HawkEar 2.0

Sam Blanchet, John Hatfield, Dimitri Wolff, Jack Myers

Table of Contents:

1 Introduction

The HawkEar 2.0 is part of a joint project headed by Dr. Lauren Kloepper at St. Mary's College, and Dr. Robert Stevenson at the University of Notre Dame. The goal of the project, is to collect information about the echolocation signals of Mexican free-tailed bats. These bats come out of caves in swarms of over a million, and must communicate with each other to avoid colliding. They do this by transmitting and receiving intense (100-120 dB), short (~10 ms), ultrasonic down-sweeps from ~60 kHz to ~20 kHz. Little is known, however, about how the signals don't interfere with each other, and how the bats are able to echolocate the others, all while flying at high speeds and densities. Furthermore, the bats produce different signals when threatened by a predator to warn the others in the colony of the danger. Biologists tentatively suggest that bats produce more of a tonal signal for these warning calls, rather than the normal frequency downsweep, but that has not been confirmed. Recording the bat calls in different contexts will hopefully shed more light on exactly how they communicate and process the signals. This unique ability is of interest to the U.S. military, as it could be applied to drone swarm technology.

The HawkEar 2.0 is a continuation of a senior design project from last year, the FALCON pack. The data from the bats is captured with microphones, processed, and stored onto a microSD card. The signal processing circuitry is strapped to the back of a Harris Hawk. The hawk is flown through the swarm of bats, and the equipment records the signals of the bats. Last year's team developed the original idea of two boards, on the neck and back of the hawk, to capture the signals of the bats. This project was

successful. However, Dr. Kloepper required more data, and better data to be able to better understand the bats. This year, our senior design group was tasked with redesigning the boards, and providing software upgrades that accomplished these objectives. Our redesigned recording system is called the HawkEar 2.0.

There were two main hardware upgrades which we achieved this year, that gave our system greater capabilities, and made it more user friendly than last year's version. One of these was the addition of a second microphone. The addition of the second microphone allows the data to be analyzed for directionality of the signals being recorded. This makes it easier for Dr. Kloepper to determine which bat signal belongs to which bat. To meet this objective, our team added a second microphone to the board on the hawk's head. This meant that we also had to add another filter, amplifier, and other associated components to process and store the additional signal. The addition of the second signal had implications for the software as well. Since we were using one ADC to digitize both signals, the ADC had to sample twice as fast,near the limit of the speed at which microSD files could be written. The memory writing will be explored more in depth later in the report.

The second major upgrade which our team accomplished this year was the simplification of the device's gain control system. When using the device last year, Dr. Stevenson needed to change the gain at times in the field. With last year's technology, he had to open up the software, and change gain values in the code each time he wanted to do this. There was also an automatic gain control, but Dr. Stevenson found that it was not very useful. This year, our team simplified the gain control. We added

three binary dip switches. These dip switches could be arranged to activate eight different predetermined gain modes. These dip switches were inputs on the microcontroller, and they told the microcontroller how to set the output for the gain. This design greatly simplified the gain control, and increased ease of use.

The HawkEar 2.0 met the basic goals that we set this year, however there were a few things that we had hoped to achieve that we could not. Dr. Stevenson expressed interest in switching our memory storage from an SD card to a flash memory chip that could be attached directly to the board. Unfortunately, we were unable to find a chip that could hold enough memory while also writing data to memory at the speeds that would be required. Another stretch goal that we had set was the addition of a UART transfer protocol to transfer the data directly from the board to a laptop without having to remove the SD card and insert it into the laptop. We determined that with the data transfer speeds of our device, it would take roughly 15 minutes to transfer the data from a 20 minute flight. This was too long of a wait to implement in the field, so we decided to stick with just removing the SD card. These two are issues that could possibly be investigated further by a future team.

2 Detailed System Requirements

- Gain control will be determined by 3 dip switches. Each dip switch will correspond to a single digit on a 3-digit binary number, allowing us to include 8 different gain options. This will reduce complexity, and is sufficient for what we are aiming to accomplish. Upon a device reset, the device must recheck the dip switches to determine the appropriate gain.

- In order to solve the sound localization problem a second microphone on the hawk's head will be included. The microphones will be placed as far apart as possible to allow the best possible sound localization. Both microphones must sample at at least 200 ksps to achieve required Nyquist sampling speeds. They must also be able to pick up frequencies in the range of 10-100 kHz, and record at amplitudes up to 120 dB.
- The device must be able to create, open, and write a new file to memory each recording session. The memory writing will be done to a microSD card using SPI. Memory writing must occur fast enough to ensure no data loss/corruption.
- The ADC system must sample at 400 ksps so that each of the two microphones samples at the necessary 200 ksps. The ADC must also switch between microphone inputs each ADC conversion.
- The MATLAB program used to analyze the data must be able to split the file data so that the left and right microphone data are clearly distinguishable. The program must also display a frequency spectrum of the data from the left and right microphones, as well as their relative voltage levels, and an audio output of the collected data.
- The device must be able to retain the majority of the data on the microSD card in the even that power is lost in the middle of a recording.
- The battery must power the device for approximately 45 minutes to an hour.
- The device weight must be under 50 grams.
- The device must consist of two seperate boards. One smaller board to be placed on the Hawk's head must contain the microphones. The second board will be placed on the hawk's back, and should contain the larger components such as the dip switches, microSD card, and microcontroller.

3 Detailed project description

3.1 System theory of operation

The user of the HawkEar 2.0 simply attaches the rig onto the hawk's training harness and plugs in the battery. This should cause a red power led to turn on indicating that the device is ready to begin audio recording. Next, the user presses a button on the board that initiates audio recording. Pressing this button will cause a green LED to turn on, to signal that recording has begun. Once that has begun, the two microphones will continuously send analog audio data to the two AD8338 amplifiers. These amplifiers filter the signal through a bandpass filter between 10-100 kHz. The signal is then sent through wires from the neck board to the PIC32 microcontroller, which is on the back board. The PIC32 will create a new file on the microSD card and convert the microphones' analog output to digital data by sampling the incoming signal at a rate of 400 kHz. This data is then dumped into the newly-created file within the microSD card in chunks of 16384 bytes using Serial Peripheral Interface (SPI). After the hawk flies through a swarm of bats, it returns to the owner, who presses the record button again to stop recording audio. The green LED blinks, and then turns off to signal that recording has ceased and the file has been closed. The microSD card would then be ejected from the board and inserted into an external computer for data processing and analysis using a MATLAB program which graphs both the frequency response and the relative voltage levels of the signals from each microphone.

