VoxBox Robo-Drummer

Team 27: Craig Bost, Nicholas Dulin, Drake Proffitt
ECE 445 Design Document Spring 2018
TA: Zhen Qin

Table of Contents

1. Introduction

1.2. Background

2. Design

2.2. Block Design

3. Cost and Schedule

3.1. Cost

1 Introduction

1.1 Objective

Our design seeks to enable a human user to convert live “beatboxing” performance into real-time physical performance by a robot. As such, our design is best thought of as musical instrument in its own right: when “played” as intended, our design will respond by instructing a robot drummer to strike the corresponding drum kit component. Correctly “playing” our system amounts to the human user constraining their vocal performance to a subset of sounds which can be identified as comprising traditional “beatboxing” (see Background below for elaboration). This mapping of performative sounds to the correct corresponding robotic performance comprises the core challenge of our design.

Our design is distinct from existing designs which similarly process “beatboxing” into equivalent musical forms, as these existing systems typically only convert recordings of “beatbox” performance into drum samples for music production; prime examples include “The Beatbox Machine” [1] and “Vocal Beater” [2]. As such, these existing systems are purely software-based and non-real-time, whereas our design will feature real-time performance, driven by hardware-software embedded systems.

One possible social application for our system, which is not met by these other existing systems, is the capacity for persons with mobility restrictions of their extremities to be able to participate in live music performance, in a manner consistent with typical rock music performance.

1.2 Background

1.2.1 What is Beatboxing?

“Beatboxing” refers to the practice of emulating conventional percussion musical instruments with only one’s mouth, and typically incorporates a voice microphone and amplification system. Correct beatboxing would be characterized by a vocal performance that approximates or imitates conventional percussion instrument to a sufficient degree of fidelity that the performance could substitute the drum track in lieu of the actual instruments:

> *When MCs starting to rap over drum machine (beat box) beats in the urban communities of New York City, especially in the Bronx, drum machines and synthesizers were not very affordable. Samplers were well out of reach even for well-paid musicians.*
> *Necessity is the mother of invention, and without machine-supplied beats to rap over, a new, more accessible instrument was adopted - the mouth - and thus human beatboxing was born.* [3]

In this respect, our design "completes the circle," by using the imitative human performance to play the very instrument they are attempting to emulate.

1.2.2 What is "real-time"?

To have a meaningful claim as to whether or not a device performs in "real time" depends largely on the context of use. In the case of music and musical instruments, this could pertain to either the perceptions of the performer, of the audience, or both.

In the ideal world our device could reach instantaneously, but given real world constraints imposed by processing time and computational complexities required, some degree of latency cannot be avoided; this is just as true of existing technologies in the world of music, such as MIDI, whose degree of latency can vary widely [4]. This has clearly not prevented MIDI-controller based electronic instruments from being used, so the question is how much latency is too much? Can we find a metric for a seemingly subjective question?

To answer the question of how to base the standard we decided to use visual perception as our basis, given that it is the observable reaction of the robot that ultimately dictates the perception of lag (sound propagation from the drum-head will be negligibly delayed after strike). As such, we propose using the International Telecommunication Union's Recommendation ITU-R BT.1359-1 threshold of acceptability of 185 msec [5].

1.3 High-Level Requirement

1) Device must be able to receive beatbox audio input from a human user/performer and distinguish between 3 different key sounds. This implies that there are 4 valid inputs in total: clap, snare, bass kick, and no input.
2) Device must be able to perform requirement (1) in real-time (defined as above in the Background section 1.2.2 as ≤ 185 msec) and drive a machine based on said inputs.
3) The machine will have 3 drums, one for each input, and will strike the correct drum which corresponds to its input, with a measurable sound-pressure intensity above 40 dBA.

2 Design

2.1 Block Diagram

Figure 2.1 provides a graphical depiction of our complete system from a high-level perspective, detailing the functional constituent components of each sub-block and the nature of their connections.

In summary, the Power Subsystem converts AC mains power to 12V DC, which is further stepped-down to 3.3V for the DSP Subsystem. A Linear Regulator ensures power remains within spec for the DSP block, and Circuit Protection A handles the induction effects of the solenoid drives.

The Input Subsystem is comprised of a series of sub-blocks which take the analog signal generated by the Microphone (a voltage level) and applies filtering and amplification to both clean-up the signal as well as boost it, so that a better quality signal may be captured by the DSP sub-block's ADC function.

The Control Subsystem contains the DSP sub-block, whose principal tasks involve the analog-to-digital conversion of the Input subsystem's output signal, processing of the digitized input, and subsequent response to qualifying stimulus by providing the trigger signals to the Electromechanical Conversion Subsystem. These output signals pass through Circuit Protection B, which protects the DSP block from any possible back emf from the solenoid drives.

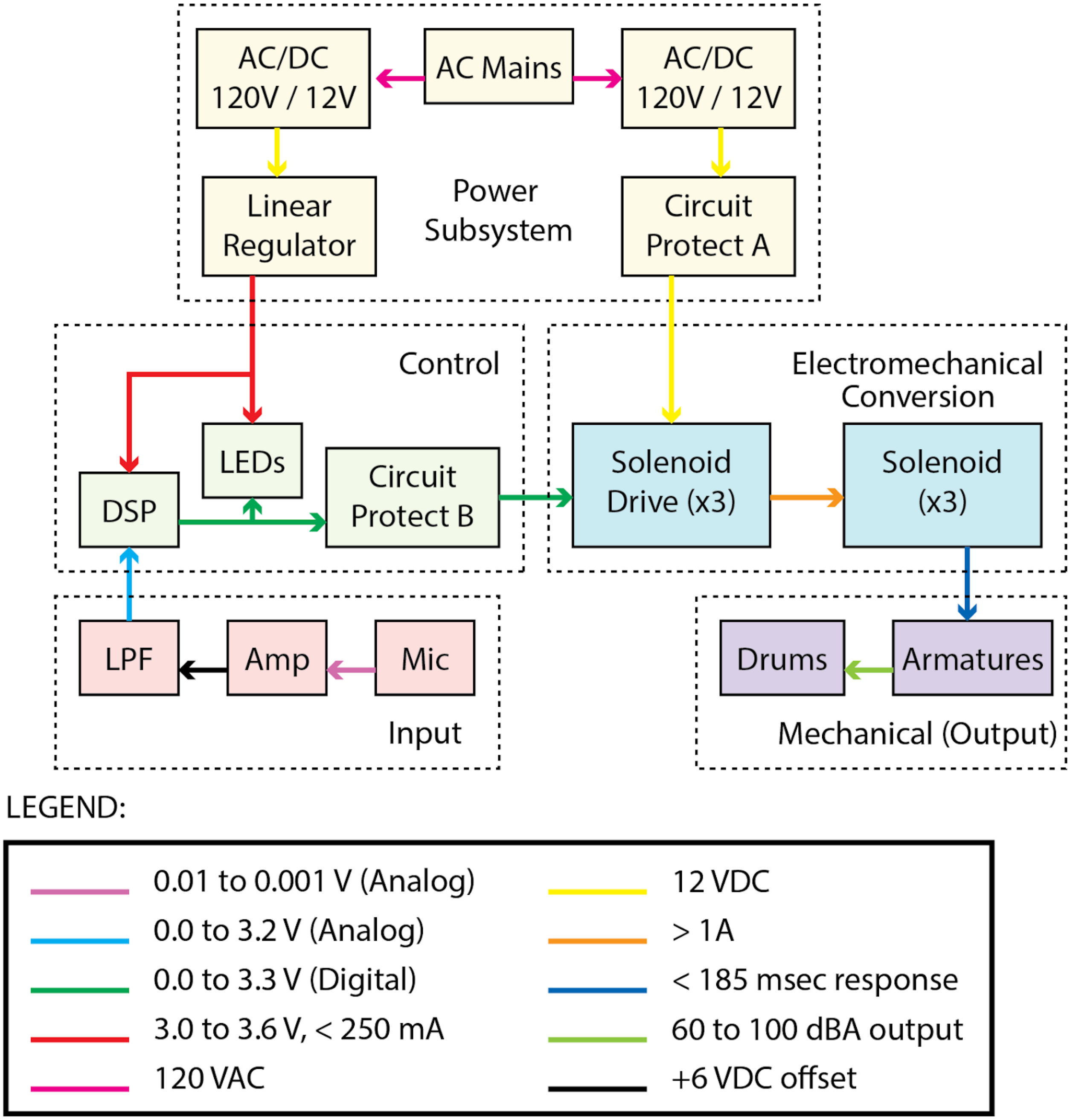

Figure 2.1: System Block Diagram

2.2 Block Design

2.2.1 Power

Functional Overview

This block contains two power converters and a AZ1117E 3.3V linear regulator. The first converter is of the low power AC/DC 120V/12V type. The second converter is of high power AC/DC 120V/12V type and is used to supply power to the solenoids. The purpose of this block is to supply power and provide protection to the circuit. The protection comes by means of a fuse.

Requirements and Verification

Requirement	Verification
Requirement 1 Low-powered power converter supplies 12V output with ±250mV, when supplied with 120VAC source..	*Equipment:* 1) 120V power mains source. 2) Breadboard 3) Oscilloscope *Procedure:* 1) Assemble circuit 2) Plug in the mains power 3) Measure output voltage on oscilloscope *Expected Result:* We should see an output voltage of 12V ±250mV..
Requirement 2 Linear regulator supplies 3.3 VDC output, within ±10%, when supplied with 12 VDC input	*Equipment:* 1) Power supply 2) Oscilloscope *Procedure:* 1) Use the power supply to supply 12VDC to the linear regulator 2) Probe the output voltage of the linear regulator 3) Check the DC value and ripple on the oscilloscope *Expected Result:* The output of the linear regulator should be 3.3V with peak-peak ripple less than ±10%.

