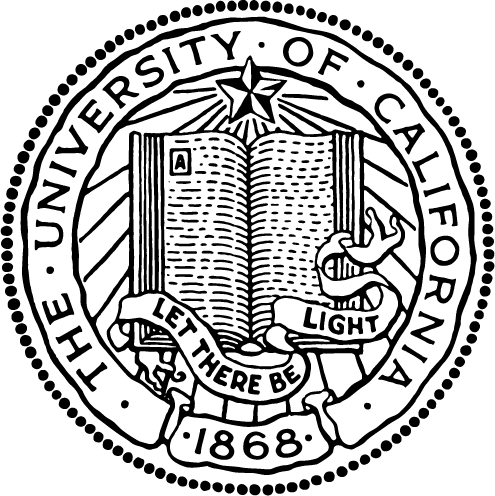
**EE 175A/B Final Report**

****

**Digitally Controlled Analog Filters**

**EE 175AB Final Report**

**Department of Electrical Engineering, UC Riverside**

|  |  |
| --- | --- |
| **Project Team Member(s)** | Ramiro Carrillo (SID:86212223)  Henry Chen (SID:862110773)  Michael Tang(SID:861209107)  Jeffrey Lee(SID:861214171) |
| **Date Submitted** | March 15, 2020 |
| **Section  Professor** | Section 022  Dr. Roman Chomko |
| **Revision** | Revision 1.0 |
| **URL of Project Wiki/Webpage** | https://git.io/JviJC  https://youtu.be/iTVZIgoqJM8 |
| **Permanent Emails of all team members** | rcarr026@ucr.edu  henryc3472@gmail.com  michaeltang528@gmail.com  jlee395@ucr.edu |

**Summary**

This report represents the collaborative efforts of the four team members behind DCAFF(Digitally Controlled Analog Filters and FFT) over the course of twenty weeks. This report outlines the complete implementation of DCAFF, the high and low level architectural design, the design considerations, and the numerous tests and experiments conducted to insure the full-functionality of the product. For reader consideration, the code used for implementation and the video-demonstrations are posted on the first page.

**Note:**

* **Sections marked with \* are required**
* **In each section, you must clearly identify which team member is responsible for which objectives, modules or tasks.**

**Revisions**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Description of Version:** | **Author(s)** | **Date Completed** | **Approval** |
| *.3* | *Initial draft. All prompts are highlighted red. Filled out all sections except 10-14* | *Ramiro Carrillo, Henry Chen* | *3/06/20* | *PASS* |
| *.5* | *2nd Draft. All sections are filled in. Sent to Professor Hossny for review.* | *Ramiro Carrillo, Henry Chen,Michael Tang* | *3/14/20* | *PASS* |
| 1.0 | *Final Report. Last minute updates are taken into consideration.* | *Henry Chen,*  *Ramiro Carrillo,*  *Michael Tang* | *3/15/20* | *PASS* |

**Table of Contents**

[**\* Executive Summary**](#_g9h6gd2ugu0b) **7**

[**\* Introduction**](#_cv218omh173t) **8**

[\* Design Objectives and System Overview](#_e5q9jomf55q0) 8

[\* Backgrounds and Prior Art](#_sw0wr713g3fm) 10

[\* Development Environment and Tools](#_2s8eyo1) 10

[\* Related Documents and Supporting Materials](#_17dp8vu) 12

[\* Definitions and Acronyms](#_26in1rg) 12

[**\* Design Considerations**](#_w29orlnnm9lm) **13**

[*\* Realistic Constraints*](#_35nkun2) *13*

[System Environment and External Interfaces](#_1ksv4uv) 13

[\*Industry Standards](#_2jxsxqh) 14

[\*Knowledge and Skills](#_z337ya) 14

[\* Budget and Cost Analysis](#_3j2qqm3) 15

[\*Safety](#_1y810tw) 16

[Performance, Security, Quality, Reliability, Aesthetics etc.](#_4i7ojhp) 16

[\*Documentation](#_2xcytpi) 17

[Risks and Volatile Areas](#_bttmc5bhwa9d) 18

[**\* Experiment Design and Feasibility Study**](#_jeha1qrs6tez) **19**

[\* Experiment Design](#_2bn6wsx) 19

[\* Experiment Results, Data Analysis and Feasibility](#_qsh70q) 21

[**\* Architecture and High Level Design**](#_87mvgwt5o9hm) **31**

[\* System Architecture and Design](#_1pxezwc) 31

[\* Hardware Architecture](#_2p2csry) 32

[\* Software Architecture (only required if your design includes software)](#_147n2zr) 33

[\* Rationale and Alternatives](#_3o7alnk) 34

[**Data Structures (include if used)**](#_spl83ku2pq1n) **35**

[Internal Software Data Structure](#_32hioqz) 35

[**\* Low Level Design**](#_r731mp383kib) **36**

[\*Low Level Design](#_vx1227) 36

[Teensy Module](#_uyt2hioerx8y) 37

[*Processing narrative for module Teensy*](#_1v1yuxt) *37*

[*Module Teensy processing details*](#_4f1mdlm) *37*

[Low Level Design](#_o1xwev5mvvc) 38

[Processing narrative for module: Python](#_al7bgpw6wq4o) 38

[Module Python processing details](#_adpeavxet78u) 38

[Low Level Design](#_egx0cmkvdc42) 39

[Amplifier and Speaker Module](#_s8eugmgms75b) 39

[Amplifier and Speaker Module interface description](#_bqdab5hh5h65) 40

[Amplifier and Speaker Module processing details](#_lhi2p6vzmx6c) 40

[**\* Technical Problem Solving**](#_u172l93pydh9) **41**

[8.1.1\* The ADC Sample Problem](#_19c6y18) 41

[8.1.2\* Solving the ADC Sample Problem](#_u4dzuzme702g) 41

[8.2.1\* The Serial Communication Problem](#_qaqow9s0gk9a) 41

[8.2.2\* Solving the Serial Communication Problem](#_vfdqs7xixsgz) 41

[8.3.1\* The Audio Formatting Problem](#_sh1cuizc75cg) 42

[8.3.2\* Solving the Audio Formatting Problem](#_v9csk3xitohg) 42

[8.4.1\* The Concurrent State Machine Problem](#_iv0rh8avsjgf) 42

[8.4.2\* Solving the Concurrent State Machine Problem](#_9ja152uf9ze) 42

[8.5.1\* The Filter Design Problem](#_xsso2lwxbyg1) 43

[8.5.2\* Solving the Filter Design Problem](#_egzsqfoqav8u) 43

[8.6.1\* The Mainloop() Problem](#_k0alqqhzhnjv) 43

[8.6.2\* Solving the Mainloop() Problem](#_eptdfeesh9gt) 43

[8.7.1\* The Speaker Audio Output Problem](#_1u07ib5616lv) 44

[8.7.2\* Solving the Speaker Audio Output Problem](#_wdautcjr448t) 44

[**User Interface Design**](#_28h4qwu) **45**

[Application Control](#_nmf14n) 45

[User Interface Screens](#_37m2jsg) 45

[User Interface Breadboard](#_chlap2fmhlt1) 47

[**\* Test Plan**](#_1mrcu09) **48**

[\* Test Design](#_46r0co2) 48

[\* Bug Tracking](#_2lwamvv) 53

[\* Quality Control](#_111kx3o) 53

[\* Identification of critical components](#_3l18frh) 56

[\* Items Not Tested by the Experiments](#_206ipza) 56

[**\* Test Report**](#_92kud1pfnc4i) **57**

[\* Test: Signal Processing and GUI (Ramiro Carrillo and Henry Chen)](#_2zbgiuw) 57

[\* Test 1: Data Acquisition](#_1egqt2p) 58

[\* Test 2: Teensy Functionality](#_t54074xwrteg) 59

[11.5 \* Class-D Amplifier and Speaker](#_qo7jyt6uhx8) 68

[11.6.1 \* Power Efficiency](#_980rcowsfs6s) 68

[11.6.2 \* Power Efficiency(Repeat:1)](#_8t6cp8wwb6sy) 69

[11.7.1 \* Total Harmonic Distortion](#_199v9umn8agc) 69

[11.7.2 \* Total Harmonic Distortion(Repeat:1)](#_r9rvhcfrcwry) 69

[**\* Conclusion and Future Work**](#_2dlolyb) **70**

[\* Conclusion](#_sqyw64) 70

[Future Work](#_3cqmetx) 71

[\* Acknowledgement](#_1rvwp1q) 72

[**\* References**](#_p29ma87gv7hq) **73**

[**\* Appendices**](#_2r0uhxc) **74**

# **\* Executive Summary**

The overall motivation behind DCAFF is to project a user-friendly environment for filtering audible signals. This environment can be used by the average person who wishes to tune his guitar, to a professional setting where audio filtering is vital. DCAFF can be used to reduce background-static noise, or to isolate desired frequencies through the implementation of an nth order butterworth filter. Essentially, this device can be used for a variety of audio-filtering applications while projecting a user-friendly environment.

The goal behind DCAFF is to implement a variety of customizable filters, while giving the user full control. This is achieved through a graphic interface which enables the user to create a low, high, or bandpass nth order butterworth filter with desired cutt-off frequencies. The interface displays corresponding graphs pertaining to the original and filtered signals, which include their fast-fourier transform and power density, along with the display of the filter and its frequency response.

DCAFF works by converting a continuous-audio signal to its discrete-digital counterpart. The signal is filtered and returned to an audio signal, where it is outputted via a class-D amplifier. DCAFF has six components: ADC/DAC conversion, microcontroller for task management, computer for filtering, a storage device used for signal data, a GUI for user interface, and an amplifier for outputting signals. DCAFF takes a signal from a microphone, converts the signal to digital. The digital signal is processed by the microcontroller and saved to the storage device. From the storage device, the data is transmitted to a computer, which is filtered using an nth butterworth filter. The data pertaining to the signal is displayed on the GUI. The filtered audio is sent back to the microcontroller and outputted to the amplifier.

A key feature behind DCAFF is the user interface. It can be utilized through buttons and knobs for the user who wants to simply filter an audio signal, or through the GUI interface to view key data behind the filtered signal.

Each individual component has been tested to ensure it is working properly. The components are integrated into the overall system ,where the system itself has gone through numerous tests. The main components in this test include: Analog-to-digital conversion, digital-to-analog conversion, microcontroller task implementation, communication protocols between architectures, proper filtering , and class-D amplifier speaker output.

Important achievements include: analog to digital and digital to analog conversion, implementing a high, low, and bandpass Nth order Butterworth filter, SPI communication to and from sd card, serial communication to and from PC, applying a GUI interface, audio formatting, audible signal output.

For project DCAFF, members are using Systems Engineering when executing the project design process, viability, and member responsibility management.

# **\* Introduction**

## **\* Design Objectives and System Overview**

The design of the project is mainly focused on allowing the user to flexibly control an audio signal. In order to maximize customization, the user will be able to use real buttons, knobs, and a graphic user interface to manipulate the audio signal alongside the Nth order Butterworth filter applied to the audio signal. This project is related to electrical engineering due to the heavy amount of signal processing, circuit design, and circuit assembly involved. In electrical engineering, it is important to analyze signals and process the signal based on specifications. In this project, the audio signal will be taken in and manipulated from voltage reading up into formatting the audio signal into a standard audio file. This project also heavily involves embedded systems as it integrates the Arduino environment and Python environment to properly relay USART communication.

The second design of the project is focused on an additional Class-D Amplifier and speaker module. This module is built with cheaper, fewer ICs and components for lower cost and can be bought at a lower price than other models. The Amplifier and Speaker Module should be user-friendly: simply plug in the audio jack, turn on the system, and use the module immediately. This portion of the project is related to electrical engineering due to communications, signal processing, circuit design, and circuit assembly. In electrical engineering and in everyday occurrences, it is important to take in sounds such as human speech and musical instruments as well as digital equivalents to reproduce, modify, and transfer auditory information to a different space and time beyond the digital plane. One example is when one stationary person needs to make a school-wide announcement.