3.2 System Block Diagram -

The following shows the overall system block diagram and how it is divided into subsystems:

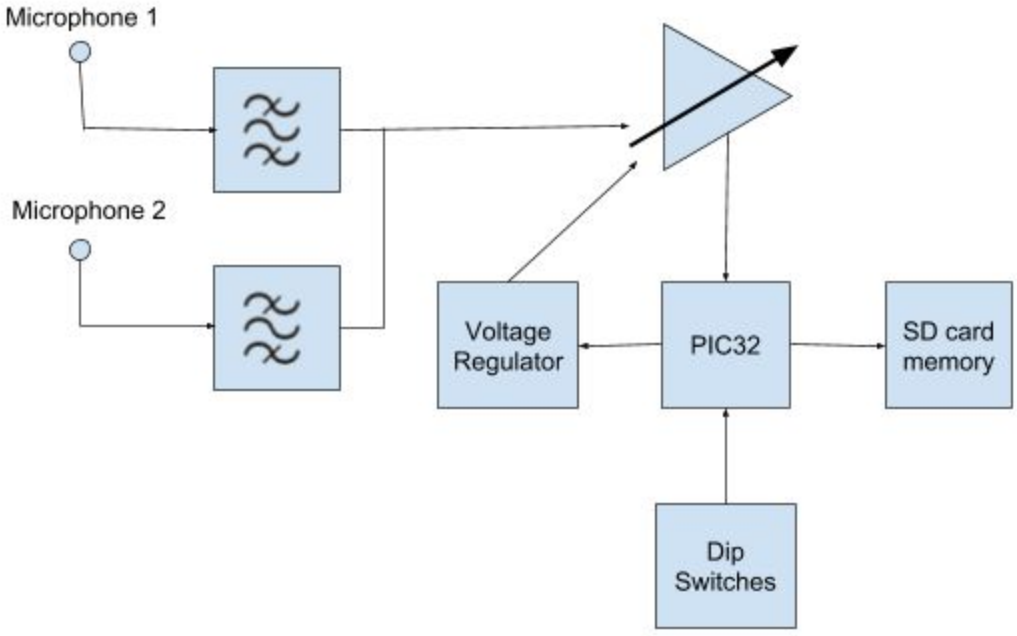

Figure 1: System Block Diagram

1. The two microphones pick up the high-frequency signal produced by the bats and output an analog signal that is fed into a bandpass filter allowing only frequencies between 10 kHz and 100 kHz. The signal then goes into a variable gain amplifier (VGA).
2. The VGA takes in the signal and adjusts its amplitude according to the CVRef voltage set by the microcontroller dip-switch-controlled variable gain mechanism.

3. The microcontroller samples the analog signal to produce a digital signal with its ADC. This signal is then written onto a file created within the microSD card.
4. The microSD card stores the digital data fed to it by the PIC32, stores it in the file, and closes the file when the data collection ends.

3.3 Detailed Design and Operation of ADC System

The Analog Digital Conversion (ADC) of our project had to be modified from last year to include the second microphone being sampled, while at the same time maintaining a sampling speed of 200 ksps per microphone to avoid data loss. For the latter problem, we utilized an ADC interrupt which contained a double buffer. Each ADC1 buffer contains 8, 32-bit buffers, which allowed us to copy buffer data into a global array when an ADC interrupt was fired, while still being able to sample ADC data into the other buffer while the interrupt was being serviced. The index of the global array was determined by a global index variable, which was incremented after each buffer was read into the global array. In order to sample two microphones at the same time, we had to increase the sampling frequency to twice the necessary Nyquist frequency, or 400 ksps. Microphone switching was done on MUX A, utilizing pins RA0 and RA1. The switching was done after each ADC sample, meaning that each time an ADC buffer was filled it contained alternating data from each microphone. Separation of this data was done using simple MATLAB code to break apart the input data array into two smaller arrays, each containing data from a single microphone.

The ADC sample form was set to output a 16-bit integer, but upon testing the ADC with the SD card memory writing, we realized that the ADC collection was

happening so fast, an ADC output array would get filled up before the SD card writing protocol had finished writing the previous array to the SD card. In order to avoid this and prevent data losses, we decided to only write 8 bits of each 10-bit ADC sample. This allowed us to reformat the ADC output array be an array of 8-bit unsigned chars as opposed to 16-bit shorts. By doing this we halved the size of each ADC output array which was being written to the SD card, and only lost the 2 least significant bits of each sample. In order to capture the 8 most significant bits of a sample, each ADC buffer was shifted two bits to the right, so that the 8 MSB's were located in the first 8 bits of the buffer. Therefore, when we copied the ADC buffer into a global array of unsigned chars, only the 8 MSB's are copied and the other 8 bits of the sample are discarded. A flowchart of the ADC ISR can be seen in **Figure 2**.

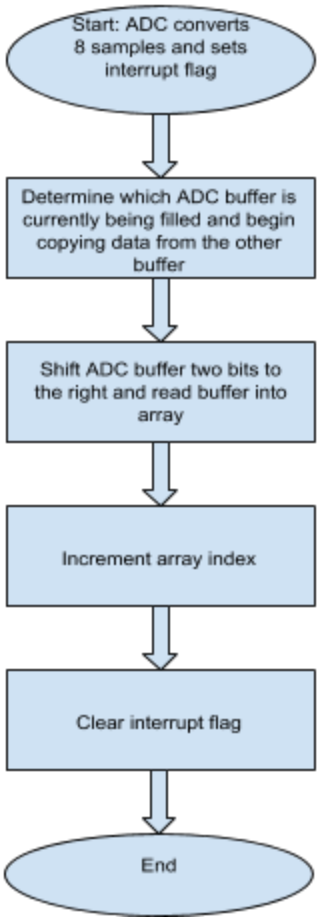

Figure 2: ADC ISR

In order to test the ADC sampling, we had to break its functionality into seperate parts and test each individually. In order to test that the microphone switching was working, we used a breakout board, and hooked one analog input to ground and the other one to a signal generator generating a ramp function from 0 to 1.6 volts. We then used the debugger in MPLAB to view the values stored in the ADC array after it had been filled. As expected, the values alternated between 0 and and numbers increasing from 0 to about 500. This was expected because 1.6V is about half the max input

voltage for an ADC sample and 500 is about half of the max value of a 10-bit ADC, which is 1023. In order to test that the sampling speed was 400 ksps we used last year's board, and modified the ADC initialization code to make it sample at 400 ksps. We then used a GPIO toggle which was fired every ADC interrupt to view how fast the ADC was sampling. Since we used the same exact microcontroller as last year, we assumed that sampling speed is the same as the verified speed on last year's board. We were unable to test our current board with the same method because the addition of additional pins for the second microphone left us with no unused pins which we could define as GPIO toggles.

