyzed for recognition-Second hand gesture: Closed hand*

Figure 30: *ANN output for initial set of four hand gestures analyzed for recognition-Third hand gesture: Spiderman hand*

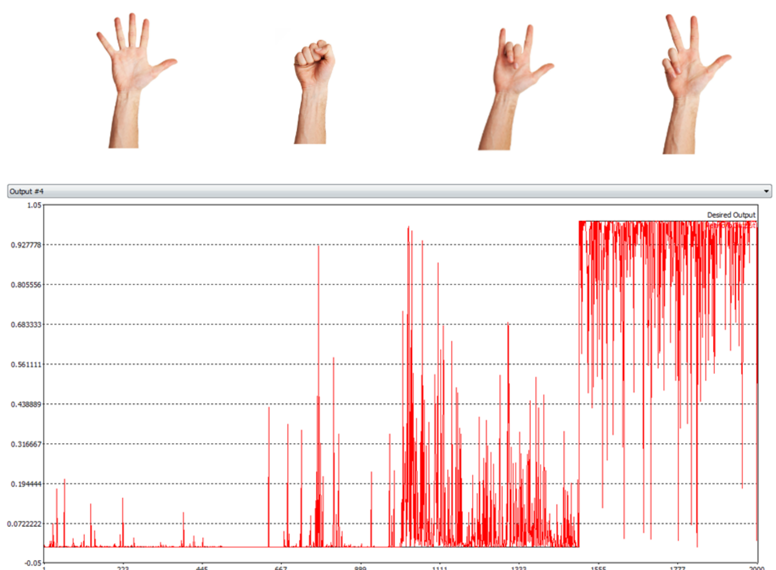

Figure 31: *ANN output for initial set of four hand gestures analyzed for recognition-Fourth hand gesture: Three hand*

From these plots the first and second hand gestures are able to be recognized with a very high degree of accuracy. However, as expected, the third and fourth hand gestures are not differentiated as well.

Overall, the system is able to reliably recognize the first three hand gestures with nearly 90% accuracy and at a very high response rate. In fact, between the motion of performing the first and second hand the gestures the system would often identify the third or fourth hand gesture being performed which in many cases was most likely accurate.

However, the system had a high error rate with recognizing the fourth hand gesture and often, about 30-40% of the time, classified it as the third hand gesture.
Because of this error a method for controlling and increases the accuracy of the HDI control, and slowing the system down, is to only activate a GPIO if the same hand gesture has been detected more than two times in a row. Using this method the system is nearly 100% accurate in identifying the correct hand gestures as it is very rare for the system to make an error, let alone the same error, repeatedly.
Furthermore, this ANN is able to be used repeatedly and accurately on different days after the ANN is trained and is transferable to others with similar arm size.

3.2 Results with Four Hand Gestures with More Training

The same training set using the four hand gestures show above and using the ANN topology shown in Figure 24 can be applied to more training.

The plot below shows the RMSE as it varies with the number of Epochs performed:

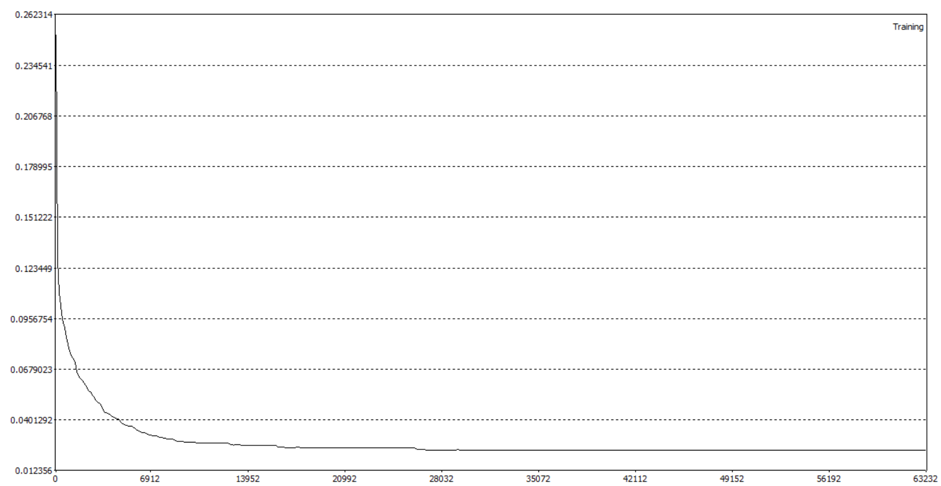

Figure 32: *RMSE of the ANN after 63345 Epochs over approximately 90 minutes*

From this test, the minimum obtained RMSE$\approx$ 0.023, from 63345 Epochs, which takes approximately 90 minutes using the Multiple Back Propagation software. Note approximately ten times more Epochs are performed and the RMSE is approximately halved relative to the previous test. The C file produced by this test can be seen in Appendix 5.6.

For the four hand gestures, the output and accuracy of the neural network can be seen below in the four plots. These plots indicate in black the known outputs of the training set and in red the believed outputs of the ANN.

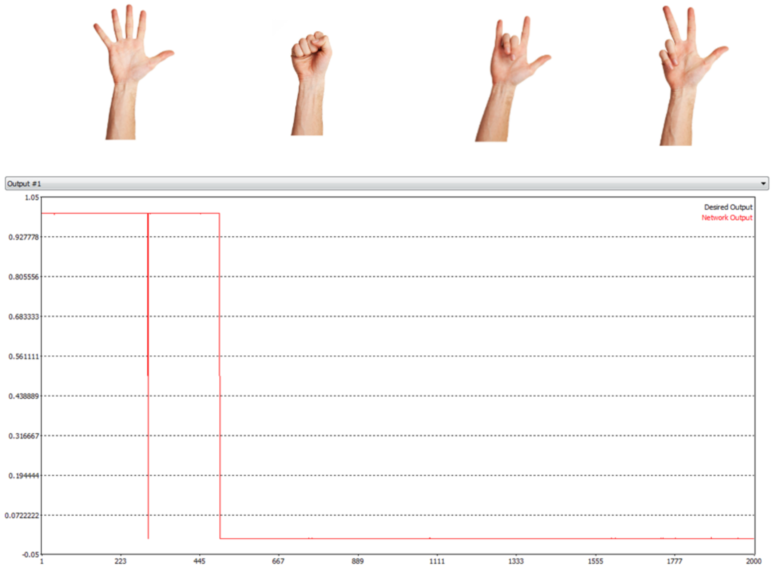

Figure 33: *ANN output for initial set of four hand gestures analyzed for recognition with more training-First hand gesture: Open hand*

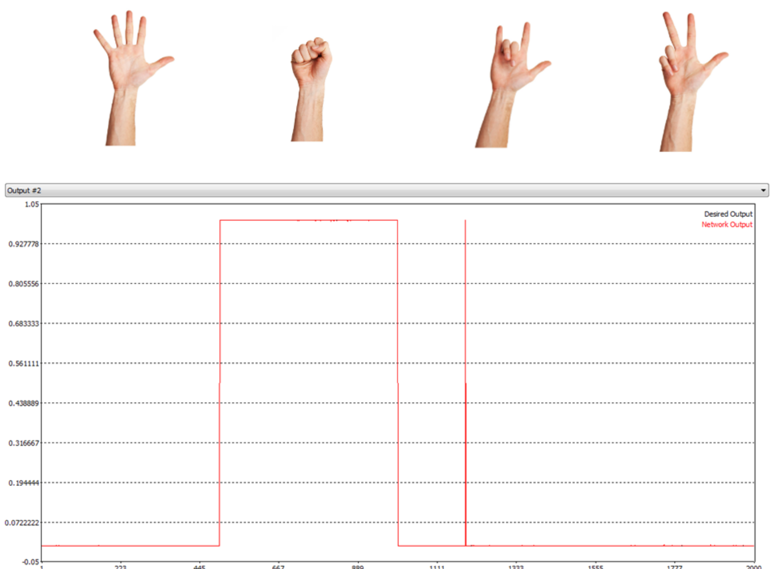

Figure 34: *ANN output for initial set of four hand gestures analyzed for recognition with more training-Second hand gesture: Closed hand*

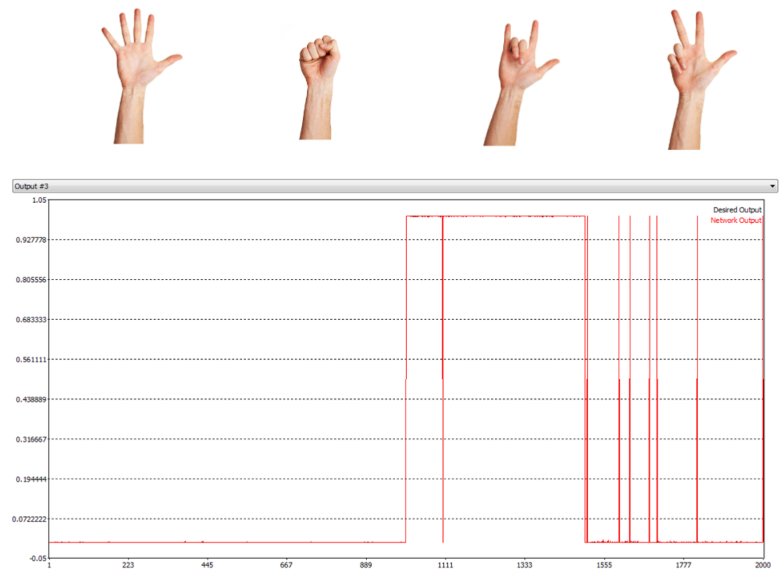

Figure 35: *ANN output for initial set of four hand gestures analyzed for recognition with more training-Third hand gesture: Spiderman hand*

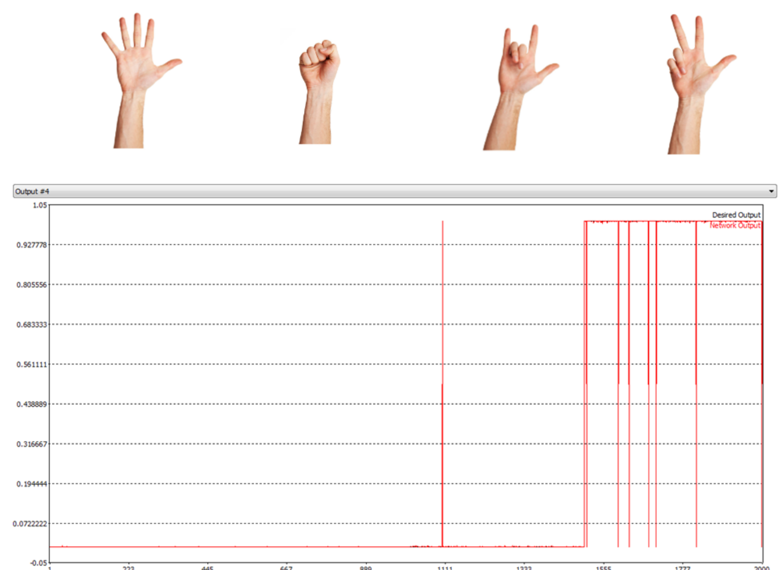

Figure 36: *ANN output for initial set of four hand gestures analyzed for recognition with more training-Fourth hand gesture: Three hand*

From these plots the hand gestures are able to be recognized with an extremely high degree of accuracy while the only significant errors are between the third and fourth hand gestures, which has decreased significantly relative to the previous test.
The system, under the same conditions as when it is trained, is able to reliably recognize the hand gestures with nearly 100% accuracy and at a very high response rate.
However, through testing it is determined that this ANN's ability to recognize EMG signals

that vary slightly from the original training set diminishes very quickly as slight changes and variations are added to the system. If applied after the arm band is removed and replaced on the arm the accuracy does diminish slightly. Furthermore, it does very poorly on others arms only characterizing the first and second hand gestures accurately and many times characterizing the third or fourth hand gestures as the second hand gesture.
Unlike the responses of the ANN of the previous test, which gave some weight to the other incorrect hand gestures, the responses of this ANN tend to be very close to 0 or 1 since it believes it is very confident about its recognition. Thus, when used immediately after training this system is nearly 100% accurate. However, its performance decreases as various factors are changed.

For these reasons the 'MBPnnEval.c' used, which can be seen in Appendix 5.5, is the C file produced from the first test with a minimum RMSE$\approx$ 0.046, training the data with 6335 Epochs over approximately 10 minutes since it is found to produce the more repeatable results.

4. Discussion

4.1 Future Direction

This project has several direction that would improve upon it and its performance.
First, utilizing all ten available GPIOs on the BeagleBone black would allow for richer control of the mouse and keyboard as opposed to the four GPIOs currently implemented. In order to achieve this, the accelerometer's integration could be completed and the change in certain angles could activate GPIOs. Furthermore, this system could be applied to more hand gestures that vary in the muscles they activate which might allow for the recognition of up to six hand gestures. All together, the accelerometer could be used to control the movement of the mouse via four GPIOs and six total hand gestures could be implemented to control the remaining six GPIOs.