Table 2.1: Power supply requirements and verifications.

Supporting Materials

Schematics

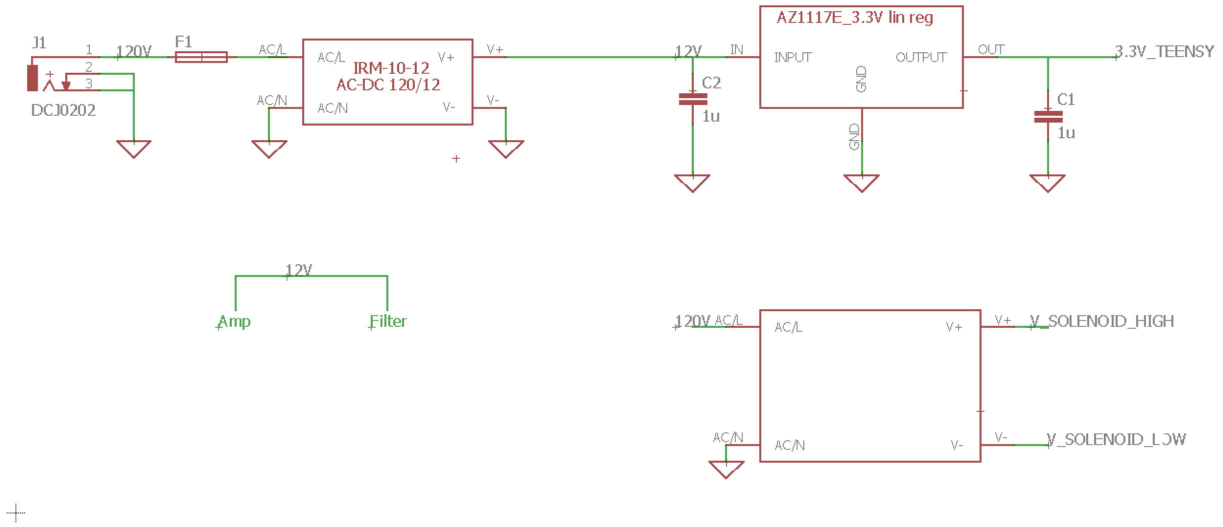

Figure 2.2: Power supply. AC/DC converters and linear regulator.

2.2.2 Microphone to Pre-Amp

Functional Overview

The microphone pre-amp is a 2-stage op-amp circuit which takes mic-level input from the Behringer Ultravoice XM8500 microphone (or comparable device), and amplifies the signal to line-level. The purpose of this amp is to raise the voltage level before it goes into the filter so that the filter will not significantly raise the SNR, and also to raise the signal amplitude before it goes into the ADC. The pre-amp circuit is comprised of LM741 op-amps and a resistive and capacitive network, as detailed in Figure 2.3.

Requirements and Verification

Requirement	**Verification**
Requirement 1 Amplifier has a voltage gain of 63dBV ±1.02% from 1mV input while loaded with filter. ±1.02% was chosen to prevent ADC saturation.	*Equipment:* 1) Breadboard 2) Power supply 3) Oscilloscope 4) Signal generator *Procedure:* 1) Setup circuit with filter load 2) Power circuit

	3) Apply Signal generator sinusoidal input of 1mV amplitude to the amp 4) Observe FFT 5) Record the gain at input frequency *Expected Result:* If we apply a 1kHz sinusoidal input, then we should see a gain of 63dBV ±5% at 1kHz.
<u>Requirement 2</u> Amplifier meets requirement 1 from 200Hz to 10kHz.	Repeat requirement 1, but sweep through frequencies at intervals of 1kHz.

Table 2.2: Amplifier requirements and verifications.

<u>Supporting Material</u>

Schematics

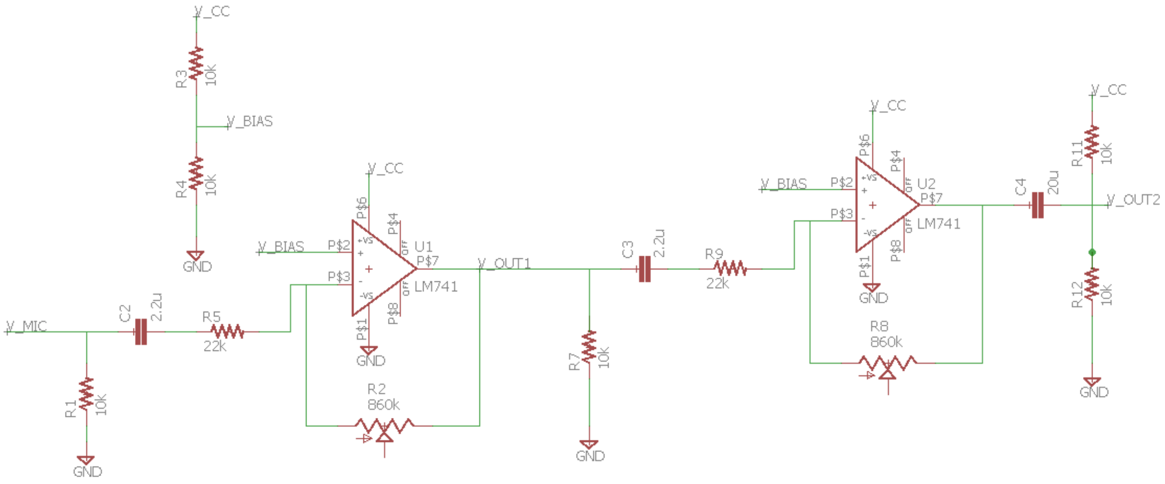

Figure 2.3: Microphone Pre-Amp Schematic

Simulations

Figure 2.4 below was simulated in LTspice with ideal resistors, capacitors, and sources. We used TI's LM741 op-amp Spice model in the simulation. The plot shows the voltage gain and group delay at the output. The units for gain are in dBV where the reference voltage is 1V.

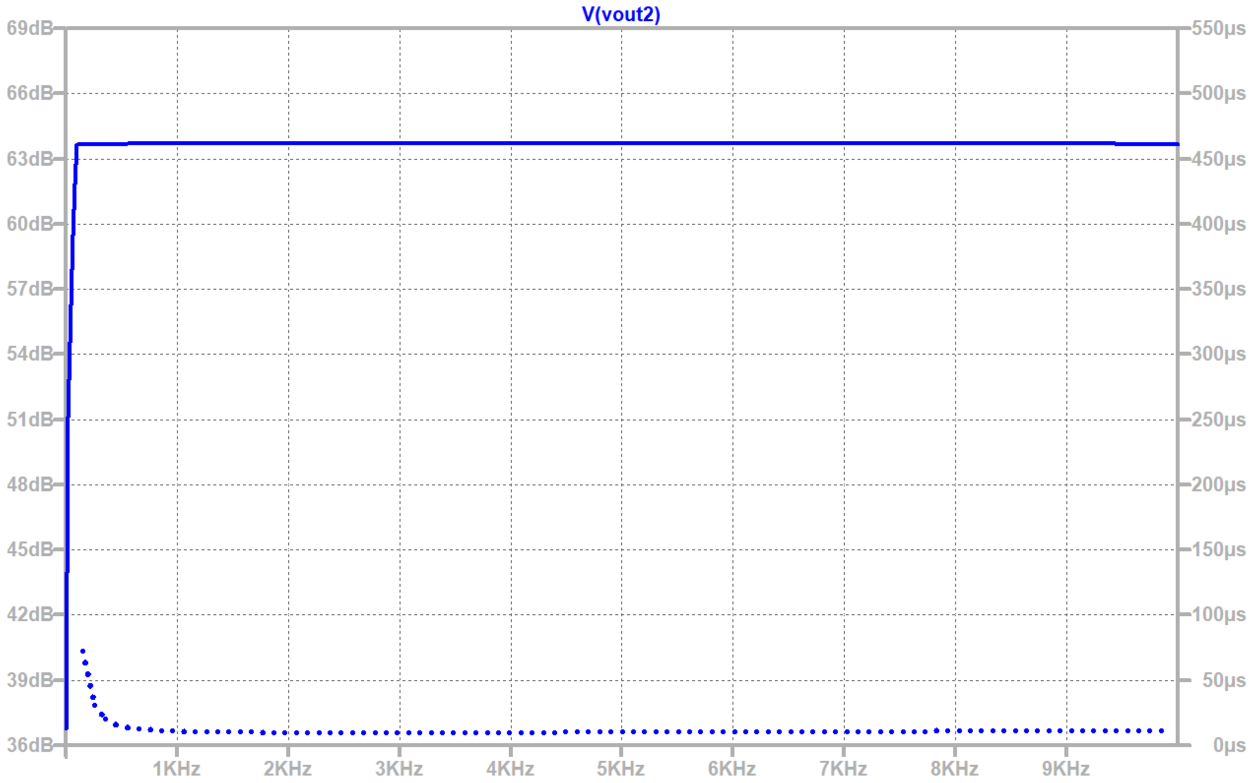

Figure 2.4: Voltage gain and group delay of amplifier from 1 to 10kHz.