3.4 Detailed Design and Operation of MicroSD Data Transfer

The Micro SD data transfer was probably the most difficult part of this project. We decided to use SPI because of our familiarity with the system and the fact that SPI was already implemented for the transfer protocol on last year's code. This code, which initialized the SD card and handled data transfers was implemented using the FatFS file system, developed by ChaN[1]. The implementation of the FatFS module was taken from last year's code, which utilized code by users 'Aiden.Morrison' and 'rileonar' on the Microchip forums[2]. Initialization of the SD card's SPI mode can be seen in **Figure 3**.

[1] http://elm-chan.org/docs/mmc/mmc_e.html
http://elm-chan.org/fsw/ff/00index_e.html

[2] http://www.microchip.com/forums/m563218.aspx

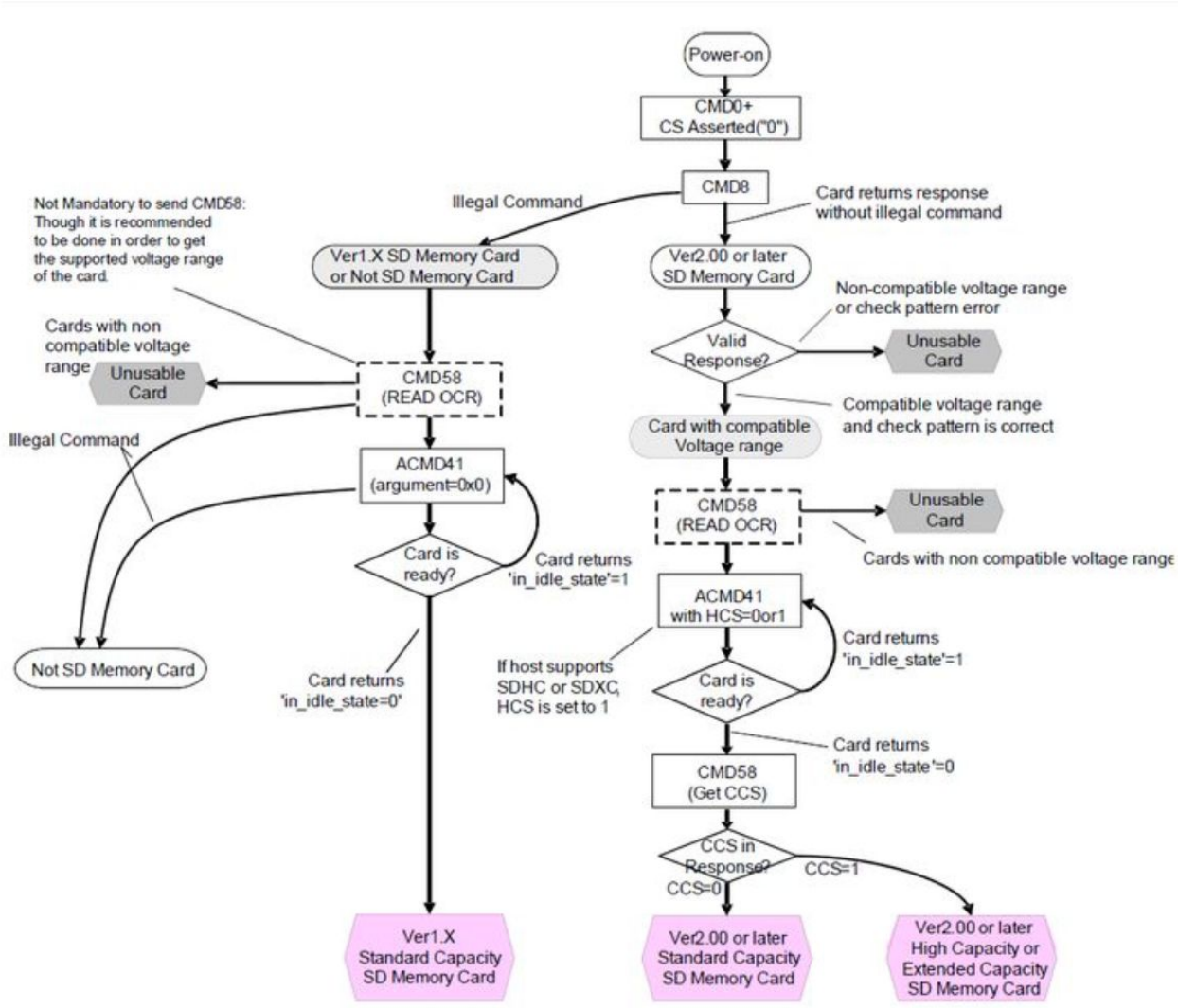

Figure 3: SD Card SPI Initialization

The basic operation of the file writing code involved creating/reading a file called FILESET.DEX, which contained an array of numbers which was incremented after each new file was created. This allowed us to name each new memory file with a number that made it easy to distinguish which file any given data collection cycle corresponded to. After the appropriate index was determined from FILESET.DEX, a new .DAT file was

created with that index number, and subsequent memory writes were directed at a pointer to that file. Each memory write occured after 16384, 8-bit entries were placed in the global ADC array. While this block of data was being written, ADC collection was switched to a second global array to ensure there was no data loss during an SD card write. After recording is finished we close out our file and wait to begin writing a new file.