This leads to the other main direction this project could go in. In order to increase the maximum number of hand gestures the system is able to recognize with a high degree of accuracy further testing should be done with varying types of hand gestures and with different ANN architectures. The Multiple Back Propagation software allows for the use of space networks, a novel method which is said to improve on recognition accuracy, which could be tested. Furthermore, different EMG signal characteristics could be tested to see which combination of characteristics achieves the best trained ANNs. In addition, it may be the case that more than four characteristics need to be acquired from each EMG signal in order to increase the number of hand gestures able to be recognized accurately. Furthermore, the AD7689 used has the potential of acquiring data for eight EMG signals. Thus, the number of surface electrodes could be increased, in fact doubled from the number used in this project, in order to acquire more EMG information and characteristics, for each hand gesture.
Since this would require a lot of repetitive testing, the Multiple Back Propagation software may not be ideal since the only way to train ANNs is through the user interface on a Windows machine. A possible alternative might be the FANN(Fast Artificial Neural Networks) library which is supported for C. This library is able to train, test and implement ANNs all through C code. Though the FANN library is not used in this project because it would have required an external third party library and installation of it on the BeagleBone Black board, it may be a valid option for future work.

Finally, in terms of replication, if this system were to be redesigned the PCB board would be redesigned to correct errors made, include the AC couple circuit and possibly allow for EMG eight channels. Furthermore, either an alternative to the BeagleBone Black might be used or the BeagleBone Black could be used without the Linux OS. This might be desired since given the application the BeagleBone Black is used for in this project the operating system many times hindered development and made communication with other devices, particularly the SPI communication with the AD7689, very difficult. In the future, perhaps a processor of similar processing power with the ability to run a compiled C program, perform SPI and I^2C protocols and activate GPIOs might be used.

4.2 Conclusion

Overall the system works as desired and is able to control the desired HDI, mouse movements and desired keys on a keyboard, in real time through hand gestures. Furthermore, the results are repeatable and the system can be used on different occasions, with the arm band being taken off and put back on, while maintaining a high level of accuracy. It is also able to be used by others but training an ANN with that person's data would most likely increase the accuracy of the system when they use it.

In conclusion, this project succeeded in developing, mostly from the ground up, a system that through EMG signals classified by ANNs allows for the control of an HDI through more natural motions and hand gestures.

Acknowledgments

I would like to thank the Swarthmore College Engineering Department for funding this project and providing me with the tools and lab space.
I would also like to acknowledge Edmond Jaoudi for all his help with the many electronic aspects of this project.
Finally, I would like to acknowledge and thank my advisor, Professor Erik Cheever, for his invaluable help in the completion of this project.

References

[1] S. F. Barrett and J. Kridner, editors. *Bad to the Bone: Crafting Electronics Systems with Beaglebone and BeagleBone Black*. Morgan & Claypool, 2013.

[2] R. N. Khushaba and S. Kodagoda. Electromyogram (emg) feature reduction using mutual components analysis for multifunction prosthetic fingers control. *Int. Conf. on Control, Automation, Robotics & Vision (ICARCV), Guangzhou*, pages 1534–1539, 2012.

[3] Precision, low power instrumentation amplifiers. `http://www.ti.com/lit/ds/sbos051b/sbos051b.pdf`, 1995 (revised-2005). Part Name: INA128.

[4] The 'rail splitter' precision to virtual ground. `http://pdf1.alldatasheet.com/datasheet-pdf/view/84557/TI/TLE2426CD.html`, 1997. Part Name: 2426C.

[5] Three-terminal positive voltage regulator. `http://pdf.datasheetcatalog.com/datasheet/motorola/MC7809.pdf`, 1997. Part Name: 5v-MC7805C, 6v-MC7806C.

[6] Micropower precision, dual/quad cmos rail-to-rail input/output amplifiers. `http://cds.linear.com/docs/en/datasheet/60789fa.pdf`, 2005. Part Name: LTC6079CGN.

[7] 16-bit, 4-channel/8-channel, 250 ksps pulsar adc. `http://www.analog.com/static/imported-files/data_sheets/AD7682_7689.pdf`, 2012. Part Name: AD7689.

[8] Mpu-6000/mpu-6050 register map and descriptions. `http://invensense.com/mems/gyro/documents/RM-MPU-6000A.pdf`, 2012. Part Name: MPU6050.

[9] 16-bit, 250 ksps, 8-channel, single supply, isolated data acquisition system. `http://www.analog.com/static/imported-files/circuit_notes/CN0254.pdf`, 2013. Part Name: CN-0254.

[10] Mpu-6000/mpu-6050 product specification. `http://www.invensense.com/mems/gyro/documents/PS-MPU-6000A-00v3.4.pdf`, 2013. Part Name: MPU6050.

[11] Documentation on beaglbone black. `http://elinux.org/Beagleboard:BeagleBoneBlack`. Accessed: 05/01/2014.

[12] Enabling spi on beaglbone black. `http://elinux.org/BeagleBone_Black_Enable_SPIDEV`. Accessed: 05/01/2014.

[13] Using i^2c on beaglbone black. `http://elinux.org/Interfacing_with_I2C_Devices`. Accessed: 05/01/2014.

[14] Beaglebone resources and tutorials. `http://derekmolloy.ie/beaglebone/`. Accessed: 05/01/2014.

[15] Latest image of angstrom for beaglebone black board. `http://beagleboard.org/latest-images/`. Accessed: 05/01/2014.

[16] Makey-makey website. `www.makeymakey.com`. Accessed: 05/01/2014.

[17] Multiple back propagation: Software source and turtorial. `http://mbp.sourceforge.net/`. Accessed: 05/01/2014.

5. Appendix

5.1 Complete List of Materials

- One Mini USB to USB cable
- One BeagleBone Black board from Texas Instruments
- One AD789 from Analog Devices
- One CN-0245 from Analog Devices
- One MPU6050 from InveSense
- One Makey Makey Board
- Nine Electrodes from Grass Technologies
- Nine color matching crimped wires to connect from Electrodes to screw terminals
- One male DB9 with screw terminals
- Ten20 Conductive Paste from Grass Technologies
- Velcro and Elastic straps for Electrodes
- Circuit Elements on AC Couple
  - Sixteen $1M\Omega$ resistors
  - Eight $0.1\mu F$ capacitors
  - One female DB9 connector with ribbon wire
  - One male DB9 connector with ribbon wire
- One Printed Circuit Board, see Figures 7, 8, 9
- Circuit Elements on PCB, see Figures 7, 8, 9
  - One LTC6079CGN from Linear Technology
  - Four INA128 from Texas Instruments
  - One 2426C from Texas Instruments
  - One mounted female DB9 adapter
  - One right angled ten pin connector, with 0.2in spacing
  - Four two pin jumpers
  - Four eight pin IC sockets
  - Twelve $0.1\mu F$ capacitors
  - Two $10\mu F$ capacitors
  - Four $51k\Omega$ resistors
  - Four 510Ω resistors
  - Eight precision $0.1\mu F$ capacitors

- Four pairs of precision $0.1\mu F$ capacitors soldered together in parallel for equivalent $0.2\mu F$ capacitors
- Eight precision $27k\Omega$ resistors
- Four pairs of precision $27k\Omega$ resistors soldered together in parallel for equivalent $13.5k\Omega$ resistors

- Mobile Power Supply
  - Batteries producing a voltage of at least 7.5V
  - Appropriate battery enclosure with wire leads
  - Two Male power to wire ground centered power adapters
  - One Female power to wire ground centered power adapter
  - One MC7805C from Motorola
  - One MC7806C from Motorola

5.2 HandGestureRec.c - Main Run File

```c
/*
handGestureRec.c By: David Nahmias
Electromyography data acquisition and analysis program for recognizing electromyography
    data via an Artificial Neural Network to drive GPIO signals.
Written by: David Nahmias
For Engineering 90 Senior Design project at Swarthmore College.

In order to run code effectively:
The number of hand gestures, variable defined as 'numOfHandGestures', initialized on line
     446, must be changed to the appropriate number.

To compile: gcc -o handGestureRec handGestureRec.c -lpthread -lm
To run ./handGestureRec
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <pthread.h>
#include <unistd.h>
#include <errno.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <linux/spi/spidev.h>
#include <linux/i2c-dev.h>
#include <linux/i2c.h>
#include <linux/types.h>

#include <math.h>
#include "classificationFunctions.c"
#include "MPBnnEval.c"
#include "beaglebone_gpio.h"

```

```c
#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))
#define MAXPATH 16

//Initialize struct for data transfer between threads
struct params{
    double data[4][128];
    pthread_mutex_t lock1;
    int flagStart;
    };

void* accelerometer_thread(void *arg){
    struct params *p = (struct params *)arg;

    //Start GPIO Setup for this Thread-----------------------------
    volatile void *gpio_addr = NULL;
    volatile unsigned int *gpio_setdataout_addr = NULL;
    volatile unsigned int *gpio_cleardataout_addr = NULL;
    int fd = open("/dev/mem", O_RDWR);

    gpio_addr = mmap(0, GPIO1_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd,
    GPIO1_START_ADDR);

    gpio_setdataout_addr = gpio_addr + GPIO_SETDATAOUT;
    gpio_cleardataout_addr = gpio_addr + GPIO_CLEARDATAOUT;

    if(gpio_addr == MAP_FAILED) {
        printf("Unable to map GPIO\n");
        exit(1);
    }

    //End GPIO Setup for this Thread-----------------------------

    uint16_t accel_x, accel_y, accel_z;
    unsigned char wBuf[64];
    unsigned char rBuf[64];

    int file;
    char *filename = "/dev/i2c-1";
    if ((file = open(filename, O_RDWR)) < 0) {
        /* ERROR HANDLING: you can check errno to see what went wrong */
        perror("Failed to open the i2c bus");
        exit(1);
    }

    unsigned char addr = 0x68;     // The I2C address of the device
    if (ioctl(file, I2C_SLAVE, addr) < 0) {
        printf("Failed to acquire bus access and/or talk to slave.\n");
        /* ERROR HANDLING; you can check errno to see what went wrong */
        exit(1);
    }

    unsigned char wBufOn[64];
    wBufOn[0] = 0x6b;
    wBufOn[1] = 0x00;
    if (write(file, wBufOn, 2) != 2) {
        // ERROR HANDLING: i2c transaction failed
        printf("Failed to read from the i2c bus: %s.\n", strerror(errno));
        printf("\n\n");
    }

    while(1){
```

```
        if (ioctl(file, I2C_SLAVE, addr) < 0) {
            printf("Failed to acquire bus access and/or talk to slave.\n");
            /* ERROR HANDLING; you can check errno to see what went wrong */
            exit(1);
        }
        //Set initial register to read
        wBuf[0] = 0x3b;

        if (write(file, wBuf, 1) != 1) {
            // ERROR HANDLING: i2c transaction failed
            printf("Failed to read from the i2c bus: %s.\n", strerror(errno));
            printf("\n\n");
        }

        if (ioctl(file, I2C_SLAVE, addr) < 0) {
            printf("Failed to acquire bus access and/or talk to slave.\n");
            /* ERROR HANDLING; you can check errno to see what went wrong */
            exit(1);
        }

        //Reading registers in burst mode
        // Using I2C Read
        if (read(file,rBuf,1) != 1) {
            /* ERROR HANDLING: i2c transaction failed */
            printf("Failed to read from the i2c bus: %s.\n", strerror(errno));
            printf("\n\n");
        }
        accel_x = rBuf[0];

        // Using I2C Read
        if (read(file,rBuf,1) != 1) {
            /* ERROR HANDLING: i2c transaction failed */
            printf("Failed to read from the i2c bus: %s.\n", strerror(errno));
            printf("\n\n");
        }
        accel_x = (accel_x<<8)+rBuf[0];
        //printf("Accel X: %X\n",accel_x);

        // Using I2C Read
        if (read(file,rBuf,1) != 1) {
            /* ERROR HANDLING: i2c transaction failed */
            printf("Failed to read from the i2c bus: %s.\n", strerror(errno));
            printf("\n\n");
        }
        accel_y = rBuf[0];

         // Using I2C Read
        if (read(file,rBuf,1) != 1) {
            /* ERROR HANDLING: i2c transaction failed */
            printf("Failed to read from the i2c bus: %s.\n", strerror(errno));
            printf("\n\n");
        }
        accel_y = (accel_y<<8)+rBuf[0];
        //printf("Accel Y: %X\n",accel_y);

        // Using I2C Read
        if (read(file,rBuf,1) != 1) {
            /* ERROR HANDLING: i2c transaction failed */
            printf("Failed to read from the i2c bus: %s.\n", strerror(errno));
            printf("\n\n");
```

```
        }
        accel_z = rBuf[0];

        // Using I2C Read
        if (read(file,rBuf,1) != 1) {
            /* ERROR HANDLING: i2c transaction failed */
            printf("Failed to read from the i2c bus: %s.\n", strerror(errno));
            printf("\n\n");
        }
        accel_z = (accel_z<<8)+rBuf[0];
        //printf("Accel Z: %X\n",accel_z);

        //Calculate angles from raw data
        float pitch = atan(accel_x/sqrt(pow(accel_y,2)+pow(accel_z,2)));
        float roll = atan(accel_y/sqrt(pow(accel_x,2)+pow(accel_z,2)));
        float yawish = atan(sqrt(pow(accel_x,2)+pow(accel_y,2))/accel_z);

        printf("pitch = %lf, roll = %lf, yaw = %lf\n",pitch,roll,yawish);
        /*Certain GPIOs could then be activated depending on change in pitch, roll or yaw
    .*/
    }
}

void* SPIdata_thread(void *arg){
    struct params *p = (struct params *)arg;

    //Start GPIO Setup-------------------------------
    volatile void *gpio_addr = NULL;
    volatile unsigned int *gpio_setdataout_addr = NULL;
    volatile unsigned int *gpio_cleardataout_addr = NULL;

    volatile unsigned int *gpio_oe_addr8_11 = NULL;
    unsigned int reg8_11;
    volatile unsigned int *gpio_oe_addr8_12 = NULL;
    unsigned int reg8_12;
    volatile unsigned int *gpio_oe_addr8_15 = NULL;
    unsigned int reg8_15;
    volatile unsigned int *gpio_oe_addr8_16 = NULL;
    unsigned int reg8_16;
    volatile unsigned int *gpio_oe_addr8_26 = NULL;
    unsigned int reg8_26;

    volatile unsigned int *gpio_oe_addr9_12 = NULL;
    unsigned int reg9_12;
    volatile unsigned int *gpio_oe_addr9_14 = NULL;
    unsigned int reg9_14;
    volatile unsigned int *gpio_oe_addr9_15 = NULL;
    unsigned int reg9_15;
    volatile unsigned int *gpio_oe_addr9_16 = NULL;
    unsigned int reg9_16;
    volatile unsigned int *gpio_oe_addr9_23 = NULL;
    unsigned int reg9_23;

    int fd = open("/dev/mem", O_RDWR);

    gpio_addr = mmap(0, GPIO1_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd,
    GPIO1_START_ADDR);

    gpio_setdataout_addr = gpio_addr + GPIO_SETDATAOUT;
    gpio_cleardataout_addr = gpio_addr + GPIO_CLEARDATAOUT;