Calculations

Both stages of the amplifier are identical, where each stage is configured as an inverting op-amp. The inverting op-amp has a voltage gain of:

(2.1) $$\frac{V_{out}}{V_{in}} = -\frac{Z_{feedback}}{Z_{in}} = \frac{860k}{23k} = 37.4\ V/V$$

$Z_{feedback}$ = 860kΩ, and Z_{in} = 23kΩ. If we assume the capacitors couple AC signals well, then each stage has a voltage gain of 37.4, which corresponds to a dBV gain of:

(2.2) $$gain[dBV] = 20log(\frac{V_{out}}{1}) = 20log(37.4) = 31.5dBV$$

Then the total gain of the 2-stage amplifier is:

(2.3) $$gain_{2-stage} = gain_{1,dBV} + gain_{2,dBV} = 31.5 + 31.5 = 63dBV$$

The ADC input requires a max signal amplitude of 1.6V, we have designed our amplifier to output a max signal amplitude of 1.5V from a 1mV input. We measured our microphone output signal amplitude to be 1mV from human beatbox input. This difference in amplitude requires a voltage gain of:

(2.4) $$gain_{desired} = 20log(\frac{1.5}{.001}) = 63.5dBV$$

2.2.3 Filter

Functional Overview

We have designed a 5-stage low-pass Butterworth filter using Texas Instrument's Filterpro design software [6]. It receives analog input from the audio pre-amp, and outputs the filtered signal to the DSP block for analog to digital conversion. It is a tenth order Sallen-Key design with non-inverting gain.

In designing the filter a couple of features were desired. An essential design requirement was the post-sampling preservation of the 0-10 kHz range, as preliminary testing has shown all our desired sounds will occupy this frequency range. To accomplish this task we required a filter with unity gain and linear phase throughout the passband.

The butterworth filter is a common choice for maximally flat gain in the passband, though it does present nonlinear phase near the edge of the passband. Noting that our analog to digital conversion will be occuring at a sampling frequency of 44.1 kHz, we decide to move our cutoff frequency to 23kHz (the -3dB point of the filter). This aided in dragging out our linear phase through most of the 0 to 10kHz frequency range, though some 'slight' nonlinearities still exist. Erring on the side of caution, we defined 33 kHz to be our band limit frequency. It corresponds to a gain of -30dB. With this definition, sampling at a rate of 44.1kHz result in no aliasing in the critical 0-10 kHz range. It should be noted that if we define our max sampling frequency to be fs$\approx 44\ kHz$, a bandwidth of up to 34 kHz is permissible, any greater and aliasing will occur in our 0 to 10 kHz range.

Requirements and Verification

Requirement	**Verification**
Requirement 1 The filter passes all frequencies in the 0 to 10,000Hz range with minimal amplitude distortion. Passband Deviation from unity gain< $\pm$%10	*Equipment:* 1) Breadboard 2) Function generator 3) Oscilloscope 4) Power Supply *Procedure:* 1. Assemble low-pass filter circuit on breadboard. 2. Using a splitter connect the function generator to both the input of the circuit and input channel 1 one of the oscilloscope. Then connect the output of the filter circuit to to oscilloscope. 3. Apply power to the circuit, and use the function generator to generate a sinusoidal signal. Apply a frequency sweep to the filter by adjusting the frequency of of the sinusoid and recoding the

	result of the output amplitude. Do this for all 100 Hz increments from 0 to 10,000Hz. *Expected Result:* The filter should pass the sinusoidal input to the output relatively unaltered in magnitude (original input amplitude $\pm \%10$)
<u>Requirement 2</u> Near linear phase shift of the output signal.	*Equipment:* 1) Oscilloscope 2) Breadboard 3) Power Supply 4) function generator *Procedure:* 1. Assemble low-pass filter circuit on breadboard. 2. Using a splitter connect the function generator to both the input of the circuit and input channel 1 one of the oscilloscope. Then connect the output of the filter circuit to to oscilloscope, channel 2. 3. Apply power to the circuit, and use the function generator to generate a sinusoidal signal. Apply a frequency sweep to the filter by adjusting the frequency of of the sinusoid and recoding the unwrapped phase shift, as well as the input sinusoid's frequency. Do this for all 100 Hz increments from 0 to 10,000 Hz 4. Plot the phase shift in radians vs the frequency in Hz and perform a linear fit. Ensure the Coefficient of Determination, R^2 is greater than .8. *Expected Result:* A linear plot implies a constant time shift for all frequencies in the 0 to 10kHz range, and thus no distortion of the signal. This is a result of the time delay property[7]: $F[x(t-t_o)] = X(j\omega)e^{j\omega t_o}$ The plot should match very closely to the LtSpice simulation results of figure 2.9 [8]. Exact degrees of delay are unimportant, as long as the resulting plot is linear.

Table 2.3: Requirements & Verification

Supporting Materials:

Schematics and Block Diagrams:

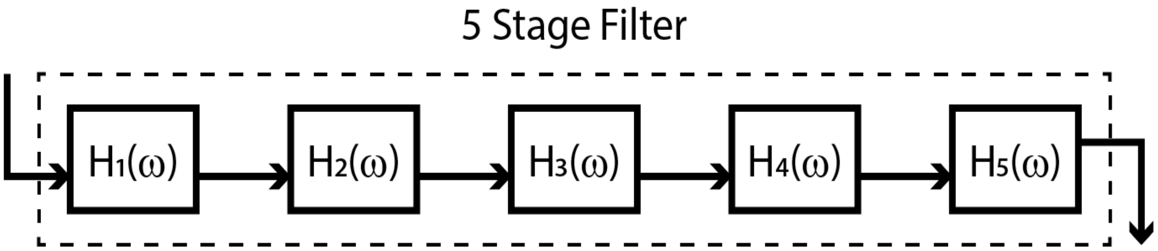

Figure 2.5: Filter Sub-Block Internal Block Diagram

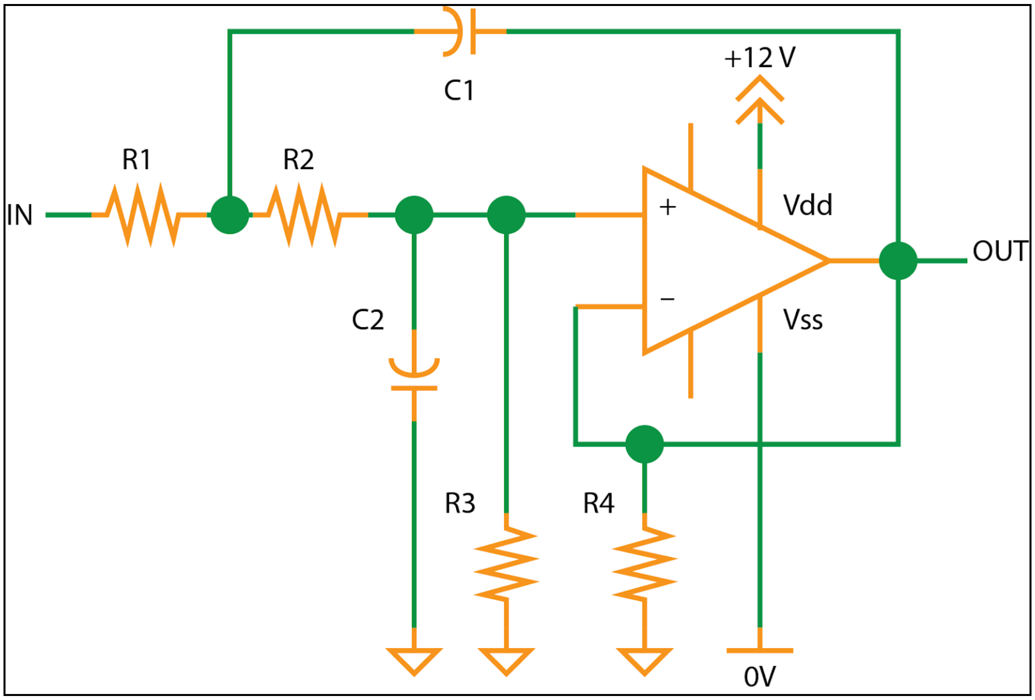

Figure 2.6: Filter Stage Schematic

Stage	R1	R2	R3	R4	C1	C2
1	4.3 kΩ	9.1 kΩ	330 kΩ	330 kΩ	1.2 nF	1nF
2	3.6 kΩ	8.2 kΩ	330k	330k	1.5 nF	1nF
3	3.3 kΩ	6.2 kΩ	330k	330k	2.2 nF	1nF
4	2 kΩ	4.3 kΩ	330k	330k	5.6 nF	1nF
5	680 Ω	1.5 kΩ	330k	330k	47 nF	1nF

Table 2.4: Filter Stage Schematic Legend

Performance Projection Plots:

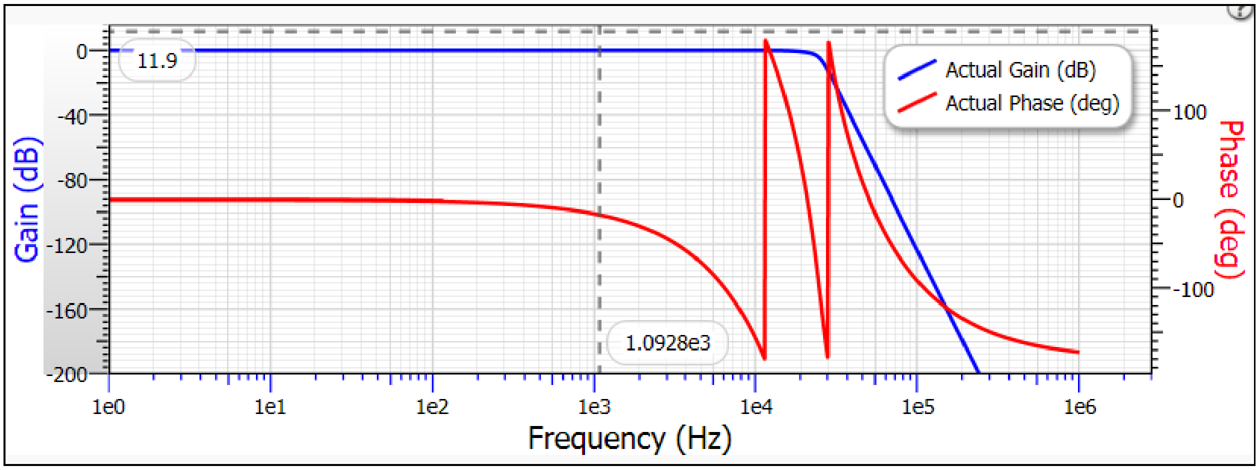

Figure 2.7: Filter Magnitude and Phase Response (via FilterPro software)