As stated in **Section 3.3**, we encountered problems with the speed at which data writing occured. Initially, we tried to rework the underlying FatFS code to speed up data transfer but this was unsuccessful. In the end, we decided that we would only write 8-bits to each ADC array entry instead of 16, and this therefore halved the size of each file we wrote, which made up for the fact that we had doubled the sampling frequency from last year's project. We also utilized a f_sync() command to avoid losing data in the case of the system losing power during a recording session. This command performs the same commands that occur when the file is closed, but without actually closing the file. Having this command occur after each data write ensures that a loss of power will only result in at most 16384 ADC results being lost.

Testing for the memory writing system involved hooking up MOSI and MISO pins to a logic analyzer. When we initially tried to write 16-bit results while sampling at 400 ksps, we found that there was no gap in between blocks of memory being written. This meant that an f_write() command was beginning to write a block of memory before the previous block had finished writing, causing data losses. After we switched to 8-bit ADC results, we found that there was a gap between block writes, indicating that the memory writing was occuring at an acceptable speed relative to the sampling frequency.

3.5 Operation of Gain Control

In the first version of the project it was required to reprogram the microcontroller every time they needed to change the gain. The research team was in the middle of the New Mexican desert, about an hour from where they were staying. The environment was very detrimental to both coding and taking the device off the hawk. Due to these difficulties that came from the environment Professor Stevenson requested the ability to manually change the gain of the amplifier in the field without having to reprogram the device. We set up the specifications for the gain control based on 8 predetermined gains that are controlled by 3 seperate dip switches.

The 3 switches are designed by using the internal pull-up resistors of the microcontroller. When the switches are closed the voltage goes to ground and a 0 is read. When the switches are open the internal pullup resistor causes the voltage to be read as a 1. These combinations of 0's and 1's make up the 8 combinations of gains. The microcontroller reads the three inputs and outputs to the reference voltage of the microcontroller. The amplifier takes these different input voltages and the amplification/gain changes based on it. The software is primarily driven by setting input and outputs of ports/tris as well as using if statements. This can be seen in the block diagram **Figure 4.**

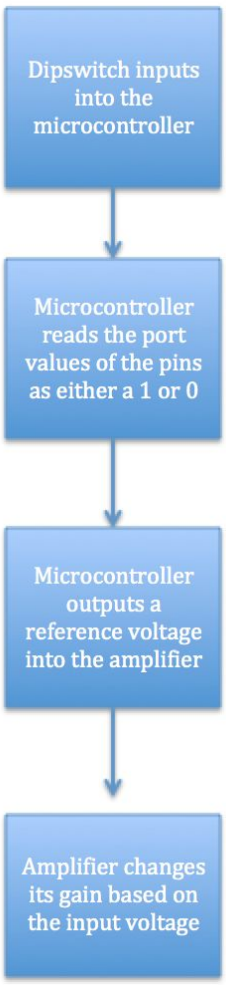

Figure 4: Gain Control Subsystem Block Diagram

The 8 gains that are being used are 5.6, 11.2, 13.6, 21.6, 32.8, 44, 55.2, and 65.6. In order to test this subsystem independently of the others, I used a breadboard as well as the kit pic 270 board. The breadboard had the ground connections to one side of the dip switches. The other end of the dip switches were connected directly into input ports. In order to test if the software was working I would measure the CVref

output on the pic and see if the voltage would change when I changed the dip switch combination.

3.6 Operation of Microphone

We are using Knowles Sisonic MEMS microphones for our audio design. These have a flat frequency response that is within the 10 to 100 KHz range that is necessary to capture the bats signals. We connect this to a capacitor in order to create a passive filter. We use two separate microphones, separated as far as possible on the neck board to capture audio signals. The microphones are AC coupled with amplifiers which has an input voltage from our manual gain control. The capacitors have a bandpass filter that is within the 10 to 100 KHz range that was previously mentioned. Class I capacitors were used so that they could compat any sound waves that the capacitors would emit. **Figure 5** is the interface that the circuit and the microphones had.

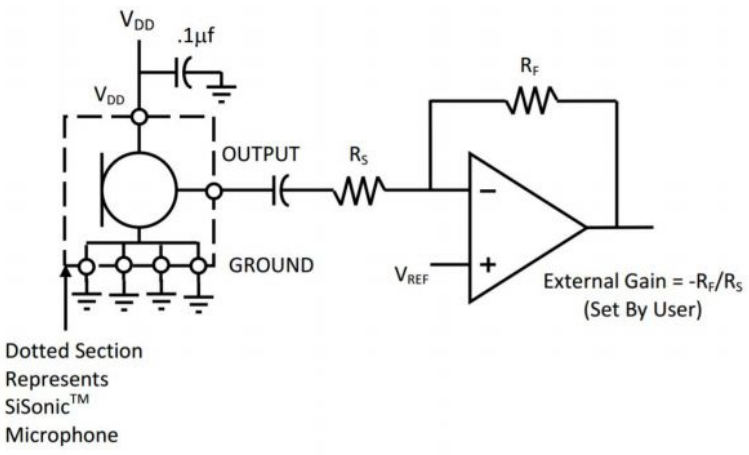

Figure 5: Microphone Interface

Bats typically produce frequencies between 15 and 60 KHz. They can produce frequencies higher than this though, which is why we needed a band pass filter from 10 to 100 KHz. The frequency response of the microphone, seen in **Figure 6,** shows that the microphone has no gain below 10 KHz which means it will not pick up sounds that humans are the source of.