```

```
    if(gpio_addr == MAP_FAILED) {
        printf("Unable to map GPIO\n");
        exit(1);
    }

    gpio_oe_addr8_11 = gpio_addr + GPIO_OE;
    reg8_11 = *gpio_oe_addr8_11;
    //printf("GPIO1 configuration: %X\n", reg8_11);
    reg8_11 = reg8_11 & (0xFFFFFFFF - GPIO8_11);
    *gpio_oe_addr8_11 = reg8_11;
    //printf("GPIO1 configuration: %X\n", reg8_11);

    gpio_oe_addr8_12 = gpio_addr + GPIO_OE;
    reg8_12 = *gpio_oe_addr8_12;
    //printf("GPIO1 configuration: %X\n", reg8_12);
    reg8_12 = reg8_12 & (0xFFFFFFFF - GPIO8_12);
    *gpio_oe_addr8_12 = reg8_12;
    //printf("GPIO1 configuration: %X\n", reg8_12);

    gpio_oe_addr8_15 = gpio_addr + GPIO_OE;
    reg8_15 = *gpio_oe_addr8_15;
    //printf("GPIO1 configuration: %X\n", reg8_15);
    reg8_15 = reg8_15 & (0xFFFFFFFF - GPIO8_15);
    *gpio_oe_addr8_15 = reg8_15;
    //printf("GPIO1 configuration: %X\n", reg8_15);

    gpio_oe_addr8_16 = gpio_addr + GPIO_OE;
    reg8_16 = *gpio_oe_addr8_16;
    //printf("GPIO1 configuration: %X\n", reg8_16);
    reg8_16 = reg8_16 & (0xFFFFFFFF - GPIO8_16);
    *gpio_oe_addr8_16 = reg8_16;
    //printf("GPIO1 configuration: %X\n", reg8_16);

    gpio_oe_addr8_26 = gpio_addr + GPIO_OE;
    reg8_26 = *gpio_oe_addr8_26;
    //printf("GPIO1 configuration: %X\n", reg8_26);
    reg8_26 = reg8_26 & (0xFFFFFFFF - GPIO8_26);
    *gpio_oe_addr8_26 = reg8_26;
    //printf("GPIO1 configuration: %X\n", reg8_26);


    gpio_oe_addr9_12 = gpio_addr + GPIO_OE;
    reg9_12 = *gpio_oe_addr9_12;
    //printf("GPIO1 configuration: %X\n", reg9_12);
    reg9_12 = reg9_12 & (0xFFFFFFFF - GPIO9_12);
    *gpio_oe_addr9_12 = reg9_12;
    //printf("GPIO1 configuration: %X\n", reg9_12);

    gpio_oe_addr9_14 = gpio_addr + GPIO_OE;
    reg9_14 = *gpio_oe_addr9_14;
    //printf("GPIO1 configuration: %X\n", reg9_14);
    reg9_14 = reg9_14 & (0xFFFFFFFF - GPIO9_14);
    *gpio_oe_addr9_14 = reg9_14;
    //printf("GPIO1 configuration: %X\n", reg9_14);

    gpio_oe_addr9_15 = gpio_addr + GPIO_OE;
    reg9_15 = *gpio_oe_addr9_15;
    //printf("GPIO1 configuration: %X\n", reg9_15);
    reg9_15 = reg9_15 & (0xFFFFFFFF - GPIO9_15);
    *gpio_oe_addr9_15 = reg9_15;
    //printf("GPIO1 configuration: %X\n", reg9_15);
```

```
    gpio_oe_addr9_16 = gpio_addr + GPIO_OE;
    reg9_16 = *gpio_oe_addr9_16;
    //printf("GPIO1 configuration: %X\n", reg9_16);
    reg9_16 = reg9_16 & (0xFFFFFFFF - GPIO9_16);
    *gpio_oe_addr9_16 = reg9_16;
    //printf("GPIO1 configuration: %X\n", reg9_16);

    gpio_oe_addr9_23 = gpio_addr + GPIO_OE;
    reg9_23 = *gpio_oe_addr9_23;
    //printf("GPIO1 configuration: %X\n", reg9_23);
    reg9_23 = reg9_23 & (0xFFFFFFFF - GPIO9_23);
    *gpio_oe_addr9_23 = reg9_23;
    //printf("GPIO1 configuration: %X\n", reg9_23);

    //End GPIO Setup------------------------------

    //Start SPI Setup------------------------------
    uint8_t bits = 16;
    int ret = 0;
    char* list;
    int length_list = 1;
    uint16_t delay = 5;
    uint32_t speed = 1000000;
    uint8_t tx[length_list];

    //Transmitted in this order since data is received at n-2
    uint16_t tx0[1]={0xf424};//Actually CH2
    uint16_t tx1[1]={0xf624};//Actually CH3
    uint16_t tx2[1]={0xf024};//Actually CH0
    uint16_t tx3[1]={0xf224};//Actually CH1

    //Initialize data received from SPI
    int8_t rx[ARRAY_SIZE(tx)];
    int16_t rx0[ARRAY_SIZE(tx0)];
    int16_t rx1[ARRAY_SIZE(tx1)];
    int16_t rx2[ARRAY_SIZE(tx2)];
    int16_t rx3[ARRAY_SIZE(tx3)];

    /*This is the transfer part, and sets up
    the details needed to transfer the data*/
    struct spi_ioc_transfer tr0 = {
  .tx_buf = (unsigned long)tx0,
  .rx_buf = (signed long)rx0,
  .len = 2*ARRAY_SIZE(tx0),
  .delay_usecs = delay,
  .speed_hz = speed,
  .bits_per_word = bits,
  };

    struct spi_ioc_transfer tr1 = {
  .tx_buf = (unsigned long)tx1,
  .rx_buf = (signed long)rx1,
  .len = 2*ARRAY_SIZE(tx1),
  .delay_usecs = delay,
  .speed_hz = speed,
  .bits_per_word = bits,
  };

    struct spi_ioc_transfer tr2 = {
  .tx_buf = (unsigned long)tx2,
```

```
.rx_buf = (signed long)rx2,
.len = 2*ARRAY_SIZE(tx2),
.delay_usecs = delay,
.speed_hz = speed,
.bits_per_word = bits,
};

  struct spi_ioc_transfer tr3 = {
.tx_buf = (unsigned long)tx3,
.rx_buf = (signed long)rx3,
.len = 2*ARRAY_SIZE(tx3),
.delay_usecs = delay,
.speed_hz = speed,
.bits_per_word = bits,
};

  int fd2 = open("/dev/spidev2.0",O_RDWR);
  if (fd2 < 0) {
      printf("Can't open device file: %X\n", fd2);
      //exit(-1);
  }

  //End SPI Setup------------------------------

  int length = 128; //Length of data packets
  int i;
  while(1){

      //Initially give mutex lock to this thread and then set flag high
      if(p->flagStart == 0){
          pthread_mutex_lock(&(p->lock1));
          p->flagStart = 1;
      }

      //printf("New Array!\n");
      //Toggle LED as data is acquired, ON indicates data is being acquired
      *gpio_setdataout_addr= USR1_LED;

      //Acquire data packets
      for(i=0;i<length;i++){
          //Perform SPI for each channel sequentially

          //CH0
          ret = ioctl(fd2, SPI_IOC_MESSAGE(1), &tr0);
          if (ret < 1){
              printf("ERROR: Can't send spi message \n");
          }

          //CH1
          ret = ioctl(fd2, SPI_IOC_MESSAGE(1), &tr1);
          if (ret < 1){
              printf("ERROR: Can't send spi message \n");
          }

          //CH2
          ret = ioctl(fd2, SPI_IOC_MESSAGE(1), &tr2);
          //*gpio_setdataout_addr = USR1_LED;
          if (ret < 1){
              printf("ERROR: Can't send spi message \n");
          }
```

```
            //CH3
            ret = ioctl(fd2, SPI_IOC_MESSAGE(1), &tr3);
            if (ret < 1){
                printf("ERROR: Can't send spi message \n");
            }

            //Allocate data from transfer
            p->data[0][i] = (double)rx0[0];
            p->data[1][i] = (double)rx1[0];
            p->data[2][i] = (double)rx2[0];
            p->data[3][i] = (double)rx3[0];

            //printf("R0 = %lf, R1 = %lf, R2= %lf, R3 = %lf\n",p->data[0][i],p->data[1][i],p->data[2][i],p->data[3][i]);
            //printf("R0 = %.4X, R1 = %.4X, R2= %.4X, R3 = %.4X\n",rx0[0],rx1[0],rx2[0],rx3[0]);

            //Delay for 1kHz sampling rate
            usleep(850);
            *gpio_cleardataout_addr = USR1_LED;
        }
        //Release mutex lock for data analysis thread to acquire
        pthread_mutex_unlock(&(p->lock1));
        //Imediately request mutex lock, to be acquired when data analysis thread releases the mutex lock
        pthread_mutex_lock(&(p->lock1));
    }
}

int main(int argc, char *argv[]) {
    //Start GPIO Setup for this Thread------------------------------
    volatile void *gpio_addr = NULL;
    volatile unsigned int *gpio_setdataout_addr = NULL;
    volatile unsigned int *gpio_cleardataout_addr = NULL;
    int fd = open("/dev/mem", O_RDWR);

    gpio_addr = mmap(0, GPIO1_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, GPIO1_START_ADDR);

    gpio_setdataout_addr = gpio_addr + GPIO_SETDATAOUT;
    gpio_cleardataout_addr = gpio_addr + GPIO_CLEARDATAOUT;

    if(gpio_addr == MAP_FAILED) {
        printf("Unable to map GPIO\n");
        exit(1);
    }

    //End GPIO Setup for this Thread------------------------------


    //CHANGE VALUE FOR NUMBER OF HAND GESTURES:
    int numOfHandGestures = 4;

    int lengthOfData = 128;
    int numOfChannels = 4;
    double dataCur[numOfChannels][lengthOfData];
    double dataCurMean[numOfChannels][lengthOfData];
    double proData[16];
    double filterCalc[2];
    double netOutput[numOfHandGestures];
    int maxLoc = 0;
```

```
    int prevHand = 0;
    int numHandSeq = 0;
    int slpChangedCur;
    int meanCross;
    int ch;
    int nnOut;

    //Initialize struct for data transfer between threads
    struct params *p = malloc(sizeof(struct params));
    //Initialize flag to low
    p->flagStart = 0;
    //Initialize mutex lock
    pthread_mutex_init(&(p->lock1), NULL);

    //Initialize data acquisition thread
    pthread_t pid;
    pthread_create(&pid,NULL,SPIdata_thread,(void*)p);

    //Initialize accelerometer thread
    pthread_t pid2;
    pthread_create(&pid2,NULL,accelerometer_thread,(void*)p);

    //Wait until flag is set high and data acquisition has begun
    while(p->flagStart != 1){
    }

    //Initialize GPIOs by setting them all low
    *gpio_cleardataout_addr = GPIO8_11;
    *gpio_cleardataout_addr = GPIO8_12;
    *gpio_cleardataout_addr = GPIO8_15;
    *gpio_cleardataout_addr = GPIO8_16;
    *gpio_cleardataout_addr = GPIO8_26;

    *gpio_cleardataout_addr = GPIO9_12;
    *gpio_cleardataout_addr = GPIO9_14;
    *gpio_cleardataout_addr = GPIO9_15;
    *gpio_cleardataout_addr = GPIO9_16;
    *gpio_cleardataout_addr = GPIO9_23;

    while(1){
        //Request mutex lock and acquire when data packet acquisition is complete
        pthread_mutex_lock(&(p->lock1));
        //Copy Data
        memcpy(dataCur,p->data,sizeof(p->data));
        //Release mutex lock to be acquired by data acquisition thread
        pthread_mutex_unlock(&(p->lock1));

        //Access the Data
        float data0,data1,data2,data3;
        data0=dataCur[0][0];
        data1=dataCur[1][0];
        data2=dataCur[2][0];
        data3=dataCur[3][0];
        //printf("First PeiceOfData: %lf, %lf, %lf, %lf\n", dataCur[0][0], dataCur[1][0],
  dataCur[2][0], dataCur[3][0]);

        //Perform analysis of data
        for(ch = 0; ch<numOfChannels; ch++){
            zeroMean(dataCur[ch],dataCurMean[ch],lengthOfData);
            filterHiLow(dataCurMean[ch],filterCalc,lengthOfData);
            proData[(0)+(ch*numOfChannels)] = filterCalc[0];
```

```
            proData[(1)+(ch*numOfChannels)] = filterCalc[1];

            slpChangedCur = slopeChange(dataCurMean[ch],lengthOfData);
            proData[(2)+(ch*numOfChannels)] = slpChangedCur;

            meanCross = meanCrossing(dataCurMean[ch],lengthOfData);
            proData[(3)+(ch*numOfChannels)] = meanCross;
         }

        //printf("Analyzed Data: %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf\n",proData[0],proData[1],proData[2],proData[3],proData[4],proData[5],proData[6],proData[7],proData[8],proData[9],proData[10],proData[11],proData[12],proData[13],proData[14],proData[15]);

/*The rest of the code is written with the HDI application in mind.*/

        //If the first element is small, less than 100, the hand is open, and the ANN can be avoided.
        if(proData[0] < 100){
            maxLoc = 0;
            printf("Hand gesture number: %d-",maxLoc+1);
            printf("Open Hand\n");
            *gpio_cleardataout_addr = GPIO8_11;
            *gpio_cleardataout_addr = GPIO8_12;
            *gpio_cleardataout_addr = GPIO8_15;
            *gpio_cleardataout_addr = GPIO8_16;
            *gpio_cleardataout_addr = GPIO8_26;

            *gpio_cleardataout_addr = GPIO9_12;
            *gpio_cleardataout_addr = GPIO9_14;
            *gpio_cleardataout_addr = GPIO9_15;
            *gpio_cleardataout_addr = GPIO9_16;
            *gpio_cleardataout_addr = GPIO9_23;
            *gpio_setdataout_addr = GPIO8_26;
        }
        else{
            //Evaluate data on ANN
            MPBnnEval(proData,netOutput);

            //ANN output for monotoring
            /*
            printf("ANN Output:");
            for(nnOut=0;nnOut<numOfHandGestures;nnOut++){
                printf(" %lf",netOutput[nnOut]);
            }
            printf("/n");
            */
            maxLoc = maxLocation(netOutput,numOfHandGestures);
        }


        //printf("Hand gesture number: %d\n",maxLoc+1);

        //Because of some errors in recognition only if the same hand gesture is decteted for a third time in a row are the GPIOs activated, so all GPIOs are set low
        if(prevHand == maxLoc){
            numHandSeq++;
        }
        else{
            numHandSeq = 0;
            prevHand = maxLoc;
```

```
            *gpio_cleardataout_addr = GPIO8_11;
            *gpio_cleardataout_addr = GPIO8_12;
            *gpio_cleardataout_addr = GPIO8_15;
            *gpio_cleardataout_addr = GPIO8_16;
            *gpio_cleardataout_addr = GPIO8_26;

            *gpio_cleardataout_addr = GPIO9_12;
            *gpio_cleardataout_addr = GPIO9_14;
            *gpio_cleardataout_addr = GPIO9_15;
            *gpio_cleardataout_addr = GPIO9_16;
            *gpio_cleardataout_addr = GPIO9_23;
        }

        if(numHandSeq > 2){
            if(maxLoc == 0){
                *gpio_cleardataout_addr = GPIO8_11;
                *gpio_cleardataout_addr = GPIO8_12;
                *gpio_cleardataout_addr = GPIO8_15;
                *gpio_cleardataout_addr = GPIO8_16;
                *gpio_cleardataout_addr = GPIO8_26;

                *gpio_cleardataout_addr = GPIO9_12;
                *gpio_cleardataout_addr = GPIO9_14;
                *gpio_cleardataout_addr = GPIO9_15;
                *gpio_cleardataout_addr = GPIO9_16;
                *gpio_cleardataout_addr = GPIO9_23;
                *gpio_setdataout_addr = GPIO8_26;
                printf("Hand gesture number: %d-",maxLoc);
                printf("Open Hand\n");
            }
            else if(maxLoc == 1){
                *gpio_cleardataout_addr = GPIO8_11;
                *gpio_cleardataout_addr = GPIO8_12;
                *gpio_cleardataout_addr = GPIO8_15;
                *gpio_cleardataout_addr = GPIO8_16;
                *gpio_cleardataout_addr = GPIO8_26;

                *gpio_cleardataout_addr = GPIO9_12;
                *gpio_cleardataout_addr = GPIO9_14;
                *gpio_cleardataout_addr = GPIO9_15;
                *gpio_cleardataout_addr = GPIO9_16;
                *gpio_cleardataout_addr = GPIO9_23;
                *gpio_setdataout_addr= GPIO8_11;
                printf("Hand gesture number: %d-",maxLoc+1);
                printf("Closed Hand\n");
            }
            else if(maxLoc == 2){
                *gpio_cleardataout_addr = GPIO8_11;
                *gpio_cleardataout_addr = GPIO8_12;
                *gpio_cleardataout_addr = GPIO8_15;
                *gpio_cleardataout_addr = GPIO8_16;
                *gpio_cleardataout_addr = GPIO8_26;

                *gpio_cleardataout_addr = GPIO9_12;
                *gpio_cleardataout_addr = GPIO9_14;
                *gpio_cleardataout_addr = GPIO9_15;
                *gpio_cleardataout_addr = GPIO9_16;
                *gpio_cleardataout_addr = GPIO9_23;
                *gpio_setdataout_addr= GPIO8_12;
                printf("Hand gesture number: %d-",maxLoc+1);
                printf("Spiderman Hand\n");
```

```
            }

            else if(maxLoc == 3){
                *gpio_cleardataout_addr = GPIO8_11;
                *gpio_cleardataout_addr = GPIO8_12;
                *gpio_cleardataout_addr = GPIO8_15;
                *gpio_cleardataout_addr = GPIO8_16;
                *gpio_cleardataout_addr = GPIO8_26;

                *gpio_cleardataout_addr = GPIO9_12;
                *gpio_cleardataout_addr = GPIO9_14;
                *gpio_cleardataout_addr = GPIO9_15;
                *gpio_cleardataout_addr = GPIO9_16;
                *gpio_cleardataout_addr = GPIO9_23;
                *gpio_setdataout_addr= GPIO8_15;
                printf("Hand gesture number: %d-",maxLoc+1);
                printf("Three Hand\n");
            }
            //Reset count to slow system down, constant GPIOs or GPIOs that are set to quickly are not detected
            numHandSeq = 0;
        }

    }
    return 0;
}
```