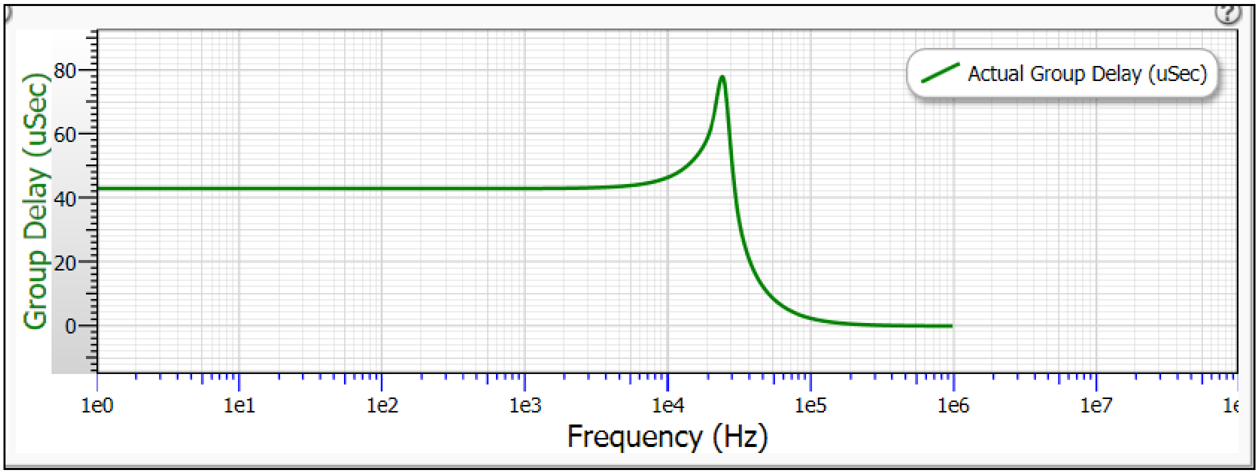

Figure 2.8: Group Delay (via FitlerPro software)

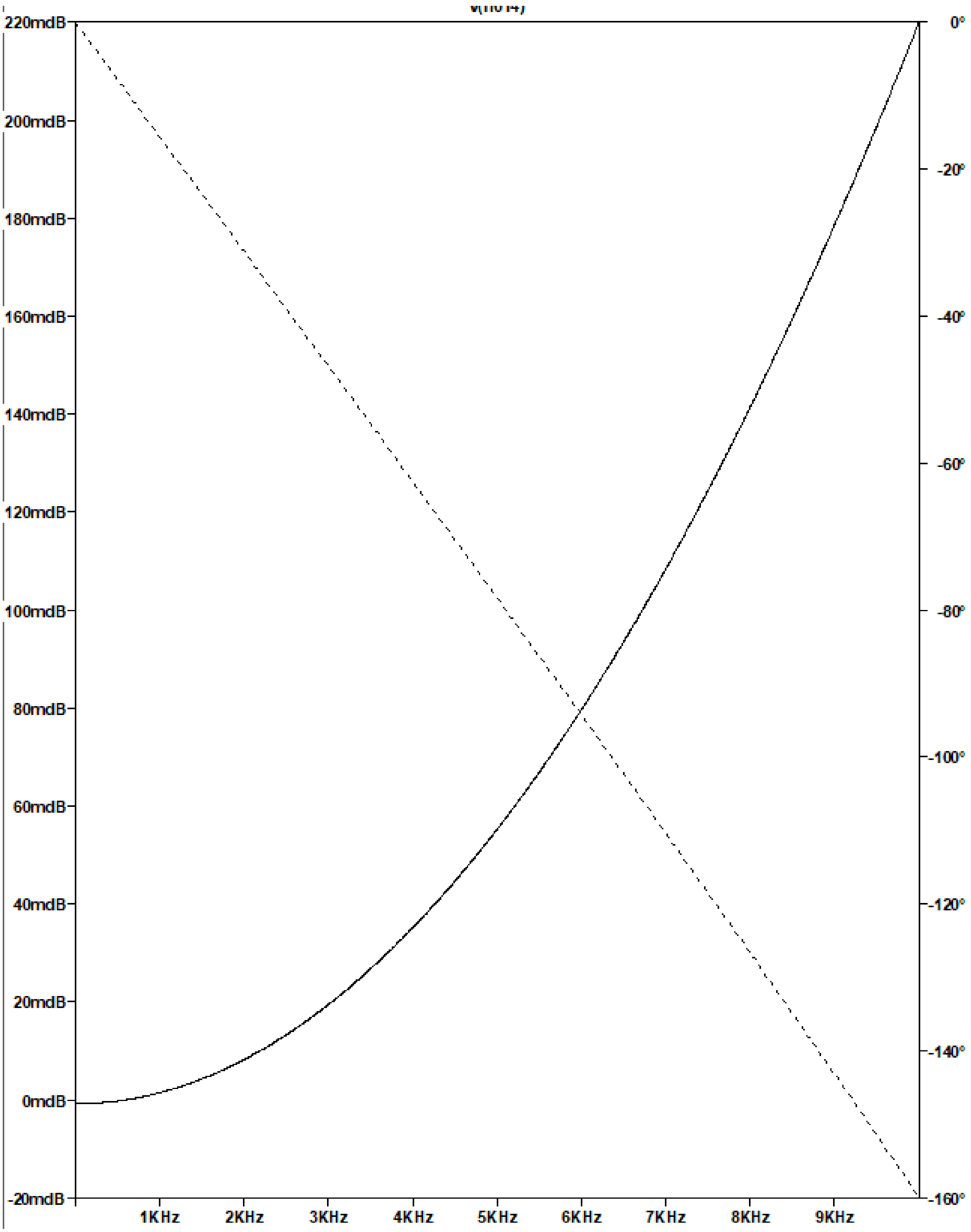

Figure 2.9: Filter Magnitude (solid) and Phase Response (dashed) 0 to 10kHz (via LTspice software)

Calculations:

Permissible bandwidth postfiltering:

The signal needs to be filtered in such a way as to avoid aliasing when sampled.

(2.5) $$fs = 44kHz$$

Max resolvable frequency after sampling,

(2.6) $$freq_{max} = fs/2 = 22kHz$$

In *discrete frequency*, spectral copies occur every 2π, where π corresponds to the original $22kHz$ *analog frequency* of the spectral copy centered at the origin.
The spectral copies can encroach up to the 10kHz point of the origin centered spectral copy.

(2.7) $$22kHz - 10kHz = 12kHz$$

Thus our signal must bandlimited to

(2.8) $$22kHz + 10kHz = 34kHz$$

2.2.4 DSP

Functional Overview

The DSP block will be comprised of a Teensy 3.6 board, which will sample the amplified and filtered microphone analog voltage signal via its ADC pins. The software routines outlined below will utilize this data for the purpose of 1) applying additional digital filtering techniques to further refine the digitized time-domain signal; 2) using time-domain and frequency-domain analysis to characterize the input; 3) apply event detection algorithms to map the input to an output trigger stimulus.

We chose the Teensy 3.6 because: 1) it can be programmed in C/C++, unlike conventional DSP chips which often require programming in proprietary assembly languages, which would require an investment of time to learn; 2) the Teensy can be programmed using a straightforward IDE; 3) the Teensy supports FFT C libraries; and lastly 4) the Teensy can be mounted to the main system PCB via header-pins allowing both quick replacement in case of accidental blow-out, and software development in parallel, separately from other aspects of the design development and fabrication.

Requirements & Verification

Requirement	**Verification**
Requirement 1 DSP (Teensy 3.6) will maintain function within range ±10% of 3.3 VDC (3.0 VDC to 3.6 VDC)	*Equipment:* 1. DC Power Supply (x1) 2. Digital Multimeter (x2) 3. Arduino IDE 4. Test Program “test_vin_performance.ino” *Procedure:* 1. Load “test_vin_performance.ino” into Teensy 3.6 via Arduino IDE. 2. Disconnect Teensy board from Computer (i.e. disconnect VUSB) 3. Power Teensy board with 3.0 VDC 4. Probe Vin and output pin 14 5. Sweep DC source voltage from 3.0 to 3.6 VDC. *Expected Result:* The output pin will continue to hold digital high level signal at Vin, within same tolerance range.
Requirement 2 DSP (Teensy 3.6) will register from 0.0 V to 3.0 V (±10%) and articulate these values to 13-bit precision.	*Equipment:* 1. Power Supply (x1) 2. Arduino IDE 3. Test program “test_adc_performance.ino” *Procedure:* 1. Load “test_freq_performance.ino” into Teensy 3.6 via Arduino IDE. 2. Run program with Serial Monitor open. 3. Set Power Supply to 0.0 VDC and connect to anode to pin 16 (a.k.a. A2) and cathode to analog ground. 4. Sweep the voltage level to 3.3 VDC. *Expected Result:* Serial monitor read-out from test program will display the 13-bit quantized digital value on a 0 to 8191 scale. As the program uses the default reference voltage level of 3.3 V, at 3.3 V the value read should be within -10% of 8191. At 1.5 V, the value should be within ±10% of 3723.
Requirement 3 For low frequencies:	*Equipment:* 1. Function Generator (x1) 2. Arduino IDE