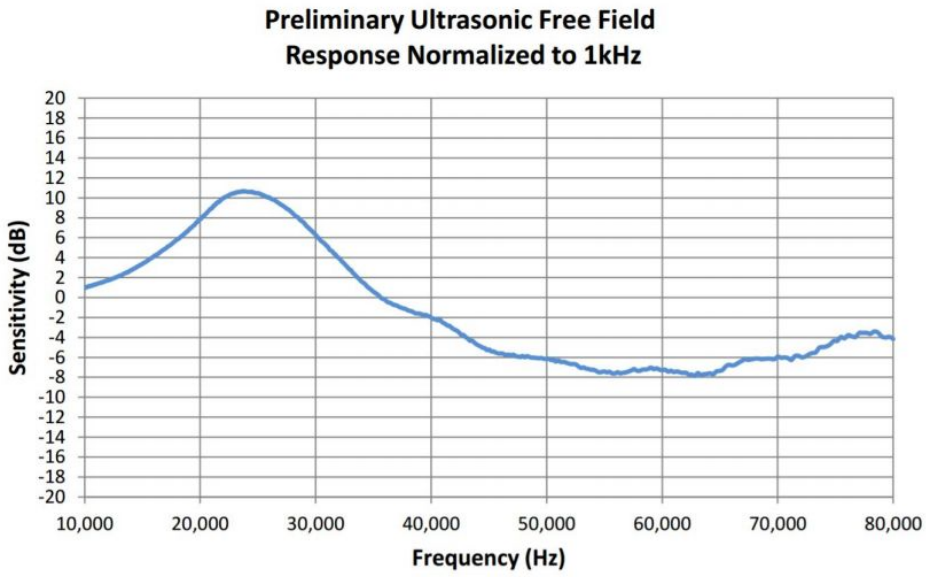

Figure 6: Microphone Frequency Response

We tested our system using a chirp generator that allowed us to generate known frequencies and measure them using our system. We could then plot the data to MatLab and see the various voltage levels and fourier transforms of the frequencies.

3.7 Interfaces

The following is a block diagram demonstrating the interface between the various subsystems:

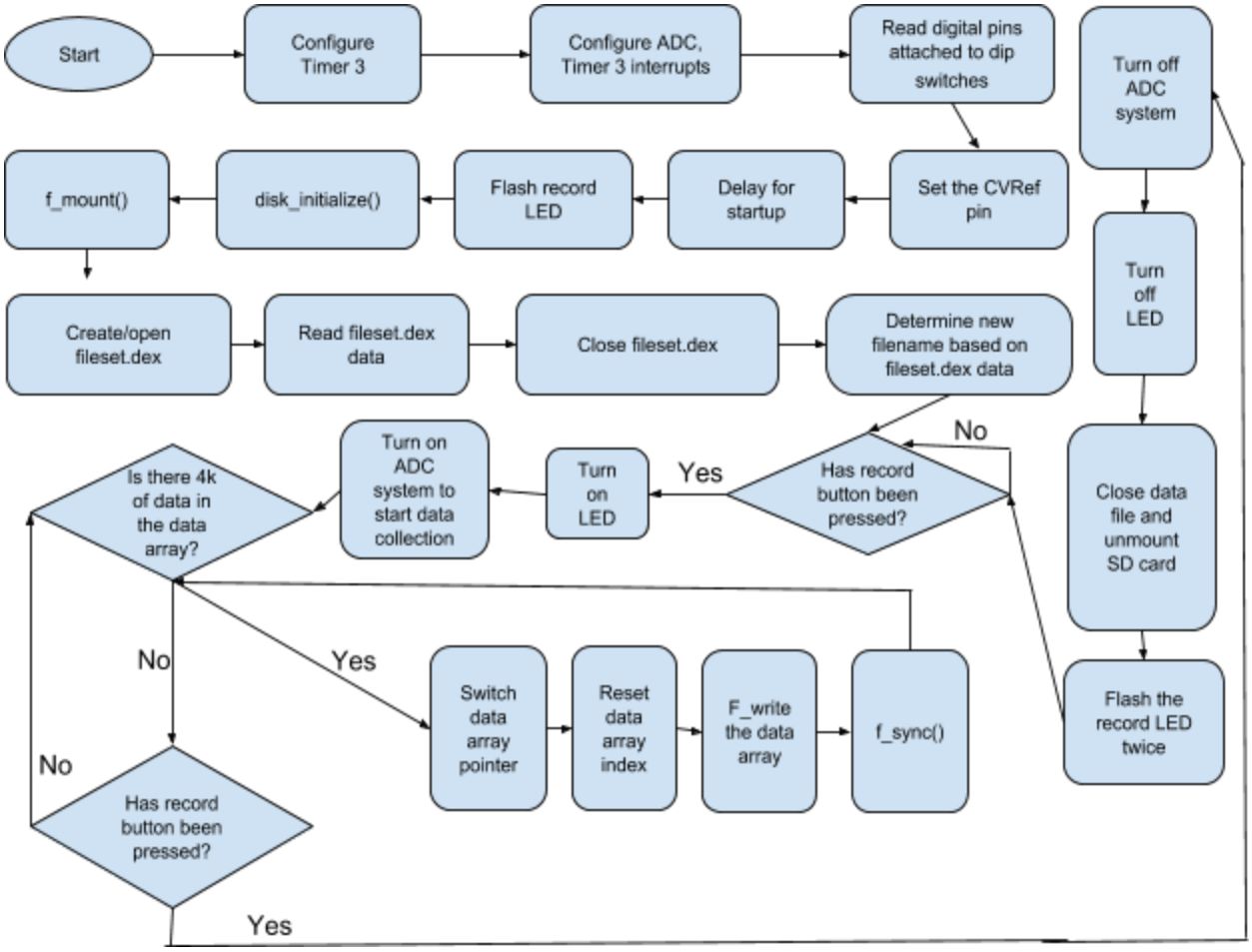

Figure 7: Overall System Interface Diagram

4 System Integration Testing

Since our project only accomplishes a singular task, writing to SD memory, we only had to demonstrate that functionality to prove that the whole system worked. By successfully writing a file to memory which was the appropriate size for the duration of the recording, we could demonstrate that ADC collection worked at 400 ksps. Further, by running this data through MATLAB and seeing what we would expect for the specified input frequency and gain, we could confirm that the gain control and memory writing also functioned properly.

5 Users Manual/Installation manual

5.1 Setup & Installation

HawkEar 2.0 consists of two boards and a series of wires connecting the two. As shown in Figure 1, the larger of the two boards is the “Back Board” and contains the parts that make up the physical user interface, such as dip switches, SD card, power and record button, etc. The smaller of the boards is the “Neck Board”, shown in Figure 2, and contains the microphones, amplifiers, and other parts responsible for recording the bats. Both boards should be attached to the falcon’s training harness with a custom-made case. The Back board should then be placed on the falcon’s back, and the Neck Board on the top of the falcon’s head alongside the camera module. Before any recordings can be made, the micro SD card must be inserted into the SD card slot on the Back Board.