5.3 trainNNGesture.c

```
/*
trainNNGesture.c By: David Nahmias
Electromyography data acquisition and analysis program for storing data to be trainned on an Artificial Neural Network.
Written by: David Nahmias
For Engineering 90 Senior Design project at Swarthmore College.

In order to run code effectively:
The number of hand gestures, variable defined as 'numOfHandGestures', initialized on line 293, must be changed to the appropriate number.

Then the program can be compiled and run.
To compile: gcc -o trainNNGesture trainNNGesture.c -lpthread -lm
To run ./handGestureRec
Once all data is collected for training, to merge data type in terminal: cat * > allData.dat
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <linux/spi/spidev.h>
```

```
#include <linux/types.h>
#include <sys/ioctl.h>
#include <math.h>
#include "classificationFunctions.c"
#include "MPBnnEval.c"
#include "beaglebone_gpio.h"


#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))
#define MAXPATH 16

//Initialize struct for data transfer between threads
struct params{
    double data[4][128];
    pthread_mutex_t lock1;
    int flagStart;
    };

void* SPIdata_thread(void *arg){
    struct params *p = (struct params *)arg;

    //Start GPIO Setup--------------------------------
    volatile void *gpio_addr = NULL;
    volatile unsigned int *gpio_setdataout_addr = NULL;
    volatile unsigned int *gpio_cleardataout_addr = NULL;

    volatile unsigned int *gpio_oe_addr8_11 = NULL;
    unsigned int reg8_11;
    volatile unsigned int *gpio_oe_addr8_12 = NULL;
    unsigned int reg8_12;
    volatile unsigned int *gpio_oe_addr8_15 = NULL;
    unsigned int reg8_15;
    volatile unsigned int *gpio_oe_addr8_16 = NULL;
    unsigned int reg8_16;
    volatile unsigned int *gpio_oe_addr8_26 = NULL;
    unsigned int reg8_26;

    volatile unsigned int *gpio_oe_addr9_12 = NULL;
    unsigned int reg9_12;
    volatile unsigned int *gpio_oe_addr9_14 = NULL;
    unsigned int reg9_14;
    volatile unsigned int *gpio_oe_addr9_15 = NULL;
    unsigned int reg9_15;
    volatile unsigned int *gpio_oe_addr9_16 = NULL;
    unsigned int reg9_16;
    volatile unsigned int *gpio_oe_addr9_23 = NULL;
    unsigned int reg9_23;

    int fd = open("/dev/mem", O_RDWR);

    gpio_addr = mmap(0, GPIO1_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd,
    GPIO1_START_ADDR);

    gpio_setdataout_addr = gpio_addr + GPIO_SETDATAOUT;
    gpio_cleardataout_addr = gpio_addr + GPIO_CLEARDATAOUT;

    if(gpio_addr == MAP_FAILED) {
        printf("Unable to map GPIO\n");
        exit(1);
    }