DSP (Teensy 3.6) will register frequency distinctions of one octave of 1 kHz (half the frequency below at 500 Hz, twice the frequency above at 2 kHz), within a range of ±10% of the boundary frequencies.	3. Test programs “test_low_freq.ino” *Procedure (Low Frequency):* 1. Load “test_low_freq.ino” into Teensy 3.6 via Arduino IDE. 2. Run program with Serial Monitor open. 3. Set Function Generator to 1.2 Vpp, +0.6V offset, sine wave, and connect to pin 16 (a.k.a. A2) on the board. 4. Set frequency to 1000 Hz, and observe the Serial Monitor read-out. 5. Set the frequency to 500 Hz and pan the frequency ±50 Hz, observing the read-out. 6. Increase the frequency to 2000Hz and pan the frequency ± 200 Hz, observing the read-out. *Expected Result:* Bin 1 (second from left) should show a peak value 10% larger than its neighbors, and no peaks anywhere else. The lower octave frequency should peak in Bin 0, and the peak should not stray into Bin 1 as the frequency is panned. The higher octave frequency should peak in Bin 2, and should not stray into Bin 1 as the frequency is panned.
<u>Requirement 4:</u> For high frequencies: DSP (Teensy 3.6) will register frequency distinctions of 40 Hz ±10% in the frequency band approaching the Nyquist rate, taking 20 kHz as a conservative cut-off frequency.	*Equipment:* 1. Function Generator (x1) 2. Arduino IDE 3. Test programs “test_high_freq.ino” *Procedure (High Frequency):* 1. Load “test_high_freq.ino” into Teensy 3.6 via Arduino IDE. 2. Run program with Serial Monitor open. 3. Set Function Generator to 1.2 Vpp, +0.6V offset, sine wave, and connect to pin 16 (a.k.a. A2) on the board. 4. Set the Frequency to 20 kHz, and observe the read-out. 5. Pan the frequency ±10 Hz, observing the read-out. 6. Increase the frequency to 20040 Hz, and pan ±4 Hz observing the read-out. 7. Decrease the frequency to 19960 Hz, and pan ±4 Hz observing the read-out. *Expected Result:*

	Bin 2 should output a peak value 10% larger than its neighbors, and no peaks anywhere else. Increasing the frequency should move the peak right one bin from Bin 2, and the peak should not stray into Bin 2 as the frequency is panned. Decreasing the frequency should move the peak left one bin from Bin 2, and the peak should not stray into Bin 2 as the frequency is panned.

Table 2.5: DSP Requirements and Verification

Supporting Materials:

Algorithm:

We recorded the target sounds and generated a spectrogram to facilitate preliminary analysis of the data sets we'd be working with. It revealed that each sound is spectrally discernible in the 0 to 10kHz frequency range and distinct from one another, as shown below in Figure 2.1.

Based these preliminary findings, our algorithm will use the following features for determining whether a valid input has been issued on not (i.e. "event detection"):

From the analog input pins, the ADC functionality of the Teensy 3.6 will allow us the data needed to compute a 1024-point FFT. We will truncate this to the lower 256 bins (representing 0 to 11 kHz), from which we derive their magnitude, and in turn calculate the total energy for that frequency band. At this juncture, we will check for a voiced-threshold (to be determined experimentally); if it is voiced the algorithm continues; if it is not, it loops back to acquire updated input data.

In either case, the algorithm checks for an event flag; a variable which tracks the state of the system, whether an event is occurring or not. If the algorithm does not detect a voiced threshold and the event has been asserted, it will deassert it; a transition which marks the end of a valid sound event. If the algorithm does detect a voiced threshold, it checks the state of the event flag: if it is asserted, it loops back to start, representing acknowledgment that the valid event is still underway; if it is not asserted the algorithm progresses to the characterization and comparison phase of its operation.

Upon a new voiced event, the frequency bin corresponding the average energy is identified and added to a circular event buffer. From this buffer a running average of their entries is compared to a series of reference values, one for each valid sound. If this average is within a radix of 5 bins (approximately 215 Hz) then the corresponding digital trigger pulse is issued to the solenoid driver. In either case, the event flag is asserted.

Our preliminary algorithm for event detection is shown in flowchart format below in Figure 2.3.4b for visual reference.

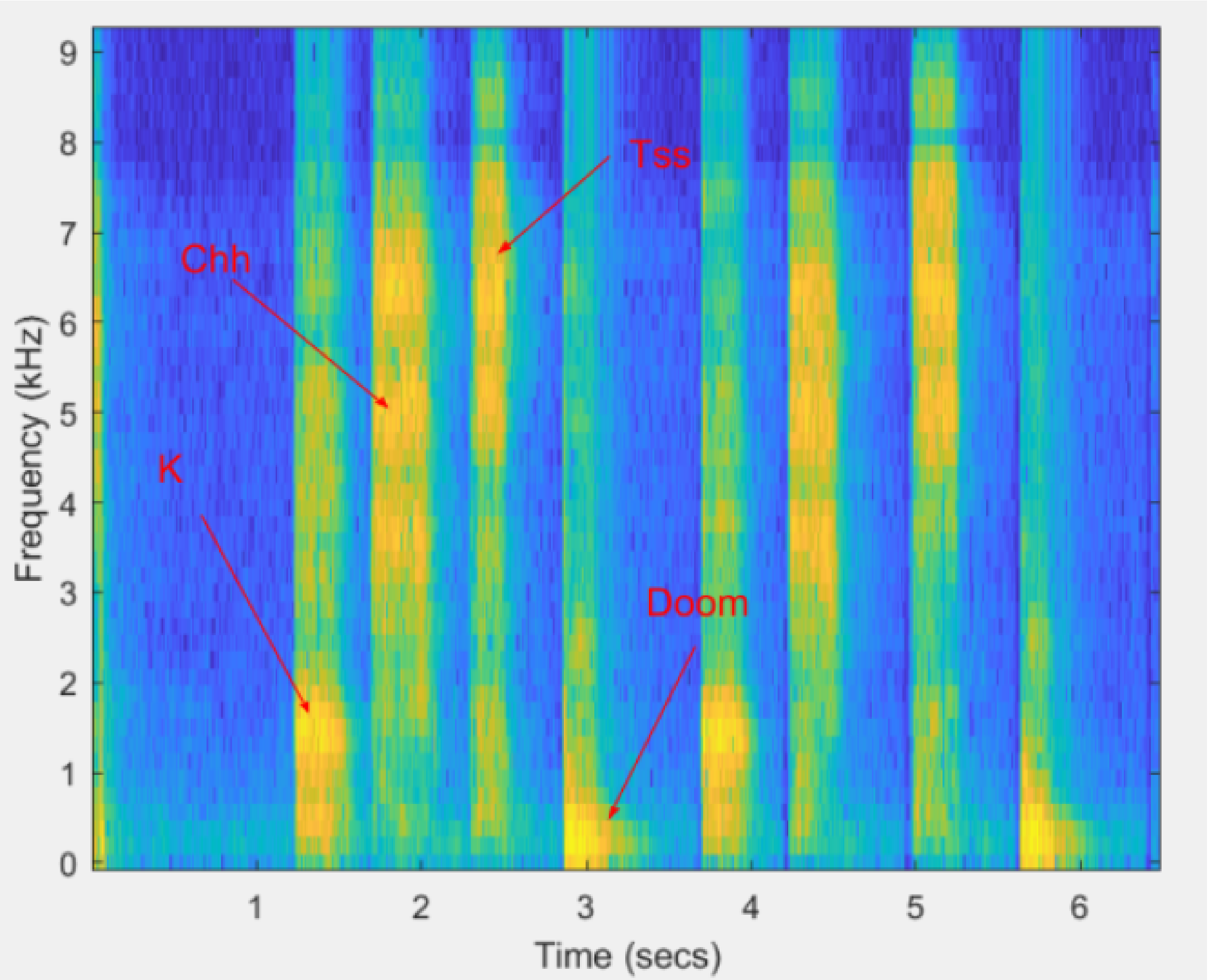

Figure 2.10: Spectrogram of Sample Beatbox Sounds.