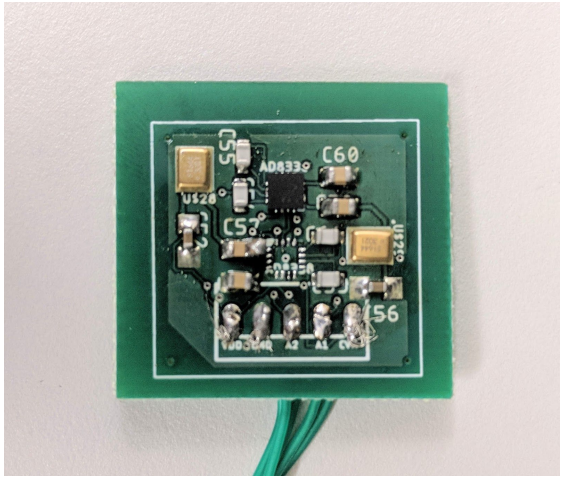

Figure 8: Neck Board

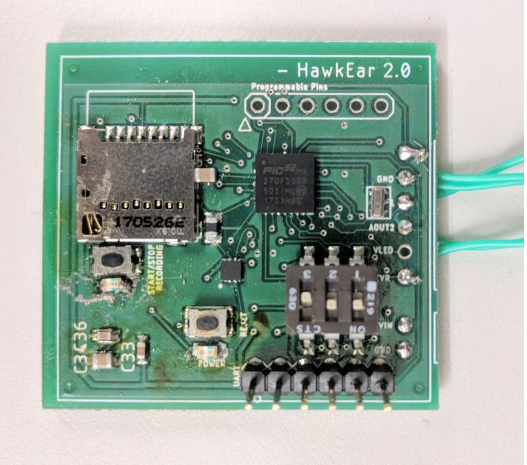

Figure 9: Back Board

Audio Data Collection System:

Operating HawkEar 2.0 to collect audio data the field can be done by following the step-by-step instructions below:

1. Power up HawkEar 2.0 by connecting a lithium-polymer battery to the red & black connector on the larger Back Board. The battery should be rated between 3.3V and 10V. Once power is supplied to the boards, a single red LED labelled “POWER” will light up on the back board.
2. Insert microSD card into the SD card slot before beginning data collection.
3. Use the 3 dip switches to set the desired gain before attempting data collection. The switches allow for 8 different gain levels, which can be attained with the switch configurations listed in Table 1 below.
4. If the falcon and user are ready for data collection, the user must press the button labelled “START/STOP RECORDING” on the Back Board, which will light up a green LED adjacent to the button.

5. Once done with the recording session, the user should press the same START/STOP RECORDING button to cease audio recording. The adjacent LED should blink, then turn off, but the POWER LED should stay lit.
6. Data is done writing and the file is ready for viewing in MATLAB.

File & Data Viewing:

1. Copy the data file into the same folder as the Matlab script "readSoundData.m".
2. Open "readSoundData.m".
3. Change the variables at the top of the readSoundData.m file in order for the program to read the data file.
4. Type a desired name for the .wav file to be outputted
5. Run the Matlab Script.
6. The .wav file of the data is created.

Note: If no edits to the .wav file are desired, change the "makeWaveFile" to equal 0. If the user doesn't want the graphical waveforms and fast Fourier transform to be outputted, change makePlots to equal 0. If the user would like to adjust the sample numbers where the sync markers were made, they can adjust the "markerPoints" variable. The raw and corrected data along with the voltage and normalized data arrays are shown in the column vectors.

Note: If the user would like to reset the device, they can press the "RESET" button at any time.

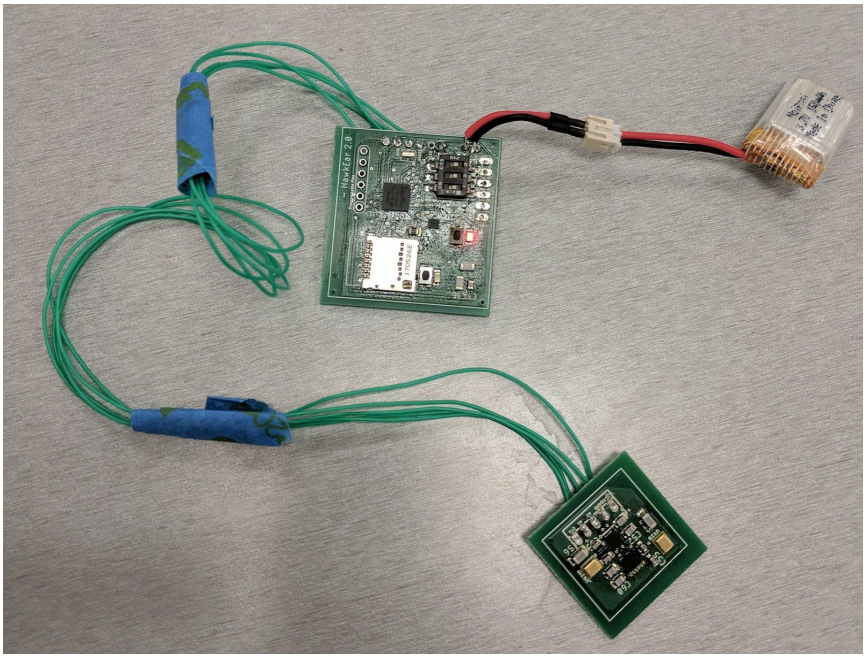

Figure 10: Battery Connected to HawkEar2.0

Switch 1	Switch 2	Switch 3	Voltage(V)	Gain(dB)
0	0	0	5.6	1.03
0	0	1	21.6	0.83
0	1	0	13.6	0.93
0	1	1	55.2	0.41
1	0	0	11.2	0.96
1	0	1	44	0.55
1	1	0	32.8	0.69
1	1	1	65.6	0.28

Table 1: Three Dip Switches outputting 8 Gain and CVR Levels

5.2 Verifying Operation

Once the battery is connected to HawkEar 2.0 a red LED labelled “POWER” should turn on and stay lit while the battery is connected. After the user starts a recording session a green LED next to the “START/STOP RECORDING button should turn on and stay lit

until the button is pressed again. If the device has responded to these inputs, it should be ready for use.