The system structure is mainly focused on the graphic user interface (GUI) as the main control system. The graphic user interface will use the Python environment to interact with the microphone through the Arduino environment in the background by constantly reading into the serial bus and vice versa. The GUI will wait for a serial input and then process the signal. GUI will output the filtered signal by sending the data back to the microcontroller and wait for the user to playback the audio. The user will have an option to output the original signal or the filtered signal through a speaker.

List of technical design objectives:

When filtering a sinusoidal wave made up of three frequencies, the filtered frequency must be within 90% of its expected frequency after reaching steady state. This is to test the accuracy of our filters, and no audio is involved.

The same test listed above will be repeated. But, the sinusoidal wave will be outputted through the audio spectrum. The audio will be filtered and compared to the expected filtered frequency. If the filtered wave is within 90% of the expected frequency, the test is passed. This will test the entirety of our project, from ADC conversion, Serial Communication, and filtering.

In order to test the ADC, we will be playing a tone with a single frequency. We will obtain the frequency played on the oscilloscope. We will then obtain the frequency from the digital signal produced through Python. If the frequencies are within 90% matching, the test passed.

DAC will be tested similarly. The analog frequency will be measured on the oscilloscope. The digital frequency will be measured on Python. If the frequencies are within 90% of each other, the test passed

Audio formatting: little-endian to discrete signal to little-endian. This will test the accuracy of the function we wrote to change the raw file sent through serial communication to a discrete signal and back to a raw file. If the data is 90% similar when the original and converted raw files are compared to one another, the test is passed.

Speaker Audio Output: To test the Class D Amplifier and Speaker, the Amplifier will be measured with a multimeter and oscilloscope for audio distortion. A signal generator is also used to generate an audible sine wave to simulate an incoming audio signal. There is a DC voltage power supply to keep the amplifier and speaker system operational. If the speaker outputs an audible signal, then this test is passed.

Amplifier Power Loss: To test the Class D Amplifier and Speaker, the Amplifier will be measured with a multimeter and oscilloscope for efficiency and, by extension, power loss. A signal generator is also used to generate an audible sine wave to simulate an incoming audio signal. There is a DC voltage power supply to keep the amplifier and speaker system operational. If the efficiency is above 90 percent of power, then this test is passed.

The cost to make this project should be below $200. If we spend more we fail the test, if we spend less, we pass.

The project will be split into 2 groups: audio processing and audio amplification. The audio processing group will consist of Henry Chen and Ramiro Carrillo. The audio amplification group consists of Michael Tang and Jeffrey Lee.

Within the Audio Processing group, Henry’s responsibilities will be audio formatting, signal processing, embedded systems programming ,filter designing, and GUI development. Ramiro’s responsibilities will include Analog to digital, precise serial communication, Arduino embedded programming, filter designing, and GUI development.

Within the audio amplification group, Michael’s responsibilities will be Amplifier schematics, Amplifier circuit assembly, general troubleshooting, and secondary testing. Jeffrey’s responsibilities will be Amplifier troubleshooting and primary testing.

## **\* Backgrounds and Prior Art**

A similar product can be found with a Google search is Audacity, Logic Pro, and FL Studios. In all these software products, the idea of filtering/ editing is done in some way. Some advantages to our design is how detailed the filters can be manipulated. Since most software will only run a default filter, there is no customization for the user to change cutoff frequencies or order. With our design, there is more information given to the user. Our amplifier and speaker are cheap. Some disadvantages to our design is that our speaker audio is a little quiet and does not readily receive power from a battery or usb cable, using a DC voltage power supply instead.

## **\* Development Environment and Tools**

**Software:**

The following software tools/environments were used: Arduino IDE, Python 2.7, Fritzing, and Audacity

**Individual Software Uses:**

The design-environments used are: Spyder 2.7 for Python programming and Arduino Integrated development for microcontroller programming.

**Hardware:**

The following physical components were used: Wires, breadboard, buttons, potentiometers, sd card, Teensy 3.2, SGTL-5000 Audio Shield, Electret Microphone (0-20kHz) , PC, USB-A to micro-USB cable, Audio Jack, Class D Amplifier and Op-Amp ICs, 4 Ohm and 3 Watt Speaker.

**Individual Hardware Use:**

Wires were used to connect buttons, potentiometer, and the SGTL shield to the Teensy 3.2. Breadboard was used to mount the button, wires, Teensy 3.2, audio shield, and potentiometers. Buttons were used to enable the low, high, and bandpass filtering done on the PC. They were also used to record, stop recording, and playback audio from the Teensy 3.2. Potentiometers were used to set the cut-off frequencies for the Butterworth filters on the PC. SD card was used to save raw data coming in from the microphone with SPI communication. Teensy 3.2 was used to communicate and send data between the microphone, SD card, SGTL-5000 Audio Shield, and Python. SGTL- 5000 Audio Shield was used for ADC and DAC conversion. Microphone was used to pick up analog signals between 0 and 20K Hz. PC was used to process audio signals and GUI interface.USB-A to micro-USB was used for serial communication between pc and Teensy 3.2

Wires were used to connect potentiometer, ICs, speaker, capacitors, and inductors on the Class-D Amplifier and Speaker Breadboard. Breadboard was used to mount the potentiometer, ICs, speaker, capacitors, diodes, and inductors. Electrolytic and ceramic capacitors are used to create a high pass connection to reduce noise between components. One example is the 1000 uF electrolytic capacitor near the end of the circuit right before the speaker. Diodes were originally used in the first Class D Amplifier prototype designed by GreatScottLab to ensure that the output of the IRL233ZN Mosfet Driver moved in a loop to ensure that the square wave is stable and concise. Inductors are used as the second half of a Inductor-Capacitor Low Pass Filter right before the speaker to cut off the high frequency square wave into an audible sound wave that the speaker accepts easily. The potentiometer is used to change the voltage offset of incoming signals and waves to create desirable outputs for the project. A 4 Ohm, 3 Watt Speaker is used as the audio output for the entire project. The TPA 3122 IC is used as the main Class D Amplifier of the entire project using a combination of Gaalaass’, Chirila’s, and Afrotechmod’s ideas and theory, including my gathered knowledge, which amplifies incoming audio signals into high frequency square waves and back before the audio signal is outputted from the speaker. The LM 386 Power Amplifier IC is used to boost incoming voltage levels to increase the output voltage peak to peak for a louder audio output.

**Testing equipment:**

The following test equipment were used: Python, digital and analog signal generators, multimeter, and oscilloscope

**Testing equipment use:**

Oscilloscope was used for testing ADC and DAC conversion. Python was used to create digital signals in order to write test files to simulate waves at certain frequencies. Signal generator was used for testing ADC on the microphone. Multimeter was used to measure resistances, current, voltage, and continuity between the circuit and its components.

**Software tools:**

The following software tools were used :Audacity, and Teensyduino.

Audacity was used to read raw files created on Teensy for testing purposes. Teensyduino was used to allow the Teensy 3.2 to use the Arduino IDE.

Fritzing Circuit Design program was used to design breadboard circuits used for better visualization of the circuit design structure, presentation, and general understanding.

**AMPLIFIER SPEAKER**

**Hardware:** wires, breadboard, TLC 2300 Amplifier IC, LM 386 Power Amplifier IC, potentiometer, ceramic and electrolytic capacitors, local power supply, 4 Ohm / 3Watt Speaker, and audio jack.

Wires were used to connect ICs, potentiometer, speaker, and capacitors.

Breadboard was used to mount the button, wires, Teensy 3.2, audio shield, and potentiometers.

Potentiometers were used to fix the voltage offset for the incoming audio signal.

Speaker is used to output an analog signal between 0 and 20K Hz

TPA 3122 IC is used to convert incoming audio signals into an amplified signal output.

LM 386 Power Amplifier IC is used to increase the incoming voltage to produce a louder audio signal.

Testing equipment: oscilloscope, analog signal generators, and multimeter

Oscilloscope was used for measuring voltage, voltage peak-to-peak, and digital and audio signals.

Signal generator was for testing components on circuits.

## **\* Related Documents and Supporting Materials**

**Serial communication, SPI, and raw format(little-endian)**

The audio formatting done on the audio signal is formatted to an industry standard. The signal is a standard raw file with 16bit, 44,100 samples per second,and pulse code modulated (PCM) that is used heavily as a base for many audio formats. The audio file can be converted into a WAV file by inputting the proper header. Furthermore, the audio can be compressed using Huffman compression. This will turn the audio file into a traditional MP3 file.

**HP DC Voltage Power Supply**

This industry standard power supply outputs a wide range of DC voltages (0 to 20+ Volts) needed to activate the components for the amplifier and speaker. The ‘Overload’ function is especially useful for automatically lowering voltages that would otherwise short-circuit the audio output system.

## **\* Definitions and Acronyms**

List any project definitions and acronyms introduced to the project by this design.

DCAFF- Digitally Controlled Audio Filter + FFT

PC = Personal Computer

PCM = Pulse Code Modulation

GUI = Graphic User Interface

IC= Integrated Circuit

# **\* Design Considerations**

## **\* Realistic Constraints**

**Processing Constraints**

* USART can only process at 9600 baud rate (8 bits per stream)
* Microphone can only work in range 0-20kHz

**Memory Constraints**

* The audio files total storage cannot exceed 16GB (max amount of SD storage)

**Filter Constraints**

* Nth order Butterworth Filter is an IIR filter unstable at high orders of 22.

**Time Constraint**

* Had to stop using our written filters (HPF,LPF , and BPF) due to time. Code is still provided. Reason was that the conversion for plotting took too much time. Forced to use Butterworth function from SciPy because of difficulty restructuring results from filter. Graphs would not look right because they need the zero, poles, and gain to graph. No simple way to implement a function like tf() or c2d() in Python with our specification. (needed a bilinear transformation for s to z).

**Output Constraints**

* Amplifier only works for signals produced at 5 volts peak-to-peak
* Amplifier isn’t built to be adjustable to amplify small signals (2 volt peak to peak)

**Power Constraints**

* Due to no battery management system, a power supply is necessary to power the amplifier

**Budget Constraints**

* Budget cannot exceed 200$

## **System Environment and External Interfaces**

This is a list of software and hardware needed in order to operate this project.

1. Arduino IDE
2. PC needs to be running Windows 7 or higher.
3. Python libraries needed:
   1. matplotlib
   2. serial
   3. sounddevice
   4. math
   5. cmath
   6. numpy
   7. scipy
   8. tkinter
4. Python 2.7 or higher needs to be used
5. Amplifier uses an external voltage supply of 12V.

## **\*Industry Standards**

The project uses standard RS232 serial communication to relay audio information from the Teensy to PC. I2C to transform data between DAC and ADC with a sampling rate of 44.1 k Hz, and 16 bit precision. SPI to transmit data between the Teensy and the sd card storage device.