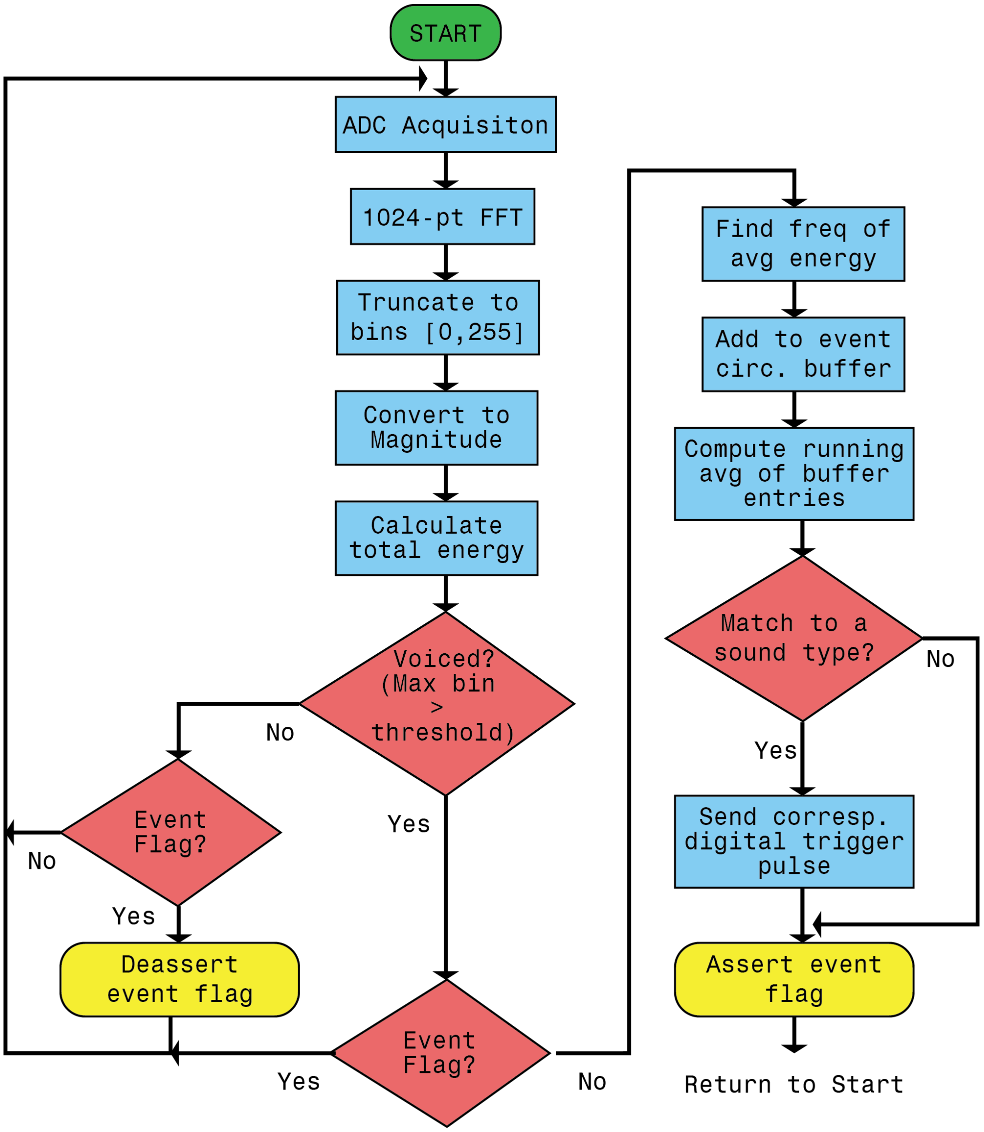

Figure 2.11: Algorithm Flowchart

Requirement and Verification Test Code:

The test code given below is based off of example code provided by the manufacturer of the Teensy, as no explicit testbench code was available for our specific needs.

“test_vin_performance.ino” is modeled after the Requirement 3 drafted by Team 1 of the Fall 2017 ECE 445 Class for their Microcontroller Requirement & Verification. [9]

“test_adc_performance.ino” is based off of the “Tutorial 4: Analog Input” tutorial on PJRC.com [10].

“test_low_freq.ino” and “test_high_freq.ino” are based off of the example code that PJRC.com’s proprietor, Paul Stoffregen, provided for the Teensy’s basic FFT functionality. [11]

```
/*
 * TEST VIN PERFORMANCE
 * for R & V
 */

int output_high = 14;
int output_low = 15;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(output_high, OUTPUT);
  pinMode(output_low, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(output_high, HIGH);
  digitalWrite(output_low, LOW);

} // END PROGRAM
```

```
/*
 * TEST ADC PERFORMANCE
 * for R & V
 */

void setup() {

  analogReadResolution(13);

}

int val;

void loop() {

  val = analogRead(A2);
  Serial.print("analog ADC_A2 is: ");
  Serial.println(val);
  delay(250);

} // END PROGRAM
```

```
/*
 * TEST LOW FREQ
 * for R & V
 */

#include <Audio.h>
```

```
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <SerialFlash.h>

AudioInputAnalog         adc1;
AudioAnalyzeFFT256       myFFT;
AudioConnection          patchCord1(adc1, myFFT);

// FFT Bins (Lower fourth of FFT)
float freqBands [16];
float freqBins_463_467 [5];

void setup() {

  // Allocate memory for Audio:
  // Input param corresponds to sample size,
  // so for 13-bit ADC, array size must be [0, 12]
  AudioMemory(12);

}

void loop() {

  // Get FFT output to capture bins:
  if(myFFT.available()){

    freqBands[0] = fft256_1.read(0, 3);
    freqBands[1] = fft256_1.read(4, 7);
    freqBands[2] = fft256_1.read(8, 11);
    freqBands[3] = fft256_1.read(12, 15);
    freqBands[4] = fft256_1.read(16, 19);
    freqBands[5] = fft256_1.read(20, 23);
    freqBands[6] = fft256_1.read(24, 27);
    freqBands[7] = fft256_1.read(28, 31);
    freqBands[8] = fft256_1.read(32, 35);
    freqBands[9] = fft256_1.read(36, 39);
    freqBands[10] = fft256_1.read(40, 43);
    freqBands[11] = fft256_1.read(44, 47);
    freqBands[12] = fft256_1.read(48, 51);
    freqBands[13] = fft256_1.read(52, 55);
    freqBands[14] = fft256_1.read(56, 59);
    freqBands[15] = fft256_1.read(60, 63);

  }

  // Print output to serial monitor:
  for(int i = 0; i < 5; i++){

    Serial.print(freqBins_0_4[i]);

    if(i == 4){
      Serial.println(" ");
    }
    else{
      Serial.print(" | ");
    }
  }
} // END PROGRAM
```

```
/*
 * TEST HIGH FREQ
 * for R & V
 */

#include <Audio.h>
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <SerialFlash.h>

AudioInputAnalog      adc1;
AudioAnalyzeFFT1024   myFFT;
AudioConnection       patchCord1(adc1, 0, myFFT, 0);

void setup() {

  AudioMemory(12);

  myFFT.windowFunction(AudioWindowHanning1024);

}

void loop() {

  float n;
  int i;

  if (myFFT.available()){
for(i = 463; i < 468; i++){

      n = myFFT.read(i);

      Serial.print(n);
      Serial.print(" ");

    }
    Serial.println();
  }

}
```

2.2.5 Solenoid Driver

Functional Overview

This block contains the solenoid drive circuit. The drive circuit takes input from the Teensy as a 0V or 3.3V digital signal, then this digital signal is amplified and used to switch the FET that controls the current to solenoid. Two forms of circuit protection are provided, a flyback diode on the solenoid to prevent high voltage spikes, and gate input resistor to prevent large current draws from the Teensy.

Requirements and Verification

Requirement	Verification
Requirement 1 Solenoid driver supplies 10V ± 1V to the gate of the power FET.	*Equipment:* 1) Power supply 2) Signal generator 3) Oscilloscope *Procedure:* 1) Connect signal generator to port V_TEENSY, and oscilloscope connected to FET gate. 2) Set signal generator to square wave, 3.3V, 10Hz. 3) Power on circuit and signal generator. 4) Observe if the requirement is met across the oscilloscope. *Expected Result:* The oscilloscope should output a square wave with 10V amplitude at 10Hz frequency.
Requirement 2 Solenoid driver supplies 6A through the solenoid for a duration of 0.1 seconds at a PWM freq of 1Hz.	*Equipment:* 1) Power supply 2) Signal generator 3) Oscilloscope *Procedure:* 1) Connect signal generator to port V_TEENSY and a current probe on the FET drain_source path. 2) Set the power supply to 12V volts and power up the circuit. 3) Set the signal generator to square wave, 3.3V, 1Hz. Turn it on. 4) Observe the solenoid current on the oscilloscope. Confirm that the expected current

	is being applied. *Expected Result:* We should see the solenoid current reach 6A before it decays.

Table 2.6: Solenoid Driver Requirements and Verification

Supporting Materials

Schematics

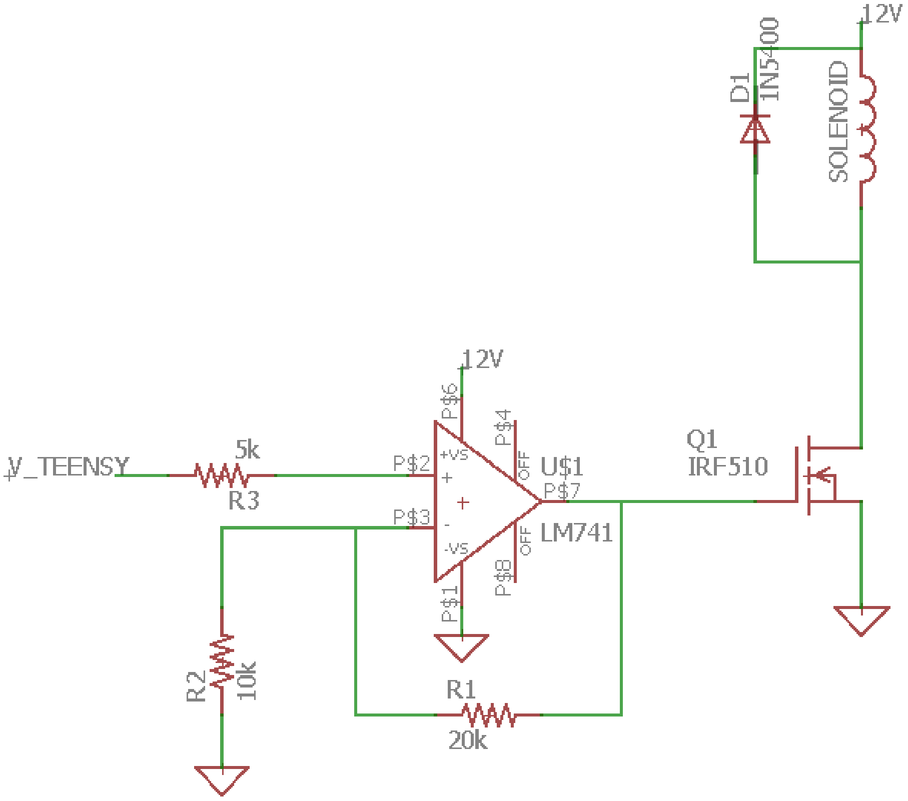

Figure 2.12: Solenoid Driver Circuit

Calculations

The amplifier used to boost the input signal is an op-amp in non-inverting configuration. The gain for a non-inverting op-amp is:

(2.9)
$$\frac{V_{out}}{V_{in}} = \left(1 + \frac{Z_{feedback}}{Z_{inverting}}\right) = 1 + \frac{20k}{10k} = 3\ V/V$$

Then an output voltage of 3.3V from the Teensy will apply 10V at the gate of the FET.