```

```
gpio_oe_addr8_11 = gpio_addr + GPIO_OE;
reg8_11 = *gpio_oe_addr8_11;
//printf("GPIO1 configuration: %X\n", reg8_11);
reg8_11 = reg8_11 & (0xFFFFFFFF - GPIO8_11);
*gpio_oe_addr8_11 = reg8_11;
//printf("GPIO1 configuration: %X\n", reg8_11);

gpio_oe_addr8_12 = gpio_addr + GPIO_OE;
reg8_12 = *gpio_oe_addr8_12;
//printf("GPIO1 configuration: %X\n", reg8_12);
reg8_12 = reg8_12 & (0xFFFFFFFF - GPIO8_12);
*gpio_oe_addr8_12 = reg8_12;
//printf("GPIO1 configuration: %X\n", reg8_12);

gpio_oe_addr8_15 = gpio_addr + GPIO_OE;
reg8_15 = *gpio_oe_addr8_15;
//printf("GPIO1 configuration: %X\n", reg8_15);
reg8_15 = reg8_15 & (0xFFFFFFFF - GPIO8_15);
*gpio_oe_addr8_15 = reg8_15;
//printf("GPIO1 configuration: %X\n", reg8_15);

gpio_oe_addr8_16 = gpio_addr + GPIO_OE;
reg8_16 = *gpio_oe_addr8_16;
//printf("GPIO1 configuration: %X\n", reg8_16);
reg8_16 = reg8_16 & (0xFFFFFFFF - GPIO8_16);
*gpio_oe_addr8_16 = reg8_16;
//printf("GPIO1 configuration: %X\n", reg8_16);

gpio_oe_addr8_26 = gpio_addr + GPIO_OE;
reg8_26 = *gpio_oe_addr8_26;
//printf("GPIO1 configuration: %X\n", reg8_26);
reg8_26 = reg8_26 & (0xFFFFFFFF - GPIO8_26);
*gpio_oe_addr8_26 = reg8_26;
//printf("GPIO1 configuration: %X\n", reg8_26);


gpio_oe_addr9_12 = gpio_addr + GPIO_OE;
reg9_12 = *gpio_oe_addr9_12;
//printf("GPIO1 configuration: %X\n", reg9_12);
reg9_12 = reg9_12 & (0xFFFFFFFF - GPIO9_12);
*gpio_oe_addr9_12 = reg9_12;
//printf("GPIO1 configuration: %X\n", reg9_12);

gpio_oe_addr9_14 = gpio_addr + GPIO_OE;
reg9_14 = *gpio_oe_addr9_14;
//printf("GPIO1 configuration: %X\n", reg9_14);
reg9_14 = reg9_14 & (0xFFFFFFFF - GPIO9_14);
*gpio_oe_addr9_14 = reg9_14;
//printf("GPIO1 configuration: %X\n", reg9_14);

gpio_oe_addr9_15 = gpio_addr + GPIO_OE;
reg9_15 = *gpio_oe_addr9_15;
//printf("GPIO1 configuration: %X\n", reg9_15);
reg9_15 = reg9_15 & (0xFFFFFFFF - GPIO9_15);
*gpio_oe_addr9_15 = reg9_15;
//printf("GPIO1 configuration: %X\n", reg9_15);

gpio_oe_addr9_16 = gpio_addr + GPIO_OE;
reg9_16 = *gpio_oe_addr9_16;
//printf("GPIO1 configuration: %X\n", reg9_16);
reg9_16 = reg9_16 & (0xFFFFFFFF - GPIO9_16);
```

```
    *gpio_oe_addr9_16 = reg9_16;
    //printf("GPIO1 configuration: %X\n", reg9_16);

    gpio_oe_addr9_23 = gpio_addr + GPIO_OE;
    reg9_23 = *gpio_oe_addr9_23;
    //printf("GPIO1 configuration: %X\n", reg9_23);
    reg9_23 = reg9_23 & (0xFFFFFFFF - GPIO9_23);
    *gpio_oe_addr9_23 = reg9_23;
    //printf("GPIO1 configuration: %X\n", reg9_23);

    //End GPIO Setup--------------------------------

    //Start SPI Setup--------------------------------
    uint8_t bits = 16;
    int ret = 0;
    char* list;
    int length_list = 1;
    uint16_t delay = 5;
    uint32_t speed = 1000000;
    uint8_t tx[length_list];

    //Transmitted in this order since data is received at n-2
    uint16_t tx0[1]={0xf424};//Actually CH2
    uint16_t tx1[1]={0xf624};//Actually CH3
    uint16_t tx2[1]={0xf024};//Actually CH0
    uint16_t tx3[1]={0xf224};//Actually CH1

    //Initialize data received from SPI
    int8_t rx[ARRAY_SIZE(tx)];
    int16_t rx0[ARRAY_SIZE(tx0)];
    int16_t rx1[ARRAY_SIZE(tx1)];
    int16_t rx2[ARRAY_SIZE(tx2)];
    int16_t rx3[ARRAY_SIZE(tx3)];

    /*This is the transfer part, and sets up
    the details needed to transfer the data*/
    struct spi_ioc_transfer tr0 = {
  .tx_buf = (unsigned long)tx0,
  .rx_buf = (signed long)rx0,
  .len = 2*ARRAY_SIZE(tx0),
  .delay_usecs = delay,
  .speed_hz = speed,
  .bits_per_word = bits,
  };

    struct spi_ioc_transfer tr1 = {
  .tx_buf = (unsigned long)tx1,
  .rx_buf = (signed long)rx1,
  .len = 2*ARRAY_SIZE(tx1),
  .delay_usecs = delay,
  .speed_hz = speed,
  .bits_per_word = bits,
  };

    struct spi_ioc_transfer tr2 = {
  .tx_buf = (unsigned long)tx2,
  .rx_buf = (signed long)rx2,
  .len = 2*ARRAY_SIZE(tx2),
  .delay_usecs = delay,
  .speed_hz = speed,
  .bits_per_word = bits,
```

```
};

  struct spi_ioc_transfer tr3 = {
.tx_buf = (unsigned long)tx3,
.rx_buf = (signed long)rx3,
.len = 2*ARRAY_SIZE(tx3),
.delay_usecs = delay,
.speed_hz = speed,
.bits_per_word = bits,
};

  int fd2 = open("/dev/spidev2.0",O_RDWR);
  if (fd2 < 0) {
      printf("Can't open device file: %X\n", fd2);
      //exit(-1);
  }

  //End SPI Setup--------------------------------

  int length = 128; //Length of data packets
  int i;
  while(1){

      //Initially give mutex lock to this thread and then set flag high
      if(p->flagStart == 0){
          pthread_mutex_lock(&(p->lock1));
          p->flagStart = 1;
      }

      //printf("New Array!\n");
      //Toggle LED as data is acquired, ON indicates data is being acquired
      *gpio_setdataout_addr= USR1_LED;

      //Acquire data packets
      for(i=0;i<length;i++){
          //Perform SPI for each channel sequentially

          //CH0
          ret = ioctl(fd2, SPI_IOC_MESSAGE(1), &tr0);
          if (ret < 1){
              printf("ERROR: Can't send spi message \n");
          }

          //CH1
          ret = ioctl(fd2, SPI_IOC_MESSAGE(1), &tr1);
          if (ret < 1){
              printf("ERROR: Can't send spi message \n");
          }

          //CH2
          ret = ioctl(fd2, SPI_IOC_MESSAGE(1), &tr2);
          //*gpio_setdataout_addr = USR1_LED;
          if (ret < 1){
              printf("ERROR: Can't send spi message \n");
          }

          //CH3
          ret = ioctl(fd2, SPI_IOC_MESSAGE(1), &tr3);
          if (ret < 1){
              printf("ERROR: Can't send spi message \n");
          }
```

```
            //Allocate data from transfer
            p->data[0][i] = (double)rx0[0];
            p->data[1][i] = (double)rx1[0];
            p->data[2][i] = (double)rx2[0];
            p->data[3][i] = (double)rx3[0];

            //printf("R0 = %lf, R1 = %lf, R2= %lf, R3 = %lf\n",p->data[0][i],p->data[1][i],p->data[2][i],p->data[3][i]);
            //printf("R0 = %.4X, R1 = %.4X, R2= %.4X, R3 = %.4X\n",rx0[0],rx1[0],rx2[0],rx3[0]);

            //Delay for 1kHz sampling rate
            usleep(850);
            *gpio_cleardataout_addr = USR1_LED;
        }
        //Release mutex lock for data analysis thread to acquire
        pthread_mutex_unlock(&(p->lock1));
        //Imediately request mutex lock, to be acquired when data analysis thread releases the mutex lock
        pthread_mutex_lock(&(p->lock1));
    }
}

int main(int argc, char *argv[]) {

    //CHANGE VALUE FOR NUMBER OF HAND GESTURES:
    int numOfHandGestures = 4;

    int lengthOfData = 128;
    int numOfChannels = 4;
    double dataCur[numOfChannels][lengthOfData];
    double dataCurMean[numOfChannels][lengthOfData];
    double proData[16];
    double filterCalc[2];
    double netOutput[numOfHandGestures];
    int maxLoc = 0;
    int slpChangedCur;
    int meanCross;
    int ch;
    int knownOut[numOfHandGestures];
    int handGest;
    int curHandKnown;
    int numOfIter=500;
    int iter;
    int storeKnown;

    printf("Beginning data acquisition\n");
    printf("Please make and hold hand gesture 1\n");
    printf("You have three seconds until data acquisition begins\n");
    usleep(3000000);

    //Initialize struct for data transfer between threads
    struct params *p = malloc(sizeof(struct params));
    //Initialize flag to low
    p->flagStart = 0;
    //Initialize mutex lock
    pthread_mutex_init(&(p->lock1), NULL);

    //Initialize data acquisition thread
    pthread_t pid;
```

```
    pthread_create(&pid,NULL,SPIdata_thread,(void*)p);

    //Wait until flag is set high and data acquisition has begun
    while(p->flagStart != 1){
    }
    //Iterate through the hand gestures
    for(handGest=0;handGest<numOfHandGestures;handGest++){
        if(handGest != 0){
            printf("Please make and hold hand gesture %d\n",handGest+1);
            printf("You have three seconds until data acquisition begins\n");
            usleep(3000000);
        }
        for(curHandKnown = 0; curHandKnown<numOfHandGestures;curHandKnown++){
            if(curHandKnown == handGest){
                knownOut[curHandKnown] = 1;
            }
            else{
                knownOut[curHandKnown] = 0;
            }

        }

        //Iterate through each trial for each hand gesture
        for(iter=0;iter<numOfIter;iter++){
            //Request mutex lock and acquire when data packet acquisition is complete
            pthread_mutex_lock(&(p->lock1));
            //Copy Data
            memcpy(dataCur,p->data,sizeof(p->data));
            //Release mutex lock to be acquired by data acquisition thread
            pthread_mutex_unlock(&(p->lock1));

            printf("Hand Gesture: %d, Trial Number: %d\n",handGest+1,iter+1);

            //Access the Data
            float data0,data1,data2,data3;
            data0=dataCur[0][0];
            data1=dataCur[1][0];
            data2=dataCur[2][0];
            data3=dataCur[3][0];
            //printf("First PeiceOfData: %lf, %lf, %lf, %lf\n", dataCur[0][0], dataCur
[1][0], dataCur[2][0], dataCur[3][0]);

            //Perform analysis of data
            for(ch = 0; ch<numOfChannels; ch++){
                zeroMean(dataCur[ch],dataCurMean[ch],lengthOfData);
                filterHiLow(dataCurMean[ch],filterCalc,lengthOfData);
                proData[(0)+(ch*numOfChannels)] = filterCalc[0];
                proData[(1)+(ch*numOfChannels)] = filterCalc[1];

                slpChangedCur = slopeChange(dataCurMean[ch],lengthOfData);
                proData[(2)+(ch*numOfChannels)] = slpChangedCur;

                meanCross = meanCrossing(dataCurMean[ch],lengthOfData);
                proData[(3)+(ch*numOfChannels)] = meanCross;
            }
            //Monotor data to be stored
            //printf("Analyzed Data: %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf
 %lf %lf %lf\n",proData[0],proData[1],proData[2],proData[3],proData[4],proData[5],
proData[6],proData[7],proData[8],proData[9],proData[10],proData[11],proData[12],
proData[13],proData[14],proData[15]);

```

```
            //Store data in .txt files for each hand gesture.
            char filename [50];
            if(handGest<10){
                sprintf(filename, "/home/root/Documents/E90/data/myProData0%d.txt",
    handGest);
            }
            else{
                sprintf(filename, "/home/root/Documents/E90/data/myProData%d.txt",
    handGest);
            }
            FILE *writeFile = fopen(filename, "a");
            fprintf(writeFile,"%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %
    lf %lf",proData[0],proData[1],proData[2],proData[3],proData[4],proData[5],proData[6],
    proData[7],proData[8],proData[9],proData[10],proData[11],proData[12],proData[13],
    proData[14],proData[15]);

            for(storeKnown = 0; storeKnown<numOfHandGestures; storeKnown++){
                fprintf(writeFile," %d",knownOut[storeKnown]);
            }

            fprintf(writeFile,"/n");

            fclose(writeFile);

        }
        printf("Done with hand gesture %d\n",handGest+1);
    }
    printf("Done with all hand gestures!\n");
    return 0;
}
```