Raw file format (16bit PCM 44,100 sample rate)

DC Voltage Power Supply 0 to 20+ V

## **\*Knowledge and Skills**

**Ramiro:**

Previous knowledge:

* Computer Science (CS 061)
* C or equivalent (CS 012, CS 010)
* Serial Communication
* Microcontroller programming (120A, 120B)
* Circuit design (EE001A, EE001B)
* Signal Processing (EE110A,EE110B,EE141,EE128)

New Knowledge:

* Python Programming
* SPI/I2C/UART communication
* Standard audio formating
* Data acquisition

**Henry:**

Previous Knowledge:

* Computer Science (CS 061)
* C or equivalent (CS 012, CS 010)
* Microcontroller programming (120A, 120B)
* Serial Communication
* Signal Processing (EE110A,EE110B,EE141,EE128)

New Knowledge:

* Standard Audio Formatting
* Data Acquisition
* Python Programming
* SPI

**Michael:**

Previous Knowledge:

* Basic Python Programming
* Basic Soldering
* Signal Processing (EE110A,EE110B,EE141,EE128)
* Circuit Design and Assembly (EE001A, EE001B, EE100A, EE100B)

New Knowledge:

* Theoretical and Conceptual Aspects of a Class D Amplifier

**Jeffrey:**

Previous Knowledge:

* Circuit Design (EE001A, EE001B, EE100A, EE100B)
* Signal Processing (EE110A, EE110B, EE128, EE141)

New Knowledge:

* Basic Soldering
* Very Basic Python Programming
* Theoretical and Conceptual Aspects of a Class D Amplifier

## **\* Budget and Cost Analysis**

The budget for the DCAFF is meant to be under $200. The following Figure 3.5 displays the items, prices, and total cost of the project.

|  |  |
| --- | --- |
| **Material:** | **Cost:** |
| Teensy 3.2 | $21.94 |
| Breadboard | $9.99 |
| Potentiometer Knobs | - |
| Audio Shield | $21.35 |
| Arduino Uno | $11.98 |
| Audio Jack IC | $4.00 |
| Audi Jack Cable | - |
| TPA 3122 Class-D Stereo Amplifier IC | $3.31 |
| IR2113-2 Mosfet Driver IC | $2.40 |
| LM 386 Power Amplifier IC | $0.70 |
| TLC 555 Timer IC | $0.90 |
| 74HC04 Hex Inverter IC | $0.80 |
| LM 393 Comparator IC | $0.46 |
| IRLZ44N Mosfet Driver IC | $2.00 |
| Ceramic and Electrolytic Capacitors | $5.21 |
| 4 Ohm, 3 Watt Speaker | $4 |
| 33uH Shielded Inductors | $6.75 |
| LM 7805 and LM 7812 Voltage Regulators | $0.93 |
| Resistors | - |
| 4007 Diodes | $0.10 |
| Poster Board for Presentation | $30.00 |
| Total | $96.82 |

*Figure 3.5 Budget Chart*

## **\*Safety**

Do not touch components when active. Consequences range from static shock, paralysis, or death by electrocution.

## **Performance, Security, Quality, Reliability, Aesthetics etc.**

**Aesthetics:**

GUI developed to be user friendly and organize functions and graphs

Amplifier and Speaker developed to be cheap and easy to use

**Quality Control:**

Had to make sure buttons and potentiometers worked by measuring the current.

Tested Serial Communication be sending data to pc and printing out to display.

Tested Serial Communication by sending data to Teensy and writing data to raw file.

Tested little endian to discrete signal conversion by using Audacity to play the raw and compare it to the discrete signal.

Tested discrete signal to little endian conversion by converting a little endian raw file to a discrete signal, and converting the discrete signal into raw file. And compared the converted original raw file to the newly created. This ensured that no data was lost.

Tested the speaker's overall volume by running the amplifier and playing music from a computer or smartphone.

**Reliability:**

Ran the Teensy 3.2 and Python code over night. In the morning Ramiro went to see if it was still running as intended. The Teensy and Python modules were still running as intended.

Ran the system numerous times to see if it would crash. It didn’t.

Ran the amplifier overnight (approximately 7 to 10 hours). Upon inspection, the amplifier and speaker are still running without overheating.

## **\*Documentation**

Every week a weekly report is generated from each team member.

The code written for the Arduino and Python code is documented by versions. Each version represents a component being added:

1. The beta tests version of the code were written for the Arduino Uno and Atmega 1284. The first versions of the code will be mainly in Arduino in ensuring serial communication between Arduino and Python work perfectly.
2. The next version of the code is generated by combining the Python and Arduino code together. The purpose of this version is to integrate the Arduino embedded system code and Python embedded system code to make sure all the data is properly sent. Afterwards, the Python code is written to send the data back into Arduino. At this version, Arduino code is done and is not touched for the remainder of the project.
3. In this version the Python code is able to read and respond to all serial communication from the Arduino. The raw audio data is sent serially from the Arduino into the Python code. The Python code is able to format the audio into a NumPy array to prepare the signal to be filtered. In this stage the design of the Nth order butterworth filter is implemented and tested. In addition, code to customize the Butterworth filter is implemented.
4. The final version of the code mainly focuses on developing the GUI for the user to be able to customize the filter. The GUI will show the original signal, the filtered signal, original FFT, filtered FFT, Power Spectral Density, Frequency Response, and graph of the Nth order Butterworth filter.

The figures added to the weekly reports represent components used, design schematics, and system block diagrams. New figures featuring components are used in a pinout form to indicate the need to make correct assumptions about wire connections between parts. Some figures represent schematics of current and past systems used in the project. Finally, system block diagrams help the reader understand an idea or provide an overview of the project.

**3.8.1 \*System Engineering**

**System Requirement Document**

* Defines system level functional and performance requirements for a system
* <https://docs.google.com/document/d/1_a8y07TGJ7p4ZNquWabxfo1ZEMxf7Ek8xymhIGfNGcs/edit?usp=sharing>

**Verification matrix**

* Document that links requirements throughout the validation process.
* <https://docs.google.com/spreadsheets/d/1G5Hni7vkYpDt2OcUKh2zke6CeKvqU2fSNPJUOXl1akM/edit?usp=sharing>

**Critical design Review**

* A Critical Design Review (CDR) is a multi-disciplined technical review to ensure that a system can proceed into fabrication, demonstration, and test and can meet stated performance requirements within cost, schedule, and risk.
* N/A

**GANTT**

* Shows activities (tasks or events) with a timeframe
* <https://docs.google.com/spreadsheets/d/1G5Hni7vkYpDt2OcUKh2zke6CeKvqU2fSNPJUOXl1akM/edit?usp=sharing>

**Preliminary Design Report**

* Report consists of design constraints, methods, considerations, and methods of implementation.
* <https://docs.google.com/presentation/d/1HgzPF1Jxc1Xbt0I7lubGXgpbe3XC9X5F2i7EaorHSAQ/edit?usp=sharing>

**Statement of Work**

* Details scope of work, period of performance, place of performance, work requirements, schedule, and acceptance criteria

<https://drive.google.com/a/ucr.edu/file/d/1NnMC8VwE5eJZIX3N-ns-M9eytnAMcPDU/view?usp=sharing>

**System Architecture**

* Functional Architecture Diagram
* System (Physical) Architecture

## **Risks and Volatile Areas**

During the first quarter of the project, DCAFF members were working on ADC and DAC conversion using an atmega1284, and later the Arduino Uno.When the full implementation of ADC was completed, the group found that the sampling rate for the Arduino Uno was not suitable for 44.1k Hz sampling and 16 bit sample precision. Due to time constraints, the team had to choose to either replace the Uno with a microcontroller that could handle a higher sampling rate, and risk not being able to get the components working, and not being able to demonstrate a working prototype for the first presentation, or to stick with the current 20k sampling rate and 8-bit precision of the Uno, and move forward with DAC conversion. As far as audio quality is concerned, 8-bit quality is not recommended for it is of terrible quality and not up to par with industry standards.

The group decided to implement a new microcontroller architecture--the Teensy 3.2--and risk not being able to demo a working prototype. 24 hours were spent heavily researching the Teensy 3.2 before ordering the product to ensure that when the product arrived, the team would be able to assemble it as fast as possible. The group made sure that the code implemented on the Uno would fully work with the Teensy 3.2, and that processing speed, serial communication, and audio quality and sampling requirements corresponded with the needs of DCAFF were met.

When the product arrived, the group was able to integrate the new component with the entirety of the project, and were able to demo a working prototype.

# **\* Experiment Design and Feasibility Study**

## **\* Experiment Design**

**4.1.1**

**Formatting Raw Data**

This experiment was designed by Henry Chen and Ramiro Carrillo

Objective: This experiment is to test if the raw audio signals are properly converted into an audio signal.

Setup: Teensy 3.2, PC, 2 computers

Procedure:

1.Record audio from microphone

2.Use Python code to convert into an audio waveform

3. Compare waveform by importing raw data into Audacity and comparing waveforms

Result: The waveform generated from our code and the waveform generated from Audacity are very similar.

**4.1.2**

**Dexterity of Filters (low, high, and bandpass)**

This experiment was designed by Ramiro Carrillo and Henry Chen

Objective: Test if the low, high, and bandpass work at isolating frequencies

Setup: Teensy 3.2,PC, 2 computers

Procedure:

1. Play a sinusoidal wave consisting of three frequencies: 1500, 2500, 3500 Hz
2. Filter each frequency with a low, high, and bandpass
3. Retrieve each filtered signal

Result: It is expected that the returned filtered signals will show the isolated 1,500, 2,500, and 3,500 Hz signal corresponding to the filter used. The number of samples for each isolated frequency should also correspond to the original signal’s frequency samples.

**4.1.3**

**Noise Removal**

This experiment was designed by Ramiro Carrillo and Henry Chen

Objective: Test how well the filters work at noise removal

Setup: Teensy 3.2, PC, 2 computers(or a tone generator)

Procedure:

1. Create and play noise
2. Talk to the microphone
3. Use a low pass filter between with frequency 1000 Hz
4. Play back the filtered signal

Result: The expected result should be a returned signal with most noise removed with the original signal produced when talked to the microphone still audible.

**4.1.4**

**Serial Communication for big file transfer**

This experiment was designed by Ramiro Carrillo and Henry Chen

Objective: test the quality of an audio signal being transferred via serial communication

Setup: Teensy 3.2, PC, 1 computer

Procedure:

1. Record an audible signal
2. Transfer it from Python to pc
3. Retrieve each signal before and after it was transferred

Result: The expected result should be two signal that are distinguishable from one another

**4.1.5**

**TLC 555 Timer Triangular Wave Generation**

This experiment was designed by Michael Tang and Jeffrey Lee

Objective: Test the TLC 555 Timer IC for a triangular output

Setup: TLC555 Timer IC, DC Voltage Power Supply, LM 7805 Voltage Regulator, 0.1uF ceramic or plastic capacitor, 1kΩ resistor, wires, breadboard, Oscilloscope

Procedure:

1. Set up the TLC 555 timer with above mentioned components and equipment
2. Use oscilloscope probes on the output
3. Turn on Power Supply
4. View oscilloscope results

Result: The expected result should be a triangle wave with a Vmax of two-thirds of the incoming Vcc (5V) and a Vmin of one-third of the incoming Vcc (5V)

**4.1.6**

**LM 393 Comparator Square Wave Output**

This experiment was designed by Michael Tang and Jeffrey Lee

Objective: Test the LM 393 Comparator for a square wave output

Setup: LM393 Comparator IC, Signal Generator, DC Voltage Power Supply, 0.1uF ceramic or plastic capacitor, 10kΩ resistor, wires, breadboard, Oscilloscope

Procedure:

1. Set up the LM 393 IC with above mentioned components and equipment
2. Use oscilloscope probes on the output (Pin 1)
3. Use signal generator at 1kHz, Z-Load, 5 Vpp, triangle Wave
4. Connect signal generator output to non-inverting input of the LM393
5. Connect 2.5 V to inverting input of the LM393
6. Turn on Power Supply
7. Adjust frequency of signal generator and view oscilloscope results

Result: The expected result should be a square wave with a high frequency (over 200kHz) and a Vmin of 0 V and a Vmax of 5 V

**4.1.7**

**IR2113 Mosfet Driver Double Output**

This experiment was designed by Michael Tang and Jeffrey Lee

Objective: Test for two square wave outputs from the IR2113 Mosfet Driver IC

Setup: IR2113 Mosfet Driver IC, 74HC04 Hex Inverter IC, LM393 Comparator IC, Audio Jack Cable, Signal Generator, Audio Jack IC, TLC555 Timer IC, DC Voltage Power Supply, LM 7805 Voltage Regulator, 0.1uF ceramic or plastic capacitor, 1kΩ and 10kΩ resistors, LM4007 diodes, wires, breadboard, Oscilloscope

Procedure:

1. Set up the IR2113 Mosfet Driver IC with above mentioned components and equipment
2. Use oscilloscope probes on the outputs (Pin 8 and 14)
3. Use signal generator at 1kHz, Z-Load, 10 Vpp, Sine Wave
4. Connect signal generator output to audio jack cable
5. Turn on Power Supply and allow signal generation output
6. View oscilloscope results

Result: The expected result should be one square wave with high frequency (over 200kHz) and high voltage values (3 to 5 volts) and one square wave with high frequency (over 200kHz) and low voltage values (-5 to 2 volts).