2.2.6 Solenoid

Functional Overview

The solenoid is the electromechanical energy conversion device of our system. Its purpose is to drive the drummer so that audible output can be heard. The device of choice is a linear actuator pull type. This actuator exerts a pulling force when current is applied, and exerts a pushing force when current is cut-off. These two forces will allow the actuator to pull the drummer arm in the direction of a drum strike, and push it back to reset it.

Requirements and Verification

Requirement	**Verification**
Requirement 1 Solenoid plunger achieves a full 1 inch stroke with 12V input.	*Equipment:* 1) Power supply 2) Ruler *Procedure:* 1) Assemble the solenoid and driver circuit with the input to the driver connected to the power supply outputting 3.3V. 2) Power up the the driver circuit but keep the driver input low. 3) Now switch the driver input on and then immediately switch it off. 4) Observe that the solenoid achieves a 1 inch stroke. *Expected Result:* The solenoid stroke should pull in when current is applied, then push back out when current is cut-off. Each action should displace 1 inch.

Table 2.7: Solenoid Requirements and Verification

Supporting Materials

Calculations

The solenoid design is shown in Figure 2.13. Our design goal is for the solenoid to have a 1.5 inch stroke in under 0.3 seconds while loaded with the drummer arm. To achieve these constraints, we must calculate the solenoid turn count and current draw. We will start by calculating the force output required from the actuator.

The solenoid will exert a downward force on both the actuator plunger and the drummer arm. Since the drummer arm swings through a small angle, we can approximate that the actuator resides below the center of mass of the drummer arm (see Figure 2.14) and plunger. Given the densities of each [12], [13], the mass of the drummer arm and solenoid plunger are:

(2.10) $mass_{arm} = (density_{PLA})(volume_{arm}) = (\frac{1250kg}{m^3})(1.055x10^{-4}m^3) = 0.0126kg$

(2.11) $mass_{plunger} = (density_{steel})(volume_{plunger}) = (\frac{8050kg}{m^3})(2.1545x10^{-6}m^3) = .0173kg$

Next we will calculate the starting force needed to pull down the plunger and drummer arm in 0.2s:

(2.12) $x_f = x_0 + v_0t + 0.5at^2 \Rightarrow a = (1in)(2)/(0.1s)^2 = 5.08m/s^2$

(2.13) $F = ma = (mass_{arm} + mass_{plung})(5.08m/s^2 + 9.81m/s^2) = 0.4454N$

The net force in the above equation accounts for the force of the spring pushing against the mass and force needed to move the mass 1 inch in 0.1s.

Next we will calculate the number of turns required to achieve the starting force for 2A of current:

(2.14) $N = \sqrt{2Fx^2/i^2u_0A} = \sqrt{2(0.4454N)(.0254m)^2/(6A)^2(4\pi x10^{-7})(2.837x10^{-5})}$

$$N = 670\ turns$$

Where F is the solenoid starting force, x is the solenoid air gap distance, i is the current, u_0 is the permeability of free space, and A is the area of the air gap.

Mechanical Diagram

a) Plunger: made from stainless steel 430 material, cylindrical shape, length = 76.2mm, radius = 3mm.
b) Pull-piece: made from stainless steel 430 material, cylindrical shape, length = 25.4mm, radius = 3mm.
c) Guide tube: made from PLA, cylindrical with flanges, inner radius = 3.1mm, outer radius = 22mm, length = 88.9mm.

d) Coil: made from 20 AWG magnet wire, number of turns = 670.
e) Yoke: made from stainless steel 430 material, rectangular box with two exposed sides, dimensions are 98.9mm x 24mm x 32mm, wall thickness = 4mm.

Drawings

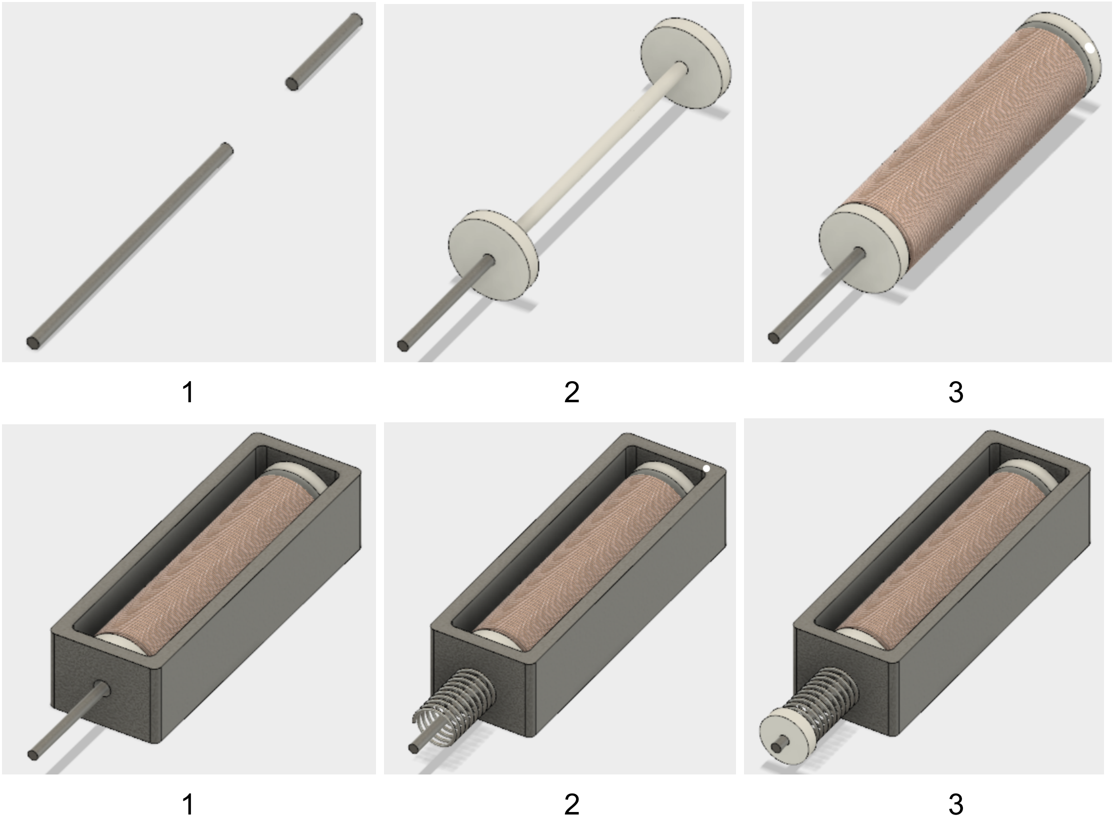

Figure 2.13: Solenoid assembly: 1) Plunger and pull-piece, 2) Guide tube, 3) Coil, 4) Yoke, 5) Spring, 6) Spring cap.

2.2.7 Drummer

Functional Overview

The drummer is the final stage of the system which is comprised of three sets of drums and drumsticks where the drumsticks are driven by the solenoids. The drummer size is small, no greater than 18in^3. The design for the drummer is a simple lever and fulcrum system where the lever is the drumstick.

Requirements and Verification

Requirement	**Verification**
Requirement 1	*Equipment:*

Drummer strikes drum when solenoid is triggered.	1) Power supply 2) Solenoid and driver circuit *Procedure:* 1) Assemble the drummer with the solenoid and driver circuit. 2) Power up the solenoid driver circuit. 3) Trigger the solenoid and observe the drummer strike the drum. 4) Listen for acoustic noise as the drummer strikes. 5) Confirm that the drummer arm resets after current is cut-off from the solenoid. *Expected Result:* The drummer should strike the drum every time the solenoid is triggered. Then the drummer arm should reset when current is cut-off from the solenoid.

Table 2.8: Drummer Requirements and Verification

Supporting Materials

Mechanical Diagram

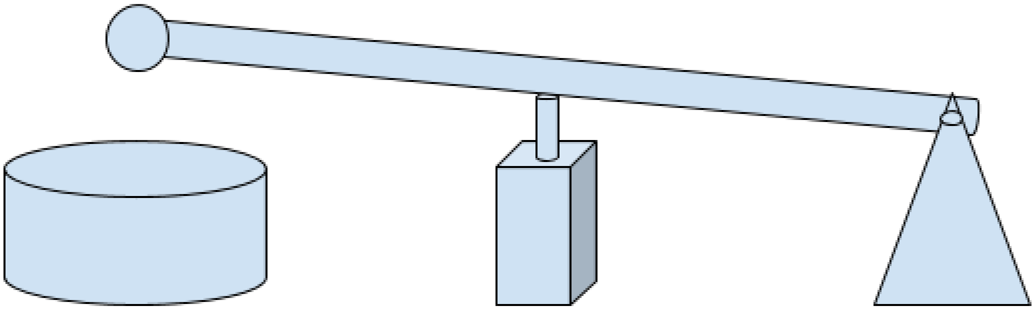

Figure 2.14: Mechanical Drummer Diagram.