5.3 Troubleshooting

If the device does not power on:

1. Ensure that the battery is fully charged and fully inserted into the power adaptor
2. Check that the wires connecting the battery to the board are properly soldered

If the recording LED quickly turns off after the start of a recording session

1. Unplug the battery to cycle power
2. Remove and reinsert the SD card, ensuring that it is fully inserted into the holder
3. Reinsert battery

For further questions, email: sblanch2@alumni.nd.edu

6 To-Market Design Changes

While we accomplished most of the goals for our final product, there are changes that we would implement, if we had the time, before taking the device to market. The PIC32 microcontroller that we used, had every single IO pin utilized by our project. If we had used a larger microcontroller, with more IO pins, we could have possibly accomplished some of our stretch goals, such as UART data transfer. Using a two-channel amplifier on the neck board, rather than 2 one-channel amplifiers, would have reduced the number of components necessary for the neck board. That in turn

would have reduced the weight of the neck board, as well as the size. The other to market change we would make would be attaching some sort of SD Card holder that can lock the card in, so that there is no chance of it falling out while in use.

7 Conclusions

This project challenged the abilities of our team members to their limits. Almost none of what we had to accomplish had been taught to us in any of our courses, so much of the work, especially early in the semester, was just doing research on the datasheets of various parts. We essentially figured out how to finish this project using the internet. Working under a professor who actually is going to use our product in field-testing provided a level of seriousness that does not normally accompany undergrad assignments. This also meant that we had to learn to stay in communication with each other as well as all the stakeholders in this project. This project provided a great learning opportunity for the team which I think will help prepare us to become engineers in the real world.

8 Appendices

8.1 Parts List

Part Description	Source/Supplier	Part Number	Quantity	Cost/piece	Total Cost	Link	Manufacturer
Microchip	digikey	PIC32MX270F256B	4	$4.280	$17.12	https://www.digikey.com/product-detail/en/microchip-techno	Microchip
SD Card holder	digikey	WM6357CT-ND	2	$1.920	$3.84	https://www.digikey.com/products/en?keywords=104031081	MOLEX
Microphone	digikey	423-1139-1-ND	6	$0.940	$5.64	https://www.digikey.com/product-detail/en/knowles/SPU041	Knowles
Dip switch	digikey	CT2193LPST-ND	4	$0.680	$2.72	https://www.digikey.com/product-detail/en/cts-electrocompo	CTS
Voltage regulator	Mouser	595-TPS62171QDSGRQ1	4	$1.500	$6.00	https://www.mouser.com/ProductDetail/Texas-Instruments/T	Texas Instruments
Amplifier	Mouser	584-AD8338ACPZ-R7	4	$10.830	$43.32	https://www.mouser.com/ProductDetail/Analog-Devices/AD8	Analog devices
Resonator	digikey	490-1196-1-ND	4	$0.480	$1.92	https://www.digikey.com/product-detail/en/murata-electronic	Murata Electronics No
						-	
				Total	$80.56		