**4.1.8**

**LM386 Low Voltage Audio Power Amplifier**

This experiment was designed by Jeffrey Lee

Objective: Amplify the audio jack input for the class D amplifier

Setup: Audio Jack IC, class D amplifier circuit, LM386 IC, Oscilloscope, DC Voltage Power Supply

Procedure:

1. Set up LM386 with above components
2. Use oscilloscope probes at original input and amplified input
3. Turn on Power Supply
4. View oscilloscope results

Result: The expected result should be a signal that is amplified by up to 20 dB

## **\* Experiment Results, Data Analysis and Feasibility**

**4.2.1**

**Formatting Raw Data**

|  |
| --- |
| **Code** |
|  |

|  |  |
| --- | --- |
| **Audacity** | **Python** |
| *Figure 4.2.1-1: Audacity Output* | *Figure 4.2.1.-2 Python Output* |

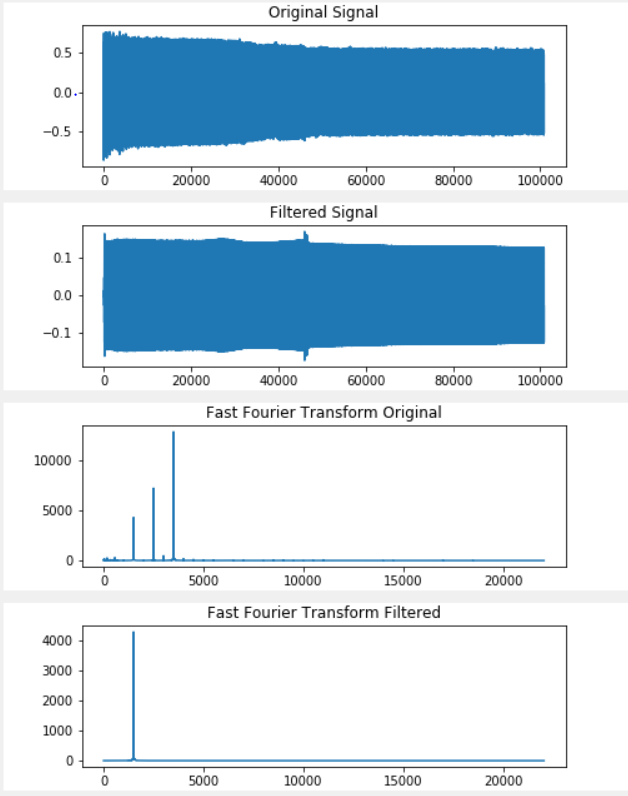
Figure 4.2.1-1 displays the raw data transformation shows an audio waveform similar to Audacity. The signal shown in Figure 4.2.1-2 is between negative one and one ,which is just like in Audacity.

**4.2.2**

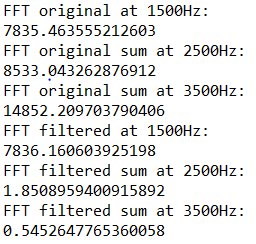
**Dexterity of Filters(low,high, and bandpass)**

**Lowpass**:

The following display shows a low pass filter with cutoff frequency of 2,000 Hz implemented on a signal consisting of 1500, 2500, and 3,500 Hz. The output shows the filtered signal and its corresponding FFT.

****

*Figure 4.2.2-1:Lowpass Graphs*

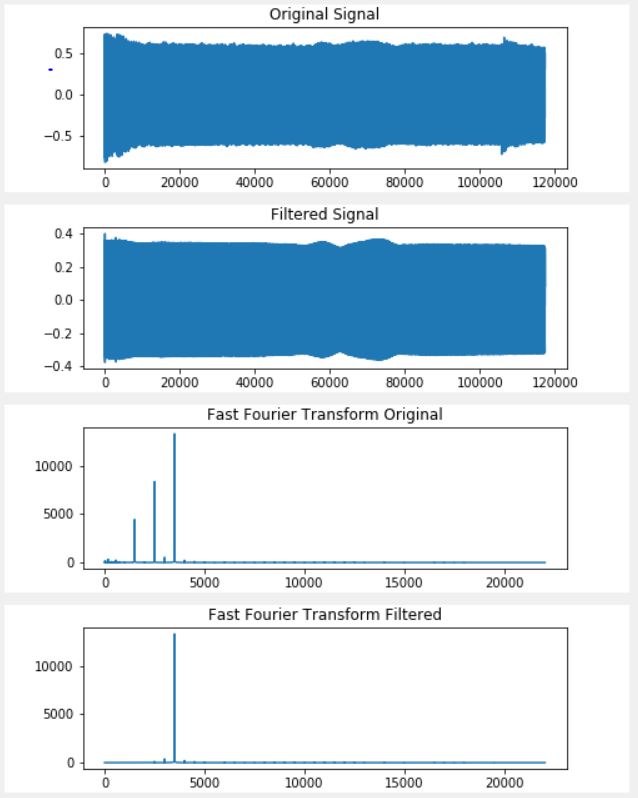
****

*Figure 4.2.2-2:Lowpass FFT Samples*

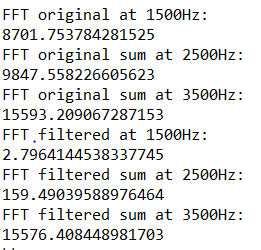
The experiment proved to be successful where in Figure 4.2.2-1 the returned FFTof the filtered signal shows only the 1500 Hz signal, without the 2,500 and 3,500. Furthermore the returned 1,500 Hz signal has the expected number of samples shown on the FFT of the original signal. On the original the fft 7835 samples at 1500Hz were found. On the filtered 7836 samples in Figure 4.2.2-2, giving us a percent error of 0.0012%

**Highpass**:

The following display shows a high pass filter with cutoff frequency of 3,000 Hz implemented on a signal consisting of 1500, 2500, and 3,500 Hz. The output shows the filtered signal and its corresponding FFT.

****

*Figure 4.2.2-3:Highpass Graphs*

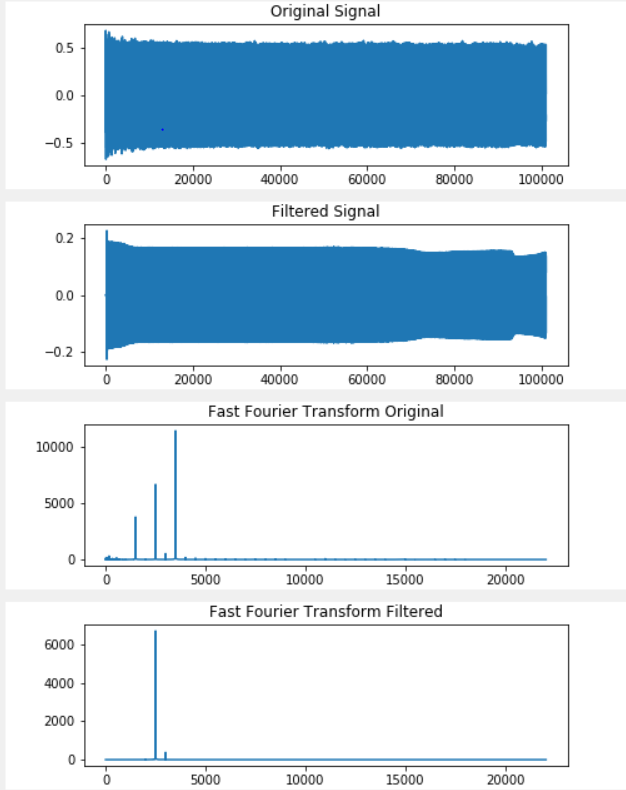
****

*Figure 4.2.2-4:Highpass FFT Samples*

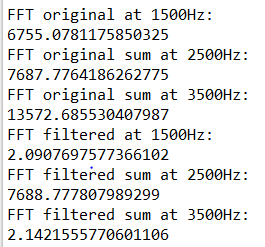
The experiment proved to be successful where the returned fft of the filtered signal in Figure 4.2.2-3 shows only the 3,500 Hz signal, without the 1,500 and 2,500. Furthermore the returned 3,500 Hz signal has the expected number of samples shown on the FFT of the original signal. For the original signal 15593 samples at 3,500 Hz are found. The filtered shows 15576 samples. Giving us a percent error of 0.0011%. All FFT samples are in Figure 4.2.2-4.

**Bandpass**:

The following display shows a band pass filter with cutoff frequency of 2,000 and 3,000 Hz implemented on a signal consisting of 1500, 2500, and 3,500 Hz. The output shows the filtered signal and its corresponding FFT.

****

*Figure 4.2.2-5:Bandpass Graphs*

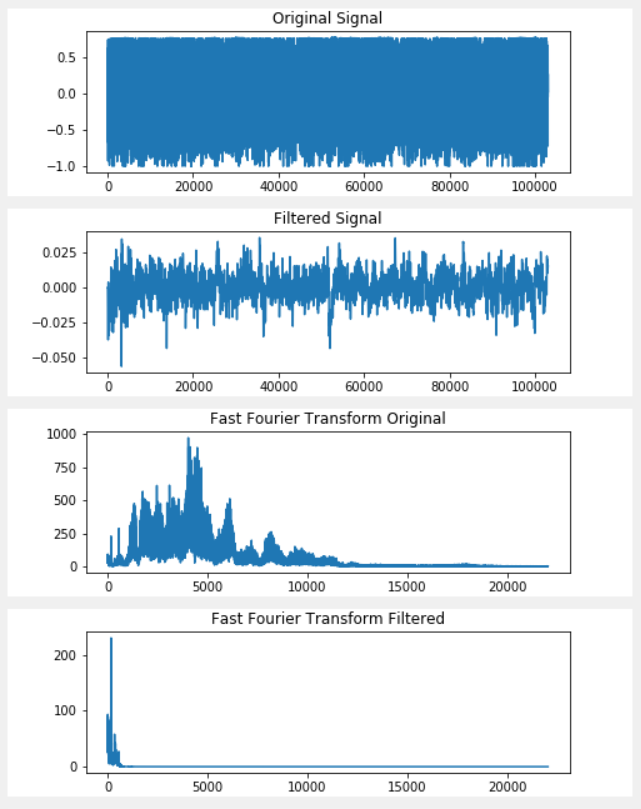
****

*Figure 4.2.2-6:Bandpass FFT Samples*

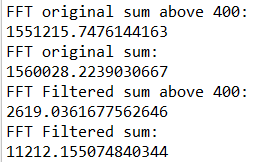
The experiment proved to be successful where the returned fft of the filtered signal in Figure 4.2.2-5 shows only the 2,500 Hz signal, without the 1,500 and 3,500 Hz. Furthermore the returned 2,500 Hz signal has the expected number of samples shown on the FFT of the original signal. For the FFT of the original signal had 7687 samples, and the filtered signal had 7688, giving us a percent error of .00012%. All samples are in Figure 4.2.2-6.