2.3 Tolerance Analysis

The ADC of our system must be supplied with a voltage amplitude between the values of 0.75V and 1.65V for adequate sampling. In order to achieve these values, we have implemented a fixed gain amplifier to boost the input signal from the microphone. The max voltage amplitude the microphone can deliver to mic level is 1mV. Our amplifier is designed to have a voltage gain of 1398 V/V, this corresponds to a max line level amplitude of 1.4V. To ensure that the ADC receives adequate signals, we will perform a tolerance analysis on the amplifier. This analysis will only consider the tolerance of the amplifier gain elements which are the feedback resistors and input resistors. We will make the following assumptions: the filter stage after the amplifier is ideal, and the mic level input into the amplifier is constant at 1mV.

Refering to equation 1, the voltage gain of each stage of the amplifier is $-Z_{feedback}/Z_{in}$. The impedance of the resistors is much greater than the impedance of the coupling capacitors so we can assume the gain is $-R_{feedback}/R_{in}$. We will represent the value of each component as follows:

(2.15) $$R_{in} = R_{in}'(1 \pm \Delta R_{in})$$

(2.16) $$R_f = R_f'(1 \pm \Delta R_f)$$

Where R_1' and R_f' represent ideal values, and ΔR_1 and ΔR_f represent percentage error.

The total gain of the 2-stage amplifier is:

(2.17) $$750 \leq \frac{R_f'(1 \pm \Delta R_{f,1}) R_f'(1 \pm \Delta R_{f,2})}{R_{in}'(1 \pm \Delta R_{in,1}) R_{in}'(1 \pm \Delta R_{in,2})} \leq 1650$$

The inequality represents the minimum and maximum gain that the amplifier can have and still meet the tolerance requirement. Although the error terms in (2.17) are uncorrelated, we will assume that corresponding error terms are equal. This will ensure that we calculate the maximum tolerance a component can have.

Let us assume the that R_{in} is a 23kΩ with 5% tolerance, we can solve (2.17) for two cases:

(2.18) $\Delta R_{in} = +10\% \Rightarrow -20.0\% \leq \Delta R_f \leq 15.1\%$

(2.19) $\Delta R_{in} = -10\% \Rightarrow -25.8\% \leq \Delta R_f \leq 3.3\%$

The results show that R_f can have a tolerance of no more than ±3.3% for the given ΔR_{in}.

2.4 Point Assignment

Block Component	Point Value
Power	7
Pre-Amplifier	7
Filter	7
DSP	8
Solenoid Driver	7
Solenoid	7
Drummer	7

Table 2.9: Requirements and Verification Point Assignment

3 Cost and Schedule

3.1 Cost

3.1.1 Labor

Labor costs are calculated based off a $65,000 annual salary with a 50 hour average work week. This is equivalent to a $25.00 hourly wage. We estimate on average 15 hours a week per individual group member, over a 16 week period (1 semester).

3.1.2 Parts

Labor	$18,000.00
Teensy Microcontroller	$33.25
Behringer Microphone	$20.00
Microphone Cable	$9.00
Microphone Stand	$7.58
Microphone Preamplifier	
LM741 (op-amp)$ x2	$1.50
Filter Components	
2 x LM741 Operational Amplifiers	2 x $0.50 = $1.00
Various ceramic capacitor ($\leq\ 0.1uF$)	$0.10
Power Components	
Linear Regulator	$2.00.
Mechanical Components	
Iron Pipe	$4.00
Iron Rod	$2.00
PVC pipe	$0.50
Gorilla Glue	$5.00
Pen Case	$0.00
Magnet Wire	$0.00

Springs	$1.00
PLA	$1.00
TOTAL	**$87.93**

Table 3.1: Parts & Labor Breakdown

3.1.3 Grand Total

Grand Total = Labor + Parts = $18087.93

3.2 Schedule

Task	Group Member	Target Completion Date
Filter Prototype	Craig	2/24/2018
Preamplifier Prototype	Drake	2/24/2018
Actuator Prototype	Drake	2/24/2018
Power Circuit Prototype (Battery, Linear Regulator, Circuit Protection)	Drake	3/10/2018
Characterize Teensy	Nick and Craig	3/10/2018
LED debugging circuit (for output testing)	Craig	3/10/2018
Interface all electrical prototypes	ALL	3/17/2018
PCB designed and ordered	Nick	3/17/2018
Mechanical Drummer	Drake	3/31/2018
Interface all components	ALL	4/7/2018
Sound Detection Algorithm	Nick and Craig	Implemented and refined throughout the semester.

Table 3.2: Project Schedule

4 Discussion of Ethics and Safety

We do acknowledge that, by the standard of the IEEE Code of Ethics [14], tenet #1, and the ACM Code of Ethics and Professional Conduct, General Moral Imperative 1.2 [15], our project does have the potential to injure and shock members of the public through misuse. As such, to the best of our skills and abilities we shall endeavor to mitigate these risks to the end-user and public at large, as demonstrated by the explicit inclusion of safety measures in the more hazardous elements of our design, as well as the observation of appropriate work practices in the lab environment itself.

.

With respect to the former, we aim to insulate sensitive and potentially dangerous circuit components (capacitors, batteries, etc.) from the user by way of a protective, electrically insulated enclosure. Note this would be a final consumer product feature, and will not currently be featured on the prototype.

An additional concern for the end-user pertains to risks of hearing damage, as our device as an acoustic-production aspect to its operation. Per OSHA-issued noise safety standards [16], in the event our realized design achieves threshold of loudness specified therein, we advise the end-user and person within its vicinity of operation to observe OSHA’s recommendations for exposure-limits and/or to take appropriate measures with personal protective equipment (e.g. ear-plugs) as seen fit.

With respect to the latter, some specific way in which we consistently practice proper lab safety techniques includes: never rewiring a circuit while it's still powered, and always powering down our lab equipment (including soldering irons) before leaving the bench to attend to another task.

From the perspective of the development process itself, and in deference to the significance placed on the spirit of collegiality by the IEEE Code of Conduct, tenets #7 and #10, we acknowledge the value inherent to recognizing the skills and background of the team-members, so that sufficient opportunity may be distributed evenly for contribution as well as professional growth. This also has implications for adhering to tenet #9, whereby misrepresentations of knowledge or ability may put members of the team, as well as the public at large, at risk, where the demands of a particular design task carry safety considerations. As such, we will seek out expert knowledge and ability to assist us in the design process, if and when such circumstances arise.

From the perspective of issuing proper credit to third-parties’ ideas and work, per the ACM Code of Ethics and Professional Conduct, General Moral Guideline 1.6, we have and shall continue to recognize the sources of inspiration for solutions to engineering challenges applied, as has been demonstrated throughout this document. It is not our intent to misrepresent our work, and will do our utmost to chronicle and cite the sources of our methodologies and implementations accordingly.

5 Citations

[1] Evolver.fm (2011). *The Beatbox Machine Turns Your 'Beats' into Actual Drumbeats* [Online]. Available:
http://evolver.fm/2011/09/26/the-beatbox-machine-turns-your-beats-into-actual-drumbeats/

[2] Synthtopia (2010). *New App Lets you Beatbox to MIDI On Your iPhone* [Online]. Available:
http://www.synthtopia.com/content/2010/09/07/new-app-lets-you-beatbox-into-your-phone-edit-the-beat-on-your-mpc/

[3] Human Beatbox (2005). *History of Beatbox: Old School* [Online]. Available:
https://www.humanbeatbox.com/articles/history-of-beatboxing-part-2/

[4] Sound On Sound (2007). *Q: How much latency is acceptable for virtual piano?* [Online]. Available:
https://www.soundonsound.com/sound-advice/q-how-much-latency-acceptable-virtual-piano

[5] *ITU Relative Timing of Sound and Vision for Broadcasting*, ITU Recommendation BT.1359-1 (1998). Available:
https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.1359-1-199811-I!!PDF-E.pdf

[6] Texas Instruments (2018) "FilterPro" [Web Application]. Available:
http://www.ti.com/design-tools/signal-chain-design/webench-filters.html

[7] Fourier.eng.hmc.edu. (2018). *Properties of Fourier Transform*. [online] Available at:
http://fourier.eng.hmc.edu/e101/lectures/handout3/node2.html [Accessed 25 Feb. 2018].

[8] Linear Technology (2017). "LtSpice" [Computer Application]

[9] Y. Hu, P. Liu, and S. Meng, "Low-Cost Solution of Thermal Cycler for LifeFoundry, Inc.: ECE 445 Mock Design Document - Fall 2017," ECE 445 Course Materials [Online].

[10] PJRC.com. *Tutorial 4: Analog Input* [Online]. Available:
https://www.pjrc.com/teensy/tutorial4.html

[11] P. Stoffregen (2015). "FFT Test" [Source Code]. Available:
https://github.com/PaulStoffregen/Audio/blob/master/examples/Analysis/FFT/FFT.ino

[12] Wikipedia. *Polylactic Acid.* [Online].
Available: https://en.wikipedia.org/wiki/Polylactic_acid

[13] AKsteel (2007). *430 Stainless Steel Product Data Sheet.* [Online]. Available:
http://www.aksteel.com/pdf/markets_products/stainless/ferritic/430_data_sheet.pdf

[14] IEEE.org. (2018). *IEEE Code of Ethics.* [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html

[15] ACM.org (20180. *ACM Code of Ethics and Professional Conduct.* [Online]. Available: https://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct

[16] *Occupational Noise Exposure*, OSHA Recommendation 1910.95 (Amended 2008). Available: https://www.osha.gov/pls/oshaweb/owadisp.show_document?p_table=STANDARDS&p_id=9735