5.4 classificationFunctions.c

```
/*
classificationFunctions.c By: David Nahmias
Electromyography data acquisition and analysis program for recognizing electromyography
    data via an Artificial Neural Network to drive GPIO signals.
Written by: David Nahmias
For Engineering 90 Senior Design project at Swarthmore College.

This code is a compilation of functions that are used in the main trainning and
    evaluation C files.
*/

#include <math.h>
#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))

/*Function for zero-ing out the mean from the vector*/
void zeroMean(double *inVec, double *outVec, int length){

    double average;
    double sum;
    double n = length;
    int  i;
    for(i = 0; i < length; i++){
        sum = sum + inVec[i];
    }
```

```
    average = sum / n;

    for(i=0;i<(length-1);i++){
        outVec[i] = inVec[i]-average;
    }
}

/*Function for calculating the number of times the slope changes sign in the vector*/
int slopeChange(double *inVec, int length){

    double diffVec[length-1];
    double signVec[length-1];
    double diffSignVec[length-2];
    int signChanges = 0;
    int i;
    for(i=0;i<(length-1);i++){
        diffVec[i] = inVec[2*i+1]-inVec[2*i];
        if(diffVec[i] < 0){
            signVec[i] = -1;
        }
        else if(diffVec[i] > 0){
            signVec[i] = 1;
        }
        else{
            signVec[i] = 0;
        }
     }
     for(i=0;i<(length-2);i++){
        diffSignVec[i] = fabs(signVec[2*i+1]-signVec[2*i]);
        if(diffSignVec[i]==2){
            signChanges++;
        }
     }
    return signChanges;
}

/*Function for calculating the number of times the array crosses the mean, or zero since
    the mean will be zero, of the vector*/
int meanCrossing(double *inVec, int length){

    int meanCrossed = 0;
    int zeroFlag = 0;
    int i;
    for(i=0;i<(length-1);i++){
        if(inVec[i]*inVec[i+1]<0){
            meanCrossed++;
        }
        else if(inVec[i]*inVec[i+1]==0){
             zeroFlag = 1;
        }
        else{
            if(zeroFlag == 1){
                meanCrossed++;
                zeroFlag= 0;
            }
        }
    }
    return meanCrossed;
}
```

```
/*Function for calculating the variance of a vector*/
double varianceCalc(double *inVec, int length){

    double average;
    double variance;
    double meanSum = 0;
    double varSum = 0;
    double n = length;
    int  i;
    for (i = 0; i < length; i++){
        meanSum = meanSum + inVec[i];
    }
    average = meanSum / n;

    for (i = 0; i < length; i++){
        varSum = varSum + pow((inVec[i] - average), 2);
    }
    variance = varSum / n;
    return variance;
}

/*Function applies an high and low pass filter to the vector and then returns the
    variance of the high and low pass frequencies*/
void filterHiLow(double *inVec, double *outVec, int length){

    int newLength;
    newLength = length - (7-1);

    //From MATLAB: firpm
    double low[7] = {-0.1195, 0.0001, 0.3133, 0.5002, 0.3133, 0.0001, -0.1195};
    double hi[7] = {0.1195, -0.0001, -0.3133, 0.4998, -0.3133, -0.0001, 0.1195};

    double sumLow =  0.8878;
    double sumHi = 0.1122;

    double weightLow;
    double weightHi;

    double varLow;
    double varHi;

    double lowPass[newLength];
    double hiPass[newLength];

    int i;
    int j;
    for(i=0;i<newLength;i++){
        weightLow = 0;
        weightHi = 0;
        for(j=0;j<7;j++){
            weightLow = weightLow + (low[j]*inVec[i+j]);
            weightHi = weightHi + (hi[j]*inVec[i+j]);
        }
        lowPass[i] = weightLow;
        hiPass[i] = weightHi;
    }
    varLow = varianceCalc(lowPass,newLength);
    varHi = varianceCalc(hiPass,newLength);
    outVec[0] = varLow;
    outVec[1] = varHi;
}
```

```
/*Function finds the location in the vector of the maximum value*/
int maxLocation(double *outputVec, int length){
    double maximum;
    maximum = outputVec[0];
    int location = 0;
    int c;
    for (c = 1; c < length; c++){
        if (outputVec[c] > maximum){
            maximum  = outputVec[c];
            location = c;
        }
    }
    return location;
}

/*Function finds the maximum value of a vector*/
double maxValue(double *outputVec, int length){
    double maximum;
    maximum = outputVec[0];
    int location = 0;
    int c;
    for (c = 1; c < length; c++){
        if (outputVec[c] > maximum){
            maximum  = outputVec[c];
            location = c;
        }
    }
    return maximum;
}
```

5.5 MPBnnEval.c - From Results in Section 3.1

```
/**
 Generated by Multiple Back-Propagation Version 2.2.4
 Multiple Back-Propagation can be freely obtained at http://dit.ipg.pt/MBP
*/

#include <math.h>
/**
 inputs  - should be an array of 16 element(s), containing the network input(s).
 outputs - should be an array of 4 element(s), that will contain the network output(s).
 Note : The array inputs will also be changed.Its values will be rescaled between -1 and
    1.
*/
void MPBnnEval(double * inputs, double * outputs) {
  double mainWeights;\\ARRAY INITIALIZED IN THE FOLLOWING APPENDIX
  double * mw = mainWeights;
  double hiddenLayer1outputs[20];
  int c;

  inputs[0] = -1.0 + (inputs[0] - 21.573288999999999) / 4287.211465999999700;
  inputs[1] = -1.0 + (inputs[1] - 22.532620999999999) / 959.162792999999970;
  inputs[2] = -1.0 + (inputs[2] - 14.000000000000000) / 16.500000000000000;
  inputs[3] = -1.0 + (inputs[3] - 3.000000000000000) / 39.500000000000000;
  inputs[4] = -1.0 + (inputs[4] - 7.401209000000000) / 2176.857930000000100;
  inputs[5] = -1.0 + (inputs[5] - 8.623840000000000) / 945.550248000000010;
```

```
inputs[6] = -1.0 + (inputs[6] - 14.000000000000000) / 16.500000000000000;
inputs[7] = -1.0 + (inputs[7] - 2.000000000000000) / 37.000000000000000;
inputs[8] = -1.0 + (inputs[8] - 32.803324000000003) / 790.981097500000030;
inputs[9] = -1.0 + (inputs[9] - 37.851123000000001) / 1588.695454500000100;
inputs[10] = -1.0 + (inputs[10] - 17.000000000000000) / 14.500000000000000;
inputs[11] = -1.0 + (inputs[11] - 15.000000000000000) / 33.500000000000000;
inputs[12] = -1.0 + (inputs[12] - 12.524685000000000) / 1522.454503000000200;
inputs[13] = -1.0 + (inputs[13] - 14.716892000000000) / 1559.156945000000000;
inputs[14] = -1.0 + (inputs[14] - 21.000000000000000) / 13.000000000000000;
inputs[15] = -1.0 + (inputs[15] - 13.000000000000000) / 33.500000000000000;
hiddenLayer1outputs[0] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[0] += *mw++ * inputs[c];
hiddenLayer1outputs[0] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[0]));
hiddenLayer1outputs[1] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[1] += *mw++ * inputs[c];
hiddenLayer1outputs[1] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[1]));
hiddenLayer1outputs[2] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[2] += *mw++ * inputs[c];
hiddenLayer1outputs[2] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[2]));
hiddenLayer1outputs[3] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[3] += *mw++ * inputs[c];
hiddenLayer1outputs[3] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[3]));
hiddenLayer1outputs[4] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[4] += *mw++ * inputs[c];
hiddenLayer1outputs[4] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[4]));
hiddenLayer1outputs[5] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[5] += *mw++ * inputs[c];
hiddenLayer1outputs[5] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[5]));
hiddenLayer1outputs[6] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[6] += *mw++ * inputs[c];
hiddenLayer1outputs[6] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[6]));
hiddenLayer1outputs[7] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[7] += *mw++ * inputs[c];
hiddenLayer1outputs[7] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[7]));
hiddenLayer1outputs[8] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[8] += *mw++ * inputs[c];
hiddenLayer1outputs[8] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[8]));
hiddenLayer1outputs[9] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[9] += *mw++ * inputs[c];
hiddenLayer1outputs[9] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[9]));
hiddenLayer1outputs[10] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[10] += *mw++ * inputs[c];
hiddenLayer1outputs[10] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[10]));
hiddenLayer1outputs[11] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[11] += *mw++ * inputs[c];
hiddenLayer1outputs[11] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[11]));
hiddenLayer1outputs[12] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[12] += *mw++ * inputs[c];
hiddenLayer1outputs[12] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[12]));
hiddenLayer1outputs[13] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[13] += *mw++ * inputs[c];
hiddenLayer1outputs[13] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[13]));
hiddenLayer1outputs[14] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[14] += *mw++ * inputs[c];
hiddenLayer1outputs[14] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[14]));
hiddenLayer1outputs[15] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[15] += *mw++ * inputs[c];
hiddenLayer1outputs[15] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[15]));
hiddenLayer1outputs[16] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[16] += *mw++ * inputs[c];
hiddenLayer1outputs[16] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[16]));
```

```
    hiddenLayer1outputs[17] = *mw++;
    for(c = 0; c < 16; c++) hiddenLayer1outputs[17] += *mw++ * inputs[c];
    hiddenLayer1outputs[17] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[17]));
    hiddenLayer1outputs[18] = *mw++;
    for(c = 0; c < 16; c++) hiddenLayer1outputs[18] += *mw++ * inputs[c];
    hiddenLayer1outputs[18] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[18]));
    hiddenLayer1outputs[19] = *mw++;
    for(c = 0; c < 16; c++) hiddenLayer1outputs[19] += *mw++ * inputs[c];
    hiddenLayer1outputs[19] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[19]));
    outputs[0] = *mw++;
    for(c = 0; c < 20; c++) outputs[0] += *mw++ * hiddenLayer1outputs[c];
    outputs[0] = 1.0 / (1.0 + exp(-outputs[0]));
    outputs[1] = *mw++;
    for(c = 0; c < 20; c++) outputs[1] += *mw++ * hiddenLayer1outputs[c];
    outputs[1] = 1.0 / (1.0 + exp(-outputs[1]));
    outputs[2] = *mw++;
    for(c = 0; c < 20; c++) outputs[2] += *mw++ * hiddenLayer1outputs[c];
    outputs[2] = 1.0 / (1.0 + exp(-outputs[2]));
    outputs[3] = *mw++;
    for(c = 0; c < 20; c++) outputs[3] += *mw++ * hiddenLayer1outputs[c];
    outputs[3] = 1.0 / (1.0 + exp(-outputs[3]));
    outputs[0] = 0.000000000000000 + (outputs[0] - 0.000000) * 1.000000000000000;
    outputs[1] = 0.000000000000000 + (outputs[1] - 0.000000) * 1.000000000000000;
    outputs[2] = 0.000000000000000 + (outputs[2] - 0.000000) * 1.000000000000000;
    outputs[3] = 0.000000000000000 + (outputs[3] - 0.000000) * 1.000000000000000;
}
```

5.5.1 mainWeights[] array from MPBnnEval.c - From Results in Section 3.1

mainWeights[] = {0.685057486267876, -1.003593845648203, -1.676608862226744, 2.759520572817846, -1.976107508630466, -0.300397806065257, -1.833566392813919, 0.393651686027290, -0.791570874695809, 3.973442875797390, -0.877755604819427, 0.536933382723225, 1.499490216281518, 1.826144752791696, 1.816447300382206, -0.174516172006360, 2.029122838843685, -44.823880538328645, -0.067448670483561, -1.215317142679038, -1.698089788637212, 1.630191445718890, 0.806198728759696, 0.128835153451791, -0.228734368784205, -0.353276029818936, -1.848567547814230, -1.884502538306788, -0.012315113746165, -0.325916819851736, -2.623195426310435, -38.081941942108976, 0.754381765612835, 0.818628784554373, 0.675791943569465, 1.785323478313909, 0.897168352562741, -5.763152038663192, -1.603692086085236, 0.029365814661941, -0.188353697214450, -1.901147563053155, 2.242399764271029, 0.045972869117599, 1.144191690451069, 0.689996850957407, -4.266455398177964, -2.358652729790622, -3.025679466115171, 2.081901291921557, -4.067890204334095, 5.250810665845856, -1.287667378884949, 0.996327022127673, -3.215593434487294, 3.040226634247922, 0.775797861253517, 2.169073428657368, 0.514126058550671, -1.890160749170135, 0.547415318323207, -1.703295072445366, -0.738535522233358, -0.675921097994960, 1.239632207453166, 1.982287908366938, 1.384455362674789, 1.222328251024730, -10.432879656963086, -0.598379529744183, -1.683285251696423, 1.301311153494513, 8.851481008532719, 0.480614381499928, -1.068100139509267, -10.003886064070269, 7.315733815509374, -5.770328952175156, 0.455406480616313, -0.307703450310253, 4.520635256255350, 1.249564669695893, 0.239244942553280, -5.395171914089017, -3.929409394484025, 2.376051826085683, 0.694159213010448, 1.249026849472953, -0.531920176489723, -0.375507139962586, -0.025023480245558, 6.720226347966567, 0.208667272622319, 0.215267258071795, 1.962301221454270, -0.544511094067143, -0.144959346873187, 0.832291517453934, 0.955270277824269, -7.533944025102121, -0.331179616835965, -0.976758945192899, 0.935163271085134, 0.071513823205666, -19.474660327108314, -4.364297688541105, 9.411498531240691, 25.503248891215581, 5.678913177432258, -5.964288703101865, -20.763476599762523, -1.514855154257174, -0.984142948706538, 1.787176915056193,