**4.2.3**

**Noise Removal**

****

*Figure 4.2.3-1:Bandpass Noise Removal Graphs*

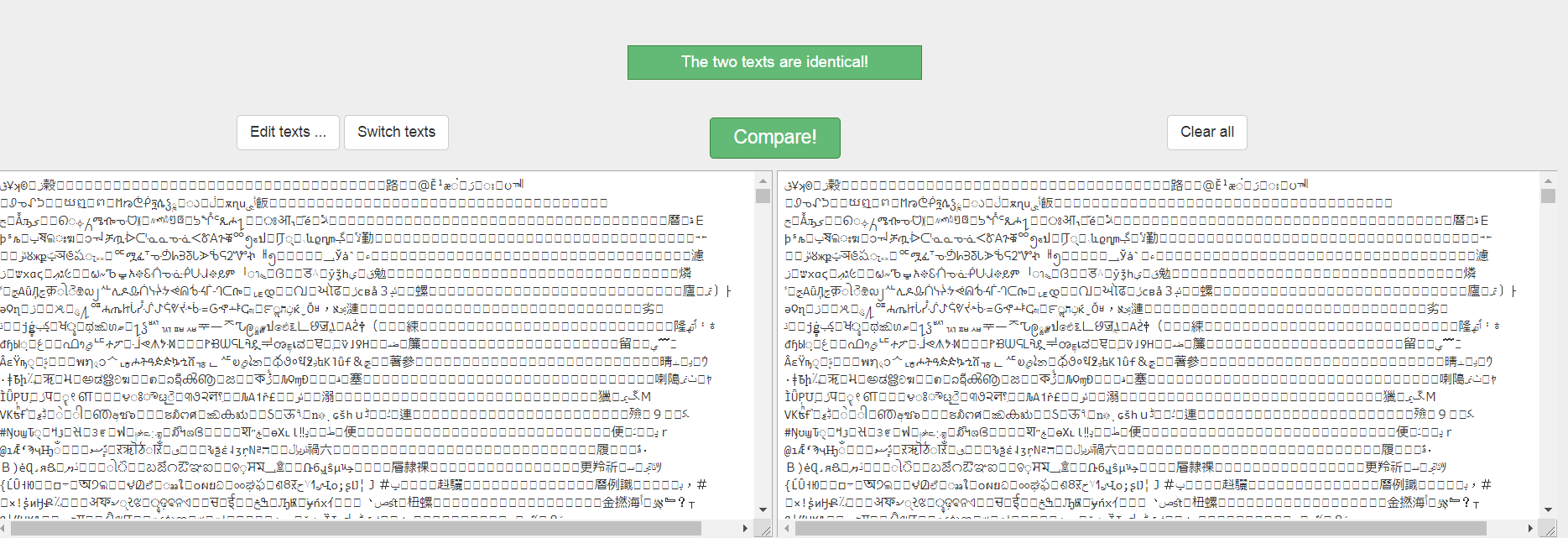


*Figure 4.2.3-2:Bandpass Noise Removal FFT Values*

The Original Signal and the Filtered Signal can be seen in the GUI image above in Figure 4.2.3-1. The filter is set to a bandpass setting with cutoffs at 0 and 400Hz. The returned number of samples for the original filtered audio above 400 Hz is 1551215, and the for thee filtered 2619 samples, giving us a percent error of 0.0011682%. All samples are in Figure 4.2.3-2 above.

**4.2.4**

**Serial Communication for big file transfer**

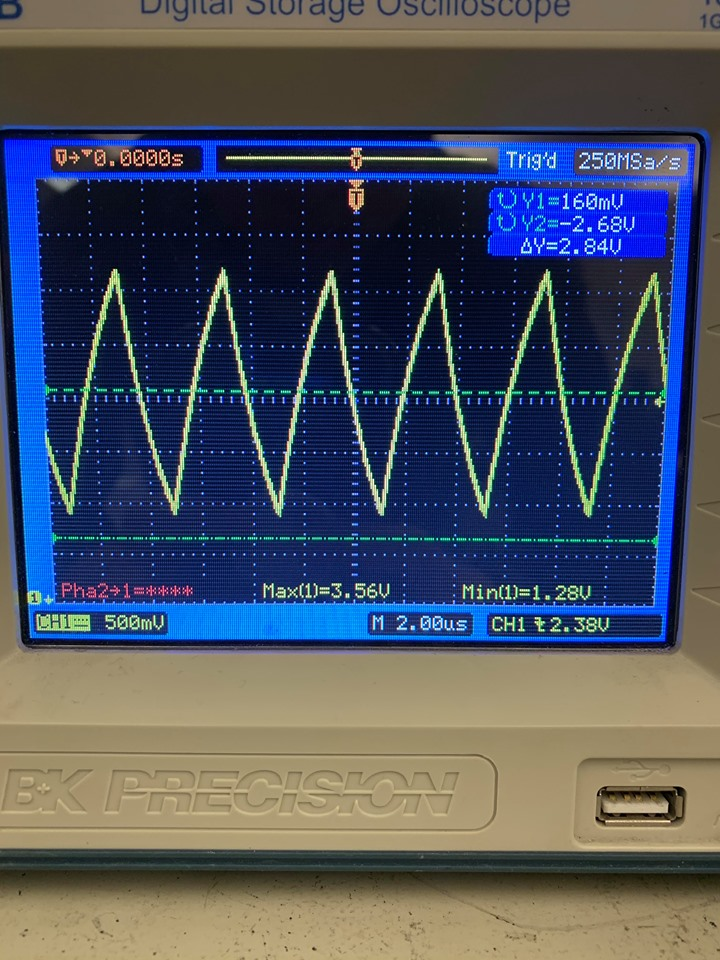


*Figure 4.2.4: Serial files comparison*

After conducting the experiment and comparing the original file before being transferred and after it was transferred, the files were 100% similar. No data was lost.

**4.2.5**

**TLC 555 Timer Triangular Wave Generation**

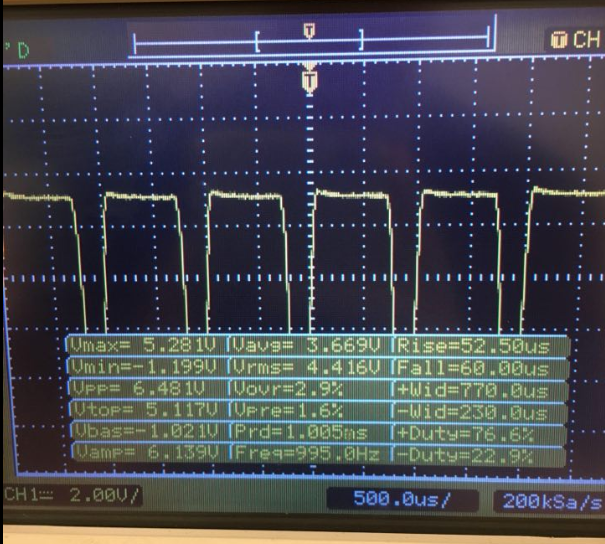
****

*Figure 4.2.5-1: TLC 555 Timer IC Output*

According to Figure 4.2.5-1, the oscilloscope displays a triangle wave with a Vmax of around two-thirds of the 5 volt supply and a Vmin of around one-third of the 5 volt supply at 10x scope. A successful test.

**4.2.6**

**LM 393 Comparator Square Wave Output**



*Figure 4.2.6-1: LM 393 Comparator Output*

According to Figure 4.2.6-1, the Oscilloscope displays a square wave with Vmin of -1.199V and a Vmax of 5.281V. There is a 1 V offset so it is not the desired 0V to 5V output. A failed test.

**4.2.7**

**IR2113 Mosfet Driver Double Output**

|  |  |
| --- | --- |
| High Output (HO) Square Wave | Low Output (LO) Square Wave |
|  |  |

*Figures 4.2.7-1 and 4.2.7-2: High Output and Low Output of Mosfet Driver of IRL223 IC*

According to Figure 4.2.7-1, the High Output Square Wave voltages are greater than the voltages displayed on the Low Output Square Wave displayed in Figure 4.2.7-2. The ‘valleys’ displayed on the Low Output Square Wave exist because of a missing capacitor that was not connected to generate the cut-off on that particular connection. Placing the probe at a different connection that is connected to the missing capacitor fixes that problem. A successful test.

**4.2.8**

**LM386 Output**

|  |  |
| --- | --- |
| **Before Amplifier** | **After Amplifier** |
|  |  |

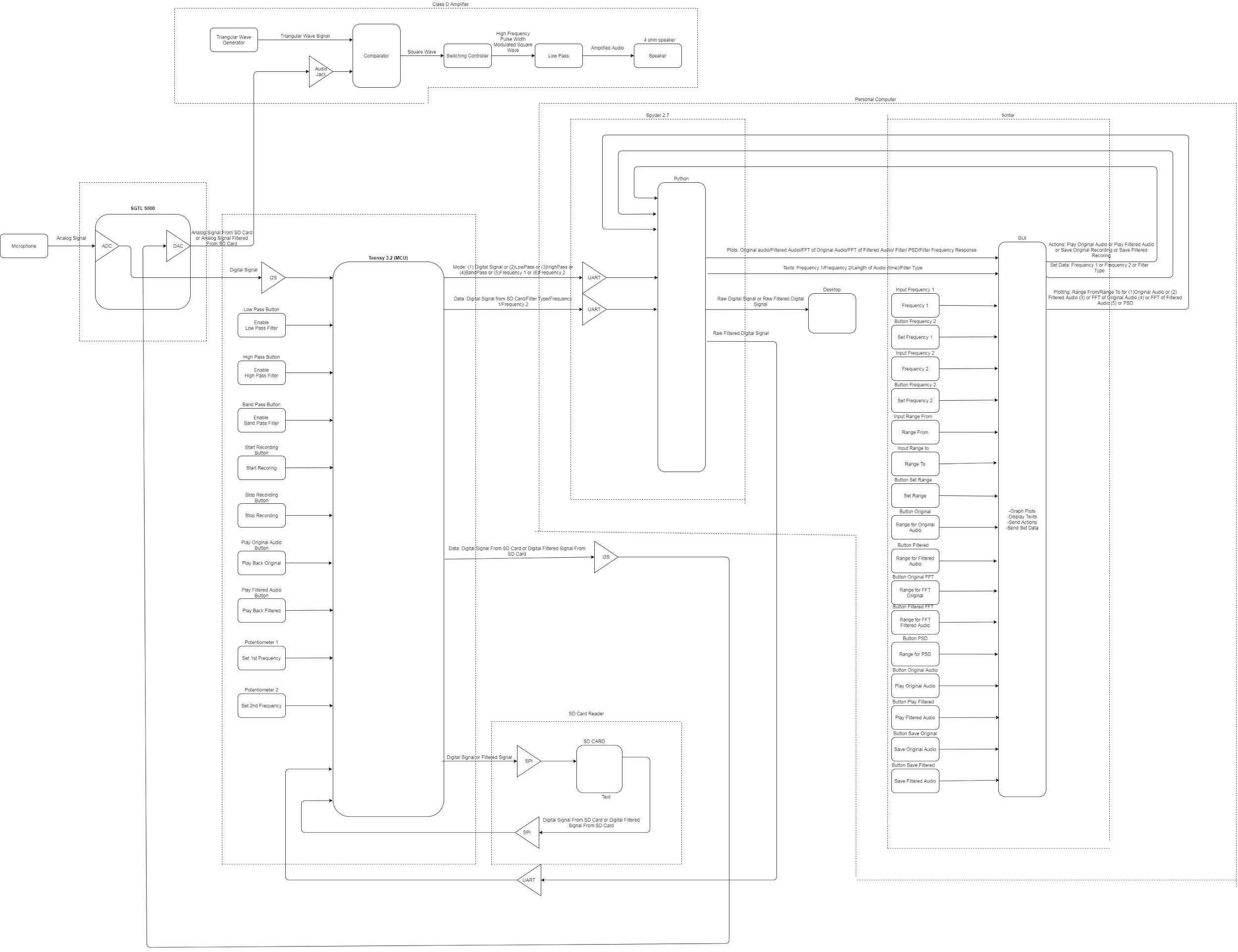
*Figures 4.2.8-1 and 4.2.8-2: Before and After LM 386 Amplifier IC Output*

The max voltage shown in Figure 4.2.8-1 before the amplifier is 2.24 V and after the amplifier is 5.68 V in Figure 4.2.8-2. It is clear that the signal was amplified. From a calculator online, it was found that the gain was about 8 dB.