8.2 Hardware

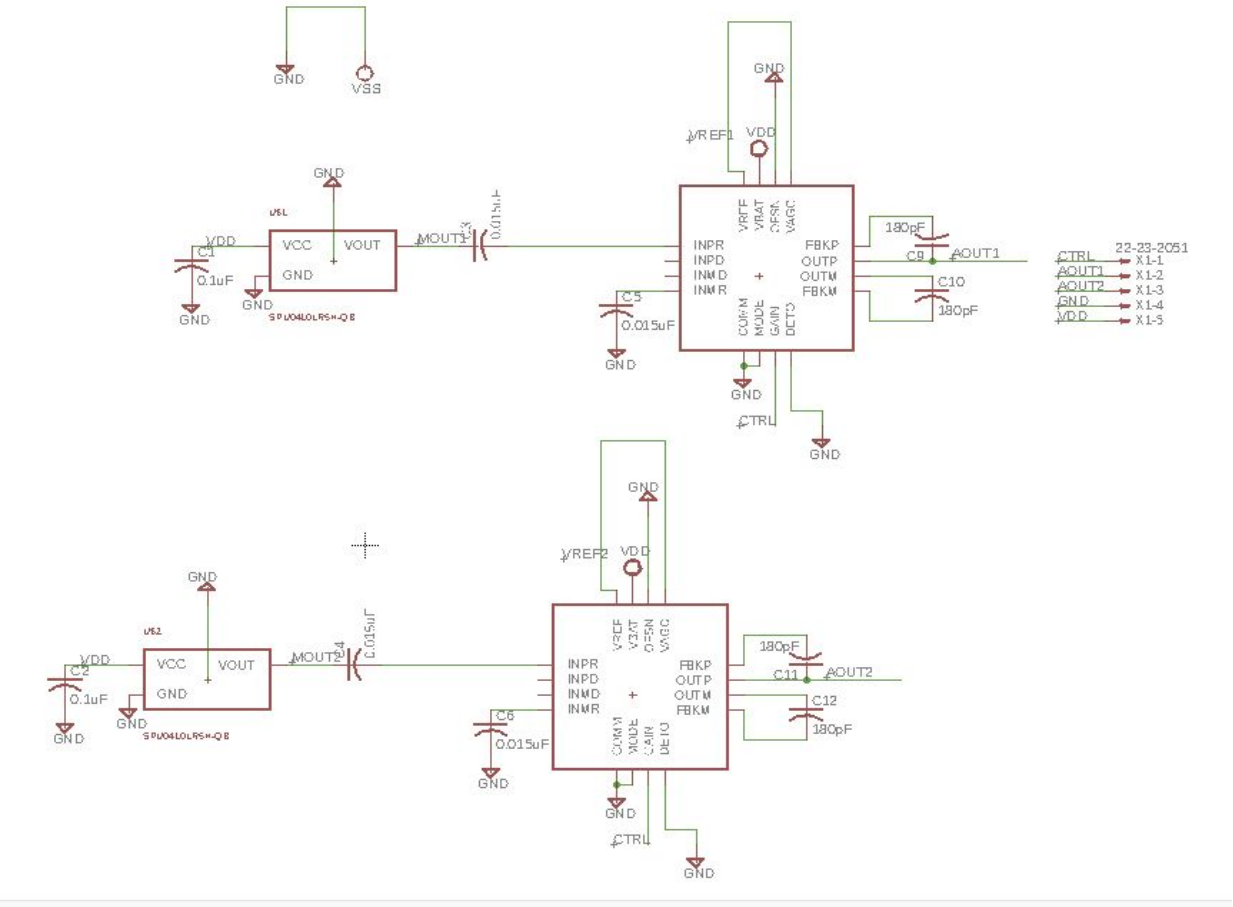

Figure 11: Neck Board schematic

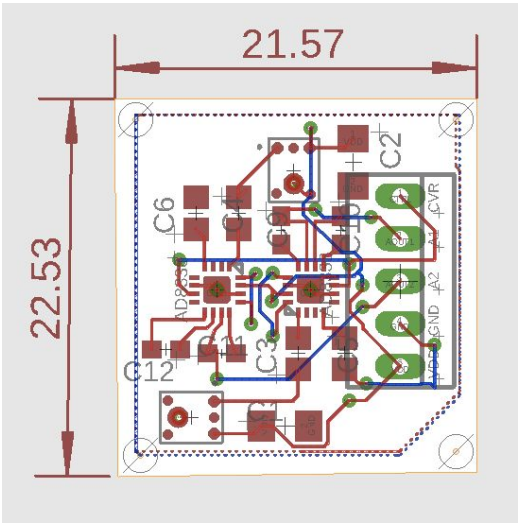

Figure 12: Neck Board Layout

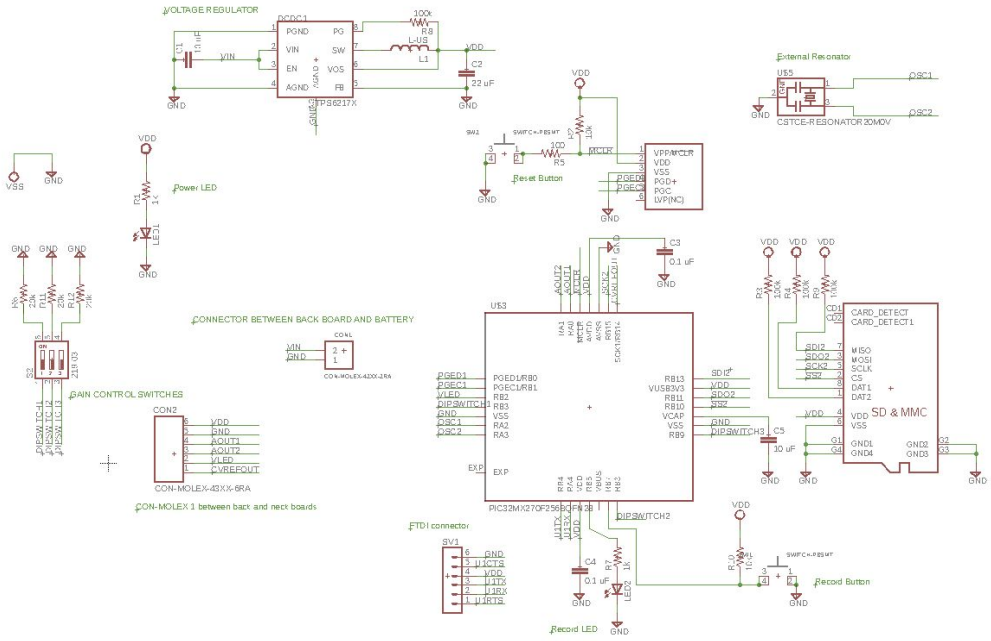

Figure 13: Back Board Schematic

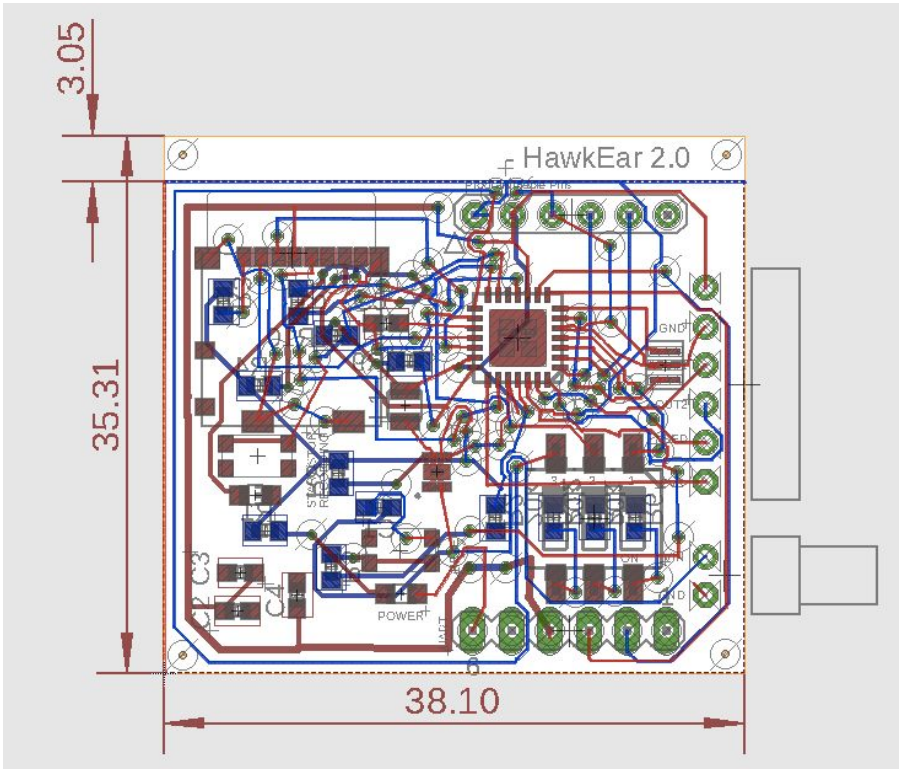

Figure 14: Back Board Layout

8.3 Software List

- **readSoundData.m** - Matlab program used to analyze the audio data collected. Takes in the files from the SD card and outputs the graphical waveform, Fourier Transform, and a .wav file of the signal recorded.
- **Hawk2.X** - MPLAB project containing the source files that program the PIC.
- **Head Board Res2.sch** - EAGLE schematic file for the head board.
- **Head Board Res2.brd** - EAGLE board file for the head board.
- **Back Board Res2.sch** - EAGLE schematic file for the back board.
- **Back Board Res2.brd** - EAGLE board file for the back board