-1.918818477205228, -0.632610516537205, -28.533982119604740, -12.177883713464427, -3.493661789953968, 1.841690740465549, -1.377408413233161, -1.965838436968509, -3.132099277438348, 6.300793846974998, 0.517120565648646, 3.695398356822787, -0.087890156886894, -7.205321347002930, -0.679809721048382, -1.239171044818355, -0.752917223625244, -0.783812752906245, 2.425783995523105, 0.759045036842516, -2.284793359981405, -0.073026847717325, 0.676227054751294, -2.750189050028979, -1.705122636897508, -12.910752809290347, 27.631801337657308, 1.500994623769449, 0.688713460870922, 14.700630063227681, -22.289309760561466, 1.038239658760087, -2.548997547570192, 5.285363632416505, 5.963274049217061, 2.241199757213401, 2.558436426862659, -4.632851134345581, -27.852915154773285, 0.305652866273470, 1.070495070774060, 1.575939316116395, -4.278297152465886, 0.048533844122452, 0.492251473209713, 31.905437069288944, 5.570282528566866, 13.197882317313312, -1.975742790127589, 1.161087776277129, 9.733934114432229, 26.224172364450734, -1.798605197639050, -0.846438227574811, 7.636231185730157, -0.295178886198835, 0.025124095678517, -0.579880857520154, -2.261306061073901, -1.412081057314887, 4.075319952964382, -0.061720714583674, 2.098617674357036, 0.750212507402547, 1.937432889213032, -0.689410815647644, -3.378953149690740, 0.594075477393104, 2.032588831940989, 3.005900723006639, 3.076606627022861, -0.699679755810932, -2.542367621524663, 5.683225807598267, -1.064966242501702, -2.791865008755474, -4.384765124638936, -0.877942064447205, -0.660487208774154, 1.823155631028940, -2.315402533149195, -0.831426582101321, 1.361094888442151, -3.245891751969088, -1.798919421783587, -0.588921482487698, 4.323756093932172, 3.824432028645134, 0.237610270221678, -1.293104343693255, 4.741155268262678, -1.904114865664002, -0.180684195269555, 0.522208846750896, -0.300256803135168, 1.054026154652748, 4.881785800022747, 0.016382901797972, 0.743472782799455, 0.229108289533166, -3.684821179959983, 0.005190172156843, -0.378335705305012, 1.887913674108564, 2.004807040069288, -0.597875123839909, -0.504966057405074, 1.550876028996906, -1.867805164911597, -0.930410868644463, -0.362544841184606, 0.284172011991663, 0.867506701309391, 4.703004795287562, 0.066965985441227, 0.666858487989078, -6.755372711715438, -5.764172854207710, -0.136491374515942, -1.572384525388857, 4.768446578366890, 7.792785641590839, -0.000388190604450, -0.797706846306004, -6.837312235906400, -0.383402743368204, 0.979482088400743, -1.942489946328174, 0.921118930352360, 0.251090651576643, 0.474365380216401, -1.186317179420349, -0.673175106219423, -8.593828074468492, -2.068926620192420, 0.185845407477266, -3.538290521262840, -1.870268993856328, 0.581770669954075, 1.577223441103127, 0.245795178147093, -8.157924707128929, 2.280704744970723, 0.342946719541682, -3.910667664613060, -4.467098159905070, -0.906470628559765, -5.467201009223641, 2.737699080237925, -0.436040756714554, -2.751220019407433, 2.166604090475928, -0.332360011770856, 4.119701674973683, -1.387306658054850, -1.503227342773925, 0.337225781627659, 0.379643101898157, 3.206528771762287, 0.538784632366727, 2.035815138835933, -0.115118725771238, 0.027542935718282, -0.691806368825379, 4.213805017227664, -1.619310067100398, 2.993913991309019, -1.110027940567203, -2.401290055549762, 1.097750207941063, 2.059889964582283, 0.291951010740415, 0.021009871562727, 0.523301343580218, 0.092153840735425, -0.744370214455877, 1.201836947413143, 0.818701345453900, -2.438169814916017, 6.625546861602578, 2.057698805626764, 0.404151989425300, -0.810596612611141, -0.417001482418377, -2.263224200847616, -1.112434018439531, -2.481887152761845, -5.493412840188150, -0.146517807605973, -1.961941892540544, 9.295504754768023, -0.689178084323587, 0.469541661616307, -0.628441773939000, 0.920579328930262, -21.284085496690270, 11.420409660621774, 3.344858445371178, 4.026568188557498, -0.889681209493754, 6.473823258114644, 3.132240934634673, 0.123852668355637, 12.237856165375049, -16.681325278355374, 0.332214467182122, -0.669033601195795, 7.652479352172455, 1.230896328387997, -2.563091085309658, 0.513630989438070, -1.611824781019923, -0.660240611474350, 1.250993541691231, -0.208091212428763, -4.448997032834758, -1.500212905406674, 0.586139129590036, 2.860195069292906, 1.600441269928579, -0.343082605895435, 0.965863299195872, 0.150424946704961, -1.033629701613839, -0.452410322950436, -0.220067156383811, -0.789410502942839, -3.154548006475654, 51.499516671634048, -3.754432804177846, -25.271304132346856, 11.860051116233862, -9.471403772814437, 1.158911675559202, -6.456000427696317, 4.218408224499989, -15.397677271144568, 11.025180101622773, -15.477260366495228, -8.564109879875261,

-6.675919970037072, 26.985834298949875, 18.642277185087849, -24.349616689535079, 3.867264750129841,
-3.021778095602352, 12.294197653675246, -6.557067670977562, -4.800357848543951, -28.264999456324553,
-17.575354719930573, -5.787970513964607, 2.983097154757307, -17.946588357835033, 12.293603495741806,
0.258603746792891, -11.076955370327202, -0.319088555554144, 16.719150833145672, 18.644101987387845,
4.267747965615421, 88.311854131870248, -2.328805978785411, -3.730736934714884, -3.535873456306964,
0.499417169099167, 6.555612829056063, -32.242505955889605, 1.487911161724635, 13.147279057526045,
-42.205604404419596, 9.546783288426134, 0.146338643375272, -11.937382198715465, -52.230525946869989,
-3.058568472538358, -8.220793532302393, 10.303742562476405, -9.271978425450637, 9.126835711013809,
-3.840690585342728, -1.143172168868982, -32.937152117539732, -4.940861815337594, 13.418707402389666,
-9.874032664589759, -1.581527065601002, -10.933313240448390, 41.121469034294378, 2.911953754543154,
-24.825962197130632, -7.008868765826922, -9.619682766587031, 0.159367503682150, 4.068205562060452,
35.543548274371155, 2.409609647064563, 0.326821442220716, -3.945595721997502, 6.920025068663102,
-11.121028549358504, 0.095470249496158, 18.768988595654751, -103.241827218111200, 13.148238823328747,
-7.039137013075766, 5.761103538183644, 1.051643974298734, 4.356613090310721, -23.993861808503681};

5.6 MPBnnEval.c - From Results in Section 3.2

```c
/**
 Generated by Multiple Back-Propagation Version 2.2.4
 Multiple Back-Propagation can be freely obtained at http://dit.ipg.pt/MBP
*/

#include <math.h>
/**
 inputs  - should be an array of 16 element(s), containing the network input(s).
 outputs - should be an array of 4 element(s), that will contain the network output(s).
 Note : The array inputs will also be changed.Its values will be rescaled between -1 and
    1.
*/
void MPBnnEval(double * inputs, double * outputs) {
  double mainWeights;\\ARRAY INITIALIZED IN THE FOLLOWING APPENDIX
  double * mw = mainWeights;
  double hiddenLayer1outputs[20];
  int c;

  inputs[0] = -1.0 + (inputs[0] - 21.573288999999999) / 4287.211465999999700;
  inputs[1] = -1.0 + (inputs[1] - 22.532620999999999) / 959.162792999999970;
  inputs[2] = -1.0 + (inputs[2] - 14.000000000000000) / 16.500000000000000;
  inputs[3] = -1.0 + (inputs[3] - 3.000000000000000) / 39.500000000000000;
  inputs[4] = -1.0 + (inputs[4] - 7.401209000000000) / 2176.857930000000100;
  inputs[5] = -1.0 + (inputs[5] - 8.623840000000000) / 945.550248000000010;
  inputs[6] = -1.0 + (inputs[6] - 14.000000000000000) / 16.500000000000000;
  inputs[7] = -1.0 + (inputs[7] - 2.000000000000000) / 37.000000000000000;
  inputs[8] = -1.0 + (inputs[8] - 32.803324000000003) / 790.981097500000030;
  inputs[9] = -1.0 + (inputs[9] - 37.851123000000001) / 1588.695454500000100;
  inputs[10] = -1.0 + (inputs[10] - 17.000000000000000) / 14.500000000000000;
  inputs[11] = -1.0 + (inputs[11] - 15.000000000000000) / 33.500000000000000;
  inputs[12] = -1.0 + (inputs[12] - 12.524685000000000) / 1522.454503000000200;
  inputs[13] = -1.0 + (inputs[13] - 14.716892000000000) / 1559.156945000000000;
  inputs[14] = -1.0 + (inputs[14] - 21.000000000000000) / 13.000000000000000;
  inputs[15] = -1.0 + (inputs[15] - 13.000000000000000) / 33.500000000000000;
  hiddenLayer1outputs[0] = *mw++;
  for(c = 0; c < 16; c++) hiddenLayer1outputs[0] += *mw++ * inputs[c];
  hiddenLayer1outputs[0] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[0]));
  hiddenLayer1outputs[1] = *mw++;
  for(c = 0; c < 16; c++) hiddenLayer1outputs[1] += *mw++ * inputs[c];
```

```
hiddenLayer1outputs[1] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[1]));
hiddenLayer1outputs[2] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[2] += *mw++ * inputs[c];
hiddenLayer1outputs[2] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[2]));
hiddenLayer1outputs[3] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[3] += *mw++ * inputs[c];
hiddenLayer1outputs[3] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[3]));
hiddenLayer1outputs[4] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[4] += *mw++ * inputs[c];
hiddenLayer1outputs[4] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[4]));
hiddenLayer1outputs[5] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[5] += *mw++ * inputs[c];
hiddenLayer1outputs[5] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[5]));
hiddenLayer1outputs[6] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[6] += *mw++ * inputs[c];
hiddenLayer1outputs[6] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[6]));
hiddenLayer1outputs[7] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[7] += *mw++ * inputs[c];
hiddenLayer1outputs[7] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[7]));
hiddenLayer1outputs[8] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[8] += *mw++ * inputs[c];
hiddenLayer1outputs[8] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[8]));
hiddenLayer1outputs[9] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[9] += *mw++ * inputs[c];
hiddenLayer1outputs[9] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[9]));
hiddenLayer1outputs[10] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[10] += *mw++ * inputs[c];
hiddenLayer1outputs[10] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[10]));
hiddenLayer1outputs[11] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[11] += *mw++ * inputs[c];
hiddenLayer1outputs[11] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[11]));
hiddenLayer1outputs[12] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[12] += *mw++ * inputs[c];
hiddenLayer1outputs[12] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[12]));
hiddenLayer1outputs[13] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[13] += *mw++ * inputs[c];
hiddenLayer1outputs[13] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[13]));
hiddenLayer1outputs[14] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[14] += *mw++ * inputs[c];
hiddenLayer1outputs[14] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[14]));
hiddenLayer1outputs[15] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[15] += *mw++ * inputs[c];
hiddenLayer1outputs[15] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[15]));
hiddenLayer1outputs[16] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[16] += *mw++ * inputs[c];
hiddenLayer1outputs[16] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[16]));
hiddenLayer1outputs[17] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[17] += *mw++ * inputs[c];
hiddenLayer1outputs[17] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[17]));
hiddenLayer1outputs[18] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[18] += *mw++ * inputs[c];
hiddenLayer1outputs[18] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[18]));
hiddenLayer1outputs[19] = *mw++;
for(c = 0; c < 16; c++) hiddenLayer1outputs[19] += *mw++ * inputs[c];
hiddenLayer1outputs[19] = 1.0 / (1.0 + exp(-hiddenLayer1outputs[19]));
outputs[0] = *mw++;
for(c = 0; c < 20; c++) outputs[0] += *mw++ * hiddenLayer1outputs[c];
outputs[0] = 1.0 / (1.0 + exp(-outputs[0]));
outputs[1] = *mw++;
for(c = 0; c < 20; c++) outputs[1] += *mw++ * hiddenLayer1outputs[c];
outputs[1] = 1.0 / (1.0 + exp(-outputs[1]));
```

```c
    outputs[2] = *mw++;
    for(c = 0; c < 20; c++) outputs[2] += *mw++ * hiddenLayer1outputs[c];
    outputs[2] = 1.0 / (1.0 + exp(-outputs[2]));
    outputs[3] = *mw++;
    for(c = 0; c < 20; c++) outputs[3] += *mw++ * hiddenLayer1outputs[c];
    outputs[3] = 1.0 / (1.0 + exp(-outputs[3]));
    outputs[0] = 0.000000000000000 + (outputs[0] - 0.000000) * 1.000000000000000;
    outputs[1] = 0.000000000000000 + (outputs[1] - 0.000000) * 1.000000000000000;
    outputs[2] = 0.000000000000000 + (outputs[2] - 0.000000) * 1.000000000000000;
    outputs[3] = 0.000000000000000 + (outputs[3] - 0.000000) * 1.000000000000000;
}
```