# **\* Architecture and High Level Design**

## **\* System Architecture and Design**

The system architecture consists of 6 parts: ADC and DAC conversion/ Data storage on SD card/ Teensy audio processing/ Python filtering/ GUI interface/Class D amplifier. The figure below shows the system block diagram. The diagram can be copied and pasted to a new window that will allow the reader to further inspect it.



*Figure 5.1: System Block Diagram*

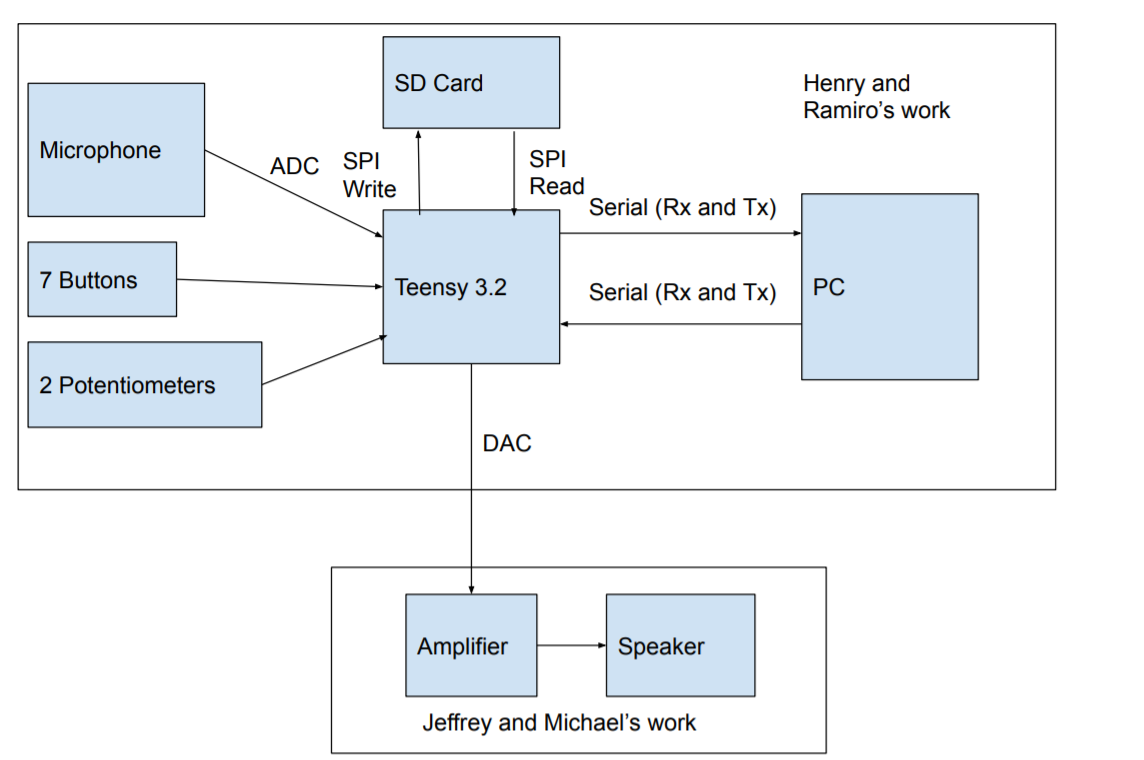
In Figure 5.1, the system block diagram above shows the route the signal takes as it is processed. It first is picked up by the microphone, converted to a digital signal, and transferred to the Teensy 3.2. The Teensy passes the signal to the SD card, retrieves it from the SD card, and passes it on to Python. Python filters the signal. Python generates corresponding graphs and displays them on the GUI. Python returns the filtered signal to the Teensy. The Teensy saves the filtered signal on the sd card, and retrieves it from the sd card. The signal is converted back to analog, and sent to the amplifier and speaker.

The system structure is mainly focused on the graphic user interface (GUI) as the main control system. The graphic user interface will use the Python environment to interact with the microphone through the Arduino environment in the background by constantly reading into the serial bus and vice versa. The GUI will wait for a serial input and then process the signal. GUI will output the filtered signal by sending the data back to the microcontroller and wait for the user to playback the audio. The user will have an option to output the original signal or the filtered signal through a speaker.

## **\* Hardware Architecture**

The hardware architecture in Figure 5.2 consists of signal processing and audio output. The signal processing is done by Henry and Ramiro. The signal processing blocks include: microphone, SD card , Teensy 3.2, and PC.

The audio output is done by Michael and Jeffrey. The main components for the audio output are the amplifier and speaker.



*Figure 5.2:Hardware Architecture Diagram*

## **\* Software Architecture (only required if your design includes software)**



*Figure 5.3:Software Architecture Diagram*

**Contributions for Figure 5.3: Henry and Ramiro**

Design: 3 Asynchronous State Machines (Teensy 3.2, Python Data Processing, and GUI)

**Teensy 3.2**

-Uses modes for recording, filters and Playback

**Python Data Processing**

**-**Waits for a character transferred from Python Mode Flag

-Depending on the character, the Python Data Processing unit will perform different tasks(For

Example, 0 means transfer whole files, 1,2, or 3 filter types , or 4 for playback)

**GUI**

-State machine uses After() (a tKinter function), to invoke the entire Python Data Processing unit

-Outputs graphs and relevant data when recording is done

-Updates info as user inputs information

## **\* Rationale and Alternatives**

The reason most of the coding was done on Python was the time constraint. Ideally, this project would be coded completely in C or C++ to optimize runtime. However, since this project is only 20 weeks, Python would be decent enough to satisfy project requirements.

The reason DCAFF uses an Audio Shield is to find the cheapest way to get 16 bit ADC, 16 bit DAC ,and 44,100 sample rate. Alternatively, a Raspberry Pi would have suffice ,but comparing the cost of the Teensy 3.2 and Audio Shield (~40$) and the current price of the Raspberry Pi at that time (~60$) plus an external ADC and DAC (~20), the Teensy 3.2 was the cheaper option by ~40$.

The reasons why the Amplifier and Speaker system has simplified from being built from scratch to using a Class D amplifier IC chip was the time constraint and budget.. Since the original class D amplifier was not able to produce an audio signal for its relatively higher cost, the alternative was a cheaper premade system that is able to produce an audio signal.

# **Data Structures (include if used)**

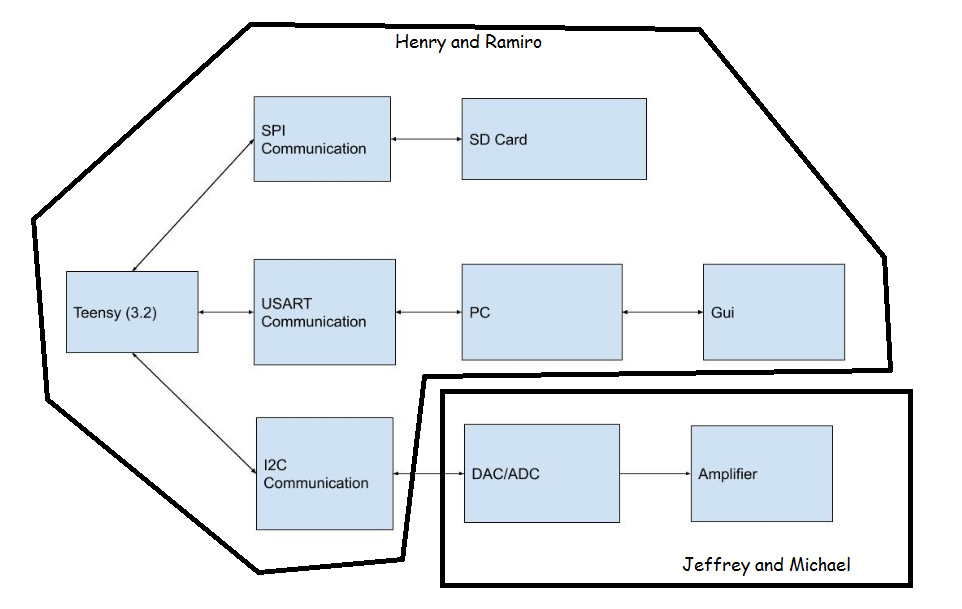
## **Internal Software Data Structure**

Not Applicable

* 1. **Global data Structure**

The global structure in Figure 6.1 below shows how data is sent between the SD card, PC, DAC/ADC, GUI, and the amplifier

**Global Data Structure**



*Figure 6.1:Global Data Structure Diagram*

* 1. **Temporary Data Structure**

Not Applicable

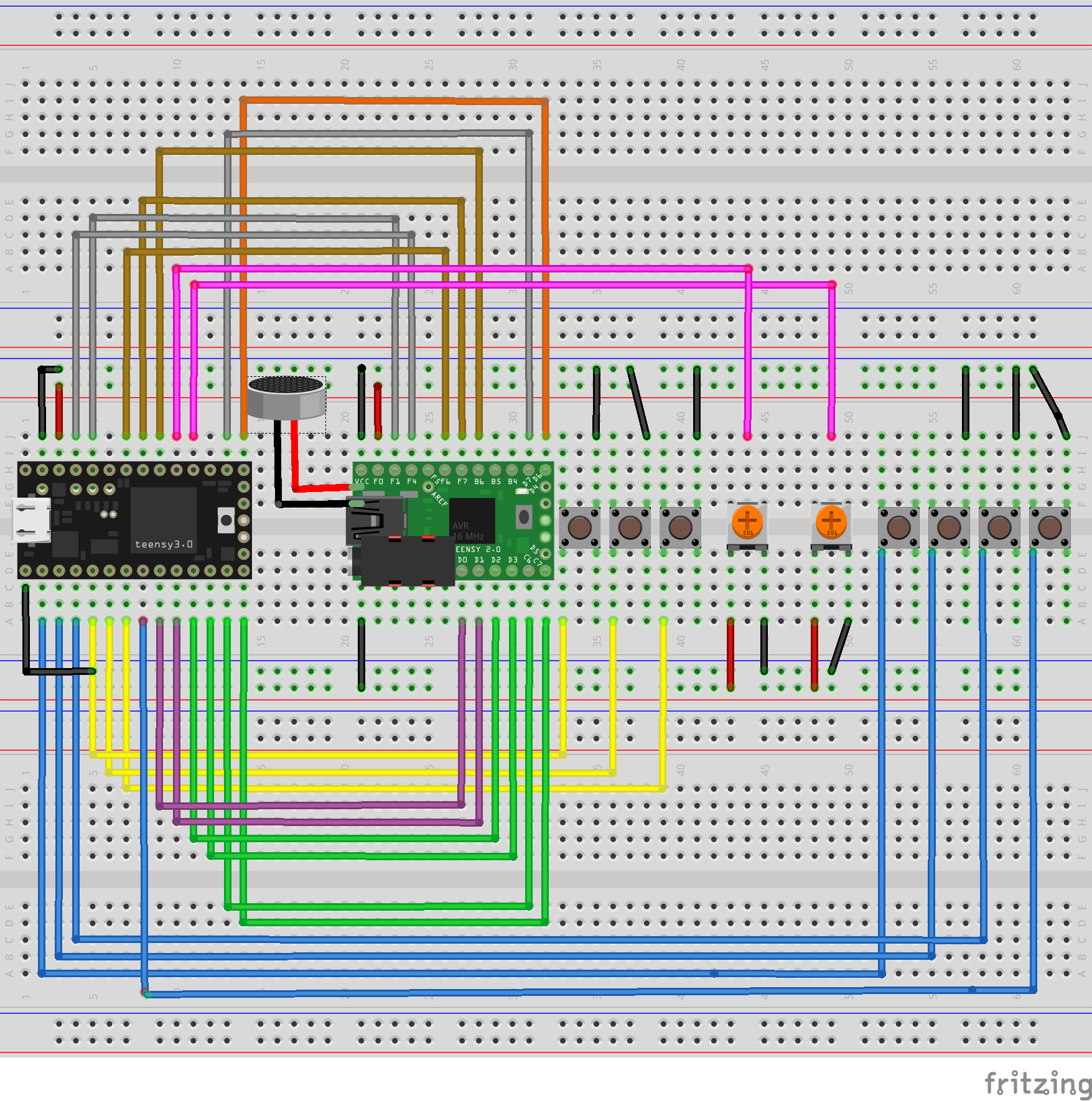
* 1. **Database Descriptions**

Not Applicable

# **\* Low Level Design**

## ***\*Low Level Design***

The module below shows how the Teensy is wired:



*Figure 7.1:Teensy Module Pinout Diagram*

### ***Teensy Module***

The following is a what the color of each wire in Figure 7.1 corresponds to:

**Yellow:** Low, High, Band Pass Filters Buttons(in descending order: left to right)

**Blue:** Record, Stop Recording, Play Recording, Play Filtered Recording Buttons(in descending order: left to right)

**Pink:** Set 1st frequency, 2nd frequency Potentiometers(in descending order: left to right)

**Green:** Save Recording to SD Card

**Purple:** SPI Communication in and out

**Brown:** I2S Communication In from Teensy

**Grey:** I2S Communication Out from Teensy

**Orange:** I2S ready to send/receive

**Black:** Ground

**Red:** 3.3 V

### ***Processing narrative for module Teensy***

The microphone picks up an analog signal which is transformed to a digital signal using the brown wires. The signal is sent to the teensy using the purple wires for audio processing [2]. The signal is then sent back to the SGTL 5000 using the purple wires. The orange wires communicate when data is ready for transfer. The signal is then sent out to the amplifier.

The yellow, blue, and pink wires communicate user input of filters, frequency, and record and audio play information.

### ***Module Teensy* processing details**

Some of the hardware used where the teensy for audio processing, the SGTL 5000 for ADC and DAC conversion, the SD card for holding digital signals, and buttons and potentiometers to set filters and frequencies.

**Algorithms:**

* ADC conversion
* DAC Conversion
* Digital to RAW format

**Design Constraints/Limitations:**

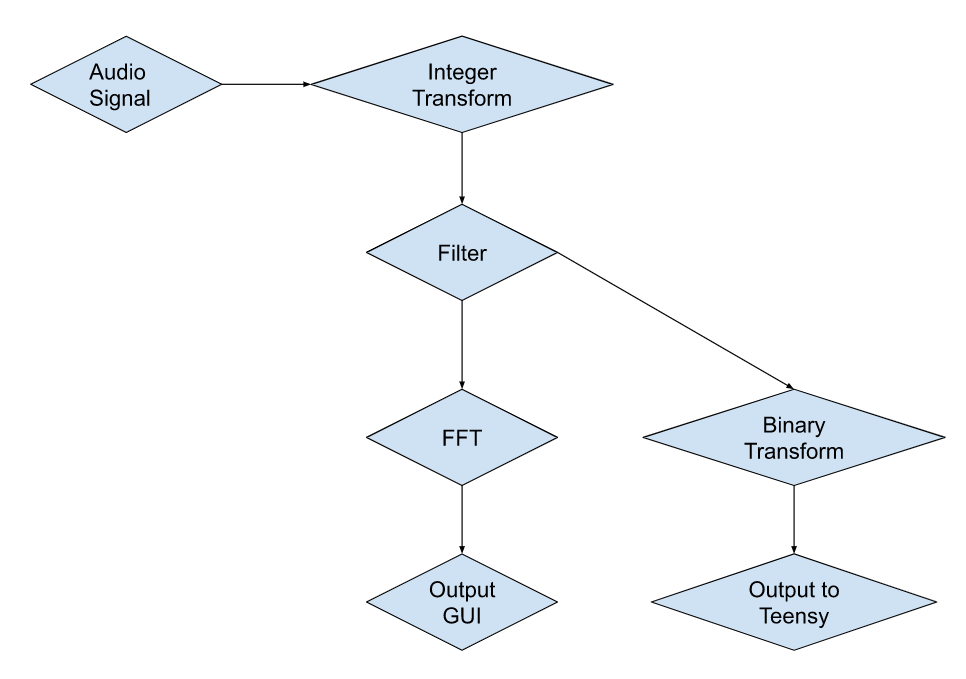
* 8 bit serial transfer
* 9600 Baud Rate (maximum baud rate with USB connection)

**Performance issues:**

* Slow big data transfer

## ***Low Level Design***

* + 1. ***Module Python***

******

*Figure 7.2:Flowchart of Python code*

### ***Processing narrative for module: Python***

From Figure 7.2, the Audio signal is taken in from the Teensy via Serial communication. The integer transform is then applied to the audio signal. Then, the filters are applied to the signal (default filter is a lowpass filter with a 1kHz cutoff frequency). The filtered signal is further processed by applying an FFT and outputting to the GUI. The filtered signal is also transformed into a binary signal using an algorithm. The binary audio signal is outputted back into the Teensy using Serial Communications.

### ***Module Python* processing details**

**Algorithms:**

* Integer Transform
* FFT
* Binary Transform
* Filters (lowpass,highpass, and bandpass)

**Design Constraints/Limitations:**

* Requires user input to be correct (due to time constraint not all edge cases are coded in)
* No operations when audio data is being processed and sent
* PC must be good (more than 4GB RAM)
* Can only process 16 bit samples

**Performance issues:**

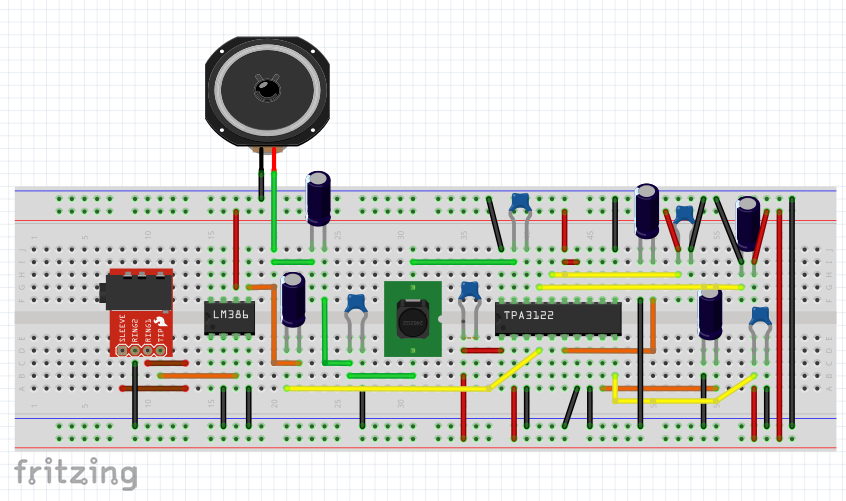
* Slow serial transfer

### ***Low Level Design***

### ***Amplifier and Speaker Module***

The module below shows how the Teensy is wired:

Amplifier and Speaker Module



*Figure 7.3:Class D Amplifier Schematic*

The following is a what the color of each wire in Figure 7.3 corresponds to:

**Yellow**: Input connections into TPA3122

**Green**: Low and High Pass Filters

**Brown**: Audio Signal Input from Audio Jack

**Orange**: High Pass output to reduce high frequency

**Black:** Ground

**Red:** 12 V

### ***Amplifier and Speaker Module* interface description**

The Amplifier and Speaker Module in Figure 7.3 uses an audio jack cable to connect to Amplifier and Speaker Module’s audio jack IC and Teensy Module’s output jack. An audio signal from the Teeny Module is sent between the two modules, one way only. The Amplifier and Speaker Module does not send any input or output back to the Teensy Module.

### ***Amplifier and Speaker Module* processing details**

The Amplifier and Speaker Module in Figure 7.3 uses hardware such as a potentiometer, ICs, speaker, capacitors, and inductors on the Class-D Amplifier Breadboard. Electrolytic and ceramic capacitors are used to create a high pass connection to reduce noise between components. Diodes were originally used in the first Class D Amplifier prototype designed by GreatScottLabs to ensure that the output of the IRL233ZN Mosfet Driver moved in a loop to ensure that the square wave is stable and concise. Inductors are used as the second half of an Inductor-Capacitor Low Pass Filter right before the speaker to cut off the high frequency square wave into an audible sound wave that the speaker accepts easily. The potentiometer is used to change the voltage offset of incoming signals and waves to create desirable outputs for the project. A 4 Ohm, 3 Watt Speaker is used as the audio output for the entire project. The TPA 3122 IC is used as the main Class D Amplifier of the entire project, a combination of Gaalaass, Chirila, Afrotechmods, and my own works which amplifies incoming audio signals into high frequency square waves and back before the audio signal is outputted from the speaker. The LM 386 Power Amplifier IC is used to boost incoming voltage levels to increase the output voltage peak to peak for a louder audio output.

Some module design constraints are size, weight, and cost. This module is kept small to around the size of the user’s hand, indicating that space is very limited and smaller components are required to accomplish that design objective. With smaller, lighter, and less components present on the breadboard, the entire module should weigh around a few ounces (no set number is established but the average user should not be overly affected by the module’s overall weight). Most importantly, Amplifier and Speaker Module uses common and mass-produced components to help lower cost. Rarer components require unique parts that are not commonly mass-produced which may cost more money to purchase. While the rare component’s size and weight could be checked, the parameters might not be a guaranteed amount. This fact is true with different companies as there is no standardized pinouts or structure.

Amplifier and Speaker Module’s limitation and performance issue is the speaker’s volume. This module is able to produce an audible sound wave and can be heard by the human ear. However, Amplifier and Speaker Module must be placed in a nearby area close to the user and it’s audio signal output cannot travel past a foot. Despite the additional LM 386 Power Amplifier, the audio range has improved but would not be able to travel far.

# **\* Technical Problem Solving**

## **8.1.1\* The ADC Sample Problem**

**Description:**

Ramiro identified this problem. The current microcontroller at the time (Arduino Uno) was not sampling the data correctly.

## **8.1.2\* Solving the ADC Sample Problem**

One of the requirements for DCAFF was to be able to take in a microphone signal ranging from 0 to 20k Hz. This means the nyquist rate would have to be at least 40kHz. Since CD quality audio was 44,100 Hz, the design requirement was set to 44,100 to match an industry standard. The issue with the sample rate from the Arduino Uno was simply just a hardware limitation. By looking at the Arduino Uno Datasheet, the maximum amount of samples that can be sampled was ~9kHz. This was significantly short of the 41khz sample rate needed. The next microcontroller tested was the ATMEGA 1284. However, by looking at the datasheet, the sample rate was only 38kHz even when overclocked. In the end, Ramiro and Henry chose the Teensy 3.2 as it could handle well over 44.1k sample rate and have 16 bit ADC and 16 bit DAC compatibility.