5.6.1 mainWeights[] array from MPBnnEval.c - From Results in Section 3.2

mainWeights[] = /-11.253214618377667, 1.405354677002241, -1.182751200141889, 2.277561189520991, 1.763158632423368, 0.684687747577315, -18.466525591333788, 1.836937988561259, 0.585166232198536, 3.371473581121434, 6.900868209699747, -1.538320296123229, -1.459701452846419, -3.985308713680459, 1.581106245051505, 1.529383609996363, 3.487162769821830, -8.175494727678975, 0.415087441860305, -4.323719621694306, 2.367000374006180, -2.714826002833101, -0.503750729153865, -4.018795331205947, 3.249307035984373, 0.354824066340192, -4.247215119449519, -3.067244483621847, -0.616067739158539, -3.353949733210090, 0.988123785600619, 4.285726449690609, -6.607689884276313, -2.398153718006349, 0.072474145016710, -3.806784212078085, 0.903744710825808, 2.683527721229612, -0.580011966663043, -3.400123998036029, 0.044506628094318, 0.596245746808816, 3.449122583757569, -0.308705256433868, -2.233383563318430, 0.650928965199364, -1.764892069283057, 4.227321738531777, 5.543186825674336, 1.157267188686691, -1.768850824604713, 4.849399587286367, -1.427099617861895, 0.532259370734713, 47.558925494472248, -22.476925369990891, -0.832185329777165, -3.271774416055841, 19.078813758499276, -35.096983709908429, 18.361530093810661, -8.093619161996436, 7.644968287097925, 7.739239094892504, 2.284867436646646, 1.684394694361457, -10.998545725678042, -2.907031385853675, -58.252746382978437, 2.619785999108773, -19.558221810100569, -0.648042871124893, 2.746452818729318, -0.561371455015863, -17.818510426446899, -0.688548142946740, 0.456730341583131, -22.245090136817058, -0.842389460747250, 1.283696198124405, 0.164120999211527, -1.917831539864210, -0.642921242438560, -0.063973056509560, -0.795969863371034, -0.596165517989300, -1.803632149604503, 19.176613746030334, -18.859304802485003, 29.062888960978654, 0.353790612120144, -28.732251212013267, 2.327241134257889, -24.401866669567493, 9.331640538587468, -2.360838911945026, 27.818477582850996, 32.474648322249138, 1.221346542703764, -0.497674061731299, 32.140461120150171, -18.051379810516131, -0.585996846850707, 7.760364640519376, -0.335675148885665, 0.477559250813896, 4.535666980286793, 1.315611826188698, 11.609669679214740, -1.685804725441409, -0.498964411986992, -3.538900684675560, -19.744093729831988, 1.600160423500295, -6.587988000368772, 1.463316869540224, -1.137952616232421, 0.943425298697442, 3.241388455263011, -5.055289902655334, -6.672919520348843, -0.616973183284566, -27.270822524858215, -6.336965184760834, 0.346652474866662, -8.881576015202187, -3.543147766200346, -11.274104597816450, -2.342208317882011, -0.468565958805616, 6.262098308839680, -11.358089150315624, 1.142053618297381, -0.409739272802340, 2.893490986845031, -4.113337368723793, 2.629929299981558, -0.520976993648334, -0.130440504334296, -9.259136394027157, -7.337530096163541, 0.128465902033073, -0.223325569190465, 11.712085733254593, -4.069770072315802, 5.593710191006419, 1.474846298839452, -4.097565286591094, -5.378158134775157, 0.037717108717175, -2.570241878259667, -7.749663975901663, 9.410438138457144, 5.235514142908440, -0.973164544882033, 1.645265378718355, 0.832066477041075, -3.720258731415364, 0.021934733561943, 7.658114727527002, 3.055152720023364, 4.255282829238984, 2.147581324346163, -0.216448157889054, 1.365672368247813, 1.011616983572298, 1.202486943438489, -1.107484399429273, 1.723543821480152, -1.132370497316068, 4.909371664480418, -1.532717903084125, 1.829669924868184, 1.378432722201081,

-0.777631934256585, -0.466589455643995, 6.937398975881122, 0.687920770190908, 0.987133037935800,
-2.336788192512186, -2.850261528690506, -1.438332930050715, -1.807559740776733, 2.869521962796848,
-0.734597539842074, 0.300734317688178, -1.546418739736867, -6.439681154801157, -1.203799485336424,
2.259659393085389, 14.760395116782528, 0.667152959318119, 7.369923863713987, -0.614687572499752,
4.822294033190837, 10.427132955417896, -1.474026774387041, 16.118205254926441, 13.751533769508034,
-14.929330141348071, -11.335429946577598, -2.966487512901178, 10.726816078027834, -19.269964082681980,
-1.566346911136094, -2.876751875337904, -11.866267529750132, 0.478483927406223, 1.864697178321802,
3.255139361924961, 33.374551823882967, 0.202312535632734, 0.320449470780405, -13.900829922514907,
-47.671711937168659, -1.166577073746478, -2.483333716984405, 13.236207713736471, 26.000628748565241,
-0.423751281874246, -0.025479066803670, -3.866700014495997, 12.761612936288433, -1.983077150295160,
1.527536070927071, -44.943473972512407, 0.843630643491530, -13.381484465142844, -55.480992605872054,
0.873978387297013, 10.128304697851229, 2.131405732562643, 10.528968372702721, 2.480224592661095,
-3.768809348869870, -1.277103707891064, 1.339050476195807, 19.456076320903261, 2.888280247775007,
3.599502734784134, -16.735932233790670, -9.688079733825070, 10.091717128179500, 0.705479468883028,
-3.964951809137887, 9.255961926694013, -6.486383932902069, 4.066808066694938, -5.127722202216567,
-15.953754756252426, -1.858263493563977, 5.965828979556429, 9.177266742715737, 5.811577769566685,
10.326576485157000, 3.447228136257702, -19.964948426847673, 0.425228872584031, -4.016681496245288,
8.867060866003662, -0.032911206428359, 24.998097411190507, 2.797695031052204, -3.659865958862204,
-4.521924156930050, 2.212336174128684, 6.394589791407523, 9.276171099593071, -1.241300015110413,
-8.451143930838965, -0.165619985263837, -14.317510065079174, -0.598045890908657, 3.069412031969292,
3.687467217654200, 0.692879948102243, 0.057422056013353, -0.664266192905896, -1.699950883024146,
-0.291946268100390, 0.756033674983505, 5.128512693030983, 3.465817340361217, -0.811809903221358,
-1.847914205241645, -4.040798847617000, -10.555652988880869, -0.403545174447597, -1.232283557899639,
1.724003825045724, -1.698829732683582, -1.505822904943960, -20.828839048640823, 5.942372762379324,
-1.734110991227174, 2.782653415888373, -7.297417383444958, 15.653083737606725, 0.063919894989207,
-0.097589912401792, -10.888019047525924, 4.702536743432335, 2.742314967911190, 5.167187958096136,
-0.006174578401743, 3.476944247886013, 0.525827390498628, 0.553996386868137, -3.192742945376964,
6.396736412198708, -7.865053989676314, 1.734601093415611, 2.059723329855185, -17.106483326576232,
10.682458741698973, -2.136309856557695, -6.965904474144568, 14.783494754798935, -7.354173086471006,
5.141362154143047, 5.073335895082181, -1.775469186159960, -15.933516543426983, 1.594851044106998,
-0.502928852178154, 1.608704693100281, 18.920043981314659, 8.203841962655078, -0.521331680887361,
0.985623962264311, -6.510798797687107, 9.922492851442691, 2.777039218314766, 2.494193933027857,
2.109833312883039, -2.905418001268204, -1.366374860013884, -2.227735218212068, -0.724523167951651,
-15.841707076423573, -1.959782916063858, -0.766067721446295, 16.660110049090452, 10.672739123572137,
-3.679208244892513, 77.818773439970528, -3.022392263831677, 5.147541987331866, -1.000270975495277,
-10.482439417191106, -30.173584381075120, -10.863440555023303, 18.761118341585309, -48.690945885518964,
-8.100686624856985, -5.030713100642368, -5.663396981208430, 2.818298214542264, -2.776716786954450,
-7.125162807691914, -8.730952147081654, -6.246147465280541, -22.103726974156391, 28.791582991111564,
36.458724089408754, 5.073626993477267, -23.699872887264419, -0.266594435090565, 21.212670378166752,
4.019823302524465, -9.497299262533632, -14.736875906378980, 6.975875001685216, -16.208953045372922,
29.557908831579564, 1.738032029059101, 9.845546797097653, -4.683771635536272, -83.831350939350386,
12.903140769612197, 2.861680290833291, -7.003579084265959, 0.772209414281079, 45.331781221809628,
41.468666659576463, 92.620759556662023, 24.409511530980609, -66.817096577766648, 15.915800962284910,
-54.792083208445774, 21.363588066751344, -19.504173932986731, -48.558129768756423, -79.498561518917853,
-32.074725378451035, -61.961138378210507, 6.458821882292659, 18.244885370075050, -27.408286521109456,
-15.833270954772699, 26.955350344688693, 22.059577095335424, -18.375593694099695, 1.753451974404965,
-43.516648161912521, -38.717821554443475, -90.120162943052875, -24.168023945919941, -43.011743000298395,
-15.122925483117141, 53.135841737386876, -20.270962791773321, 17.901181697546580, 46.751122942678073,

77.649119518224950, 28.939908847824459, -100.998030722326060, -6.243115135324385, -17.654192647423574, 26.384680276131391, 12.666988152305956, -25.906945615500184, -21.109544620362232, 17.619410871179831/;

5.7 beaglebone_gpio.h

```c
/*
beaglebone_gpio.h By: David Nahmias
Electromyography data acquisition and analysis program for recognizing electromyography data via an Artificial Neural Network to drive GPIO signals.
Written by: David Nahmias
For Engineering 90 Senior Design project at Swarthmore College.

This code defines the memory locations of the GPIO in Mode 1 used in the main recognition C file for HDI control.
*/

#ifndef _BEAGLEBONE_GPIO_H_
#define _BEAGLEBONE_GPIO_H_

#define GPIO1_START_ADDR 0x4804C000
#define GPIO1_END_ADDR 0x4804DFFF
#define GPIO1_SIZE (GPIO1_END_ADDR - GPIO1_START_ADDR)
#define GPIO_OE 0x134
#define GPIO_SETDATAOUT 0x194
#define GPIO_CLEARDATAOUT 0x190

#define GPIO8_11 (1<<13)
#define GPIO8_12 (1<<12)
#define GPIO8_15 (1<<15)
#define GPIO8_16 (1<<14)
#define GPIO8_26 (1<<29)

#define GPIO9_12 (1<<28)
#define GPIO9_14 (1<<18)
#define GPIO9_15 (1<<16)
#define GPIO9_16 (1<<19)
#define GPIO9_23 (1<<17)

#define USR1_LED (1<<22)


#endif
```

5.8 startup.sh

```bash
#!/bin/bash
dtc -O dtb -o BB-SPI1-01-00A0.dtbo -b 0 -@ BB-SPI1-01-00A0.dts
cp BB-SPI1-01-00A0.dtbo /lib/firmware/
echo BB-SPI1-01 > /sys/devices/bone_capemgr.*/slots
optargs=quiet drm.debug=7 capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN capemgr.enable_partno=BB-SPI1-01
dtc -O dtb -o BB-SPI0-01-00A0.dtbo -b 0 -@ BB-SPI0-01-00A0.dts
cp BB-SPI0-01-00A0.dtbo /lib/firmware/
echo BB-SPI0-01 > /sys/devices/bone_capemgr.*/slots
optargs=quiet drm.debug=7 capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN capemgr.enable_partno=BB-SPI0-01
```

```
depmod -a
echo 60 > /sys/class/gpio/export
echo high > /sys/class/gpio/gpio60/direction
echo 1 > /sys/class/gpio/gpio60/value
echo 0 > /sys/class/gpio/gpio60/value
```