## **8.2.1\* The Serial Communication Problem**

**Description:**

Ramiro identified an issue with sending the audio data to Python. The audio data wasn’t playing properly. By comparing the raw audio file from Python and the SD card raw audio file, the unicode characters didn’t match exactly. This will cause a significant issue in formatting the raw data to a proper audible signal. In future versions, this also became an issue when writing back to the SD card from Python.

## **8.2.2\* Solving the Serial Communication Problem**

In order to tackle this issue, there was a significant problem to be tackled. Since Teensy is using Serial communication, only 8 bits can be sent at one time. This causes a huge issue since our samples are 16 bits long. One alternative found was to emulate the Teensy as an SD card. This way, Teensy is able to use MTP(media transfer protocol). This way, the Teensy would be able to transfer the files with ease. However, emulating the Teensy as an SD card would not allow the Teensy to record data or any other functionality. As a result, we were forced to use serial communication as bluetooth would end up with the same error anyways. However, Ramiro realized by making sure every single bit is transferred correctly, the 16 bit ADC can be ignored. This is because by sending every single bit into Python correctly, although Python will interpret the bits using Unicode, **each bit is still the same.**  Overall, Ramiro solved this issue by transferring the entire raw audio file bit by bit and recreating the file on Python. Next, was the issue of serial communication from Python to Teensy.

In order to understand this problem, it is important to know how Python interprets hex values. In Python, hex values are not a data type that can be used. Python artificially creates hex values by turning the string representation of a hex number into an integer. As a result, the audio file sent back to Teensy is a big string representation of hex. This is an issue as Python will serially send the hex string “0x0032” as “32”. This is because before the hex values are sent, Python will convert the string into an integer. This results with the hex representation of 0 being removed entirely. Henry was able to fix this issue by implementing zero padding the front of the hex values and making sure the hex representation was the same after serially sending the raw audio back.

## **8.3.1\* The Audio Formatting Problem**

**Description:**

Henry identified this problem. The problem was the audio file was not being formatted correctly. This created a near inaudible playback with noise.

## **8.3.2\* Solving the Audio Formatting Problem**

After successfully sending every bit into Python, the data was converted into hex, but the audio could not be played at all. After doing some research on standard raw files. The 16 bit PCM raw audio file required an internal ordering of bits. Thus, little endian had to be implemented to correctly order the hex numbers. After multiple times recording Henry noticed that the signal was barely audible in lower frequencies.To do a quick check, since the total number of samples divided by the sample rate (44.1kHz) the real time playback can be calculated. As a result, the real playback time was not sectioned into 16 bit samples correctly. Henry was able to identify this as the calculations yielded a playback time that was twice as fast as the recorded audio. This resulted in 32 bit samples.After correctly partitioning the data into 16 bits, the audio became more audible, but a lot of noise was still present. In order to be correctly played as an audio file, the audio must once again be formatted to fit a discrete LTI system (ranging from negative one to one)[1]. Finally, after fitting the audio into an LTI system, the playback was perfectly reproduced on Python.

## **8.4.1\* The Concurrent State Machine Problem**

**Description:**

Henry identified this problem. The problem was having 2 infinite loops (concurrent asynchronous state machines) interact indefinitely. huge issue since the Python code to serially communicate with the Teensy would not run.

## **8.4.2\* Solving the Concurrent State Machine Problem**

In order to tackle this problem, Ramiro and Henry decided to use knowledge from the embedded systems class. The main design choice in getting Python and Teensy to work together asynchronously was to create 2 state machines that continuously waited for each other. In order to do this, a “mode” was introduced to both Teensy and Python to prevent accidental switching ( for example, going from recording to playback due to misreading the serial bus). This way, the functions in Python will only run in response to changes in modes rather than iterating through a loop that executes every function each loop. This also saved on computing as the heavy filtering processes didn’t have to execute every loop.

## **8.5.1\* The Filter Design Problem**

**Description:**

Ramiro Identified this problem. The designed bandpass filter wasn’t producing the correct result.This is a huge issue since the core of the project was to be able to use 3 different filters(lowpass, high pass, and bandpass).

## **8.5.2\* Solving the Filter Design Problem**

In the running tests of the filters, there was a significant amount of noise being produced. While looking at the code, the sample frequency (omega\_s) wasn’t correctly implemented. As a result, when the cutoff frequency divided the sample frequency was over one, the Python code would either crash or create a sample greater than 1 , thus violating the LTI system(outside of -1 to 1 range)[1]. As a result, the noise was coming from the causes where the samples were greater than 1. To fix this, the sample frequency was matched to 44.1kHz and limited the cutoff frequency to less than 44.1khz to prevent noise.

## **8.6.1\* The Mainloop() Problem**

**Description:**

Henry identified this problem. Tkinter’s GUI only works as an uninterrupted while loop. This is a huge issue since the Python code to serially communicate with the Teensy would not run. As a consequence to the *Mainloop()* problem,the Python code that had all the filtering and data conversion could not be accessed due to the GUI’s while loop. This is a big issue since the GUI could not access the filters and audio data.

## **8.6.2\* Solving the Mainloop() Problem**

As a result of implementing a GUI, the nature of the Tkinter library does not allow background processes to run. What this means is the state machine wrote to interact with the Teensy would not work as it will never be called. In order to fix this, the GUI had to be able to break out of itself to do data processing and data acquisition. One option was to use jump commands and force the GUI to end and jump back into the code. As a result, the GUI would constantly crash everytime Python jumped out of the loop. Another method was to do multi threading and run both state machines at the same time. However, given the time constraints Henry and Ramiro decided to work around what Tkinter could do. As it turns out, Tkinter does have a built-in function called after() which will execute a function every few seconds. This way multithreading could be avoided and the main data processing code could be integrated into the GUI. However, this fix would require a way to access the data processing variables.

The way the data processing variables were solved was to declare global variables. Given only 2 weeks left, the best way to make sure the signal data and real time interaction was working was to use global variables inside the GUI code to remotely call in variables. This choice was made by Henry and Ramiro as the processing power wasn’t a huge impact and created a quick fix.

## **8.7.1\* The Speaker Audio Output Problem**

**Description:**

Michael identified an issue with the speaker in the second Class-D Amplifier design. The speaker’s sound is too quiet (dB). The oscilloscope displayed a low voltage peak to peak (0 to 3 volts) and needed to be raised.

## **8.7.2\* Solving the Speaker Audio Output Problem**

To fix this problem, Jeffrey added an additional LM 386 Power Amplifier IC to boost the incoming voltage to raise up the low voltage peak to peak into a higher one (1 to 5 volts). The speaker is able to play louder now.

# **User Interface Design**

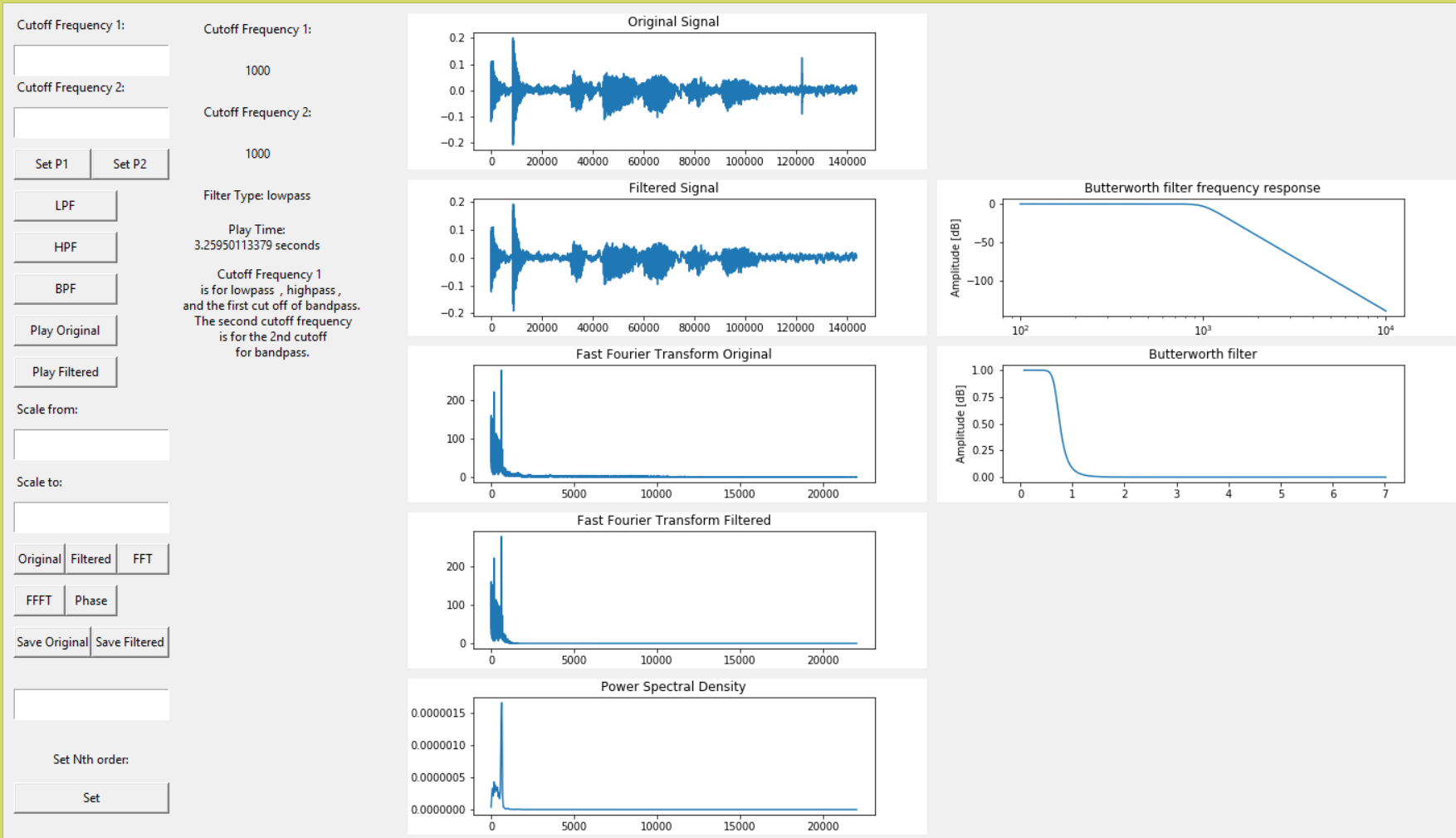
## **Application Control**

**GUI Features and Control:**

* The user can set the desired frequency by inputting values on the “Frequency 1:” and “Frequency 2:” input boxes and pressing the “Set F1” and “Set F2”
* The user can set which filter (low, high, or bandpass) to use by pressing the “LPF,” “HPF,” and “BPF” buttons
* The user can play the original and filtered recording by pressing the “Play Original” or “Play Filtered” buttons
* The user can zoom on the original signal, filtered signal, fft original, fft filtered, or the power density graph by input values into the “Set From:” and “Set To:” input boxes and pressing “Original,” “Filtered,” “FFT Original,” “FFT Filtered,” or “PSD” buttons.
* The user can save the original or filtered audio locally by pressing the “Save Original,” or “Save Filtered” buttons
* The user can set the order of the butterworth filter by inputting the order (0-22 recommended) into “Set order” box, and pressing the “Set” button.
* The 7 graphs correspond to the original signal, the filtered signal, the fft of the original signal, the fft of the filtered signal, the power spectrum density of the filtered signal, the Butterworth frequency response used, and the butterworth filter itself
* The length of time of the audio signal is also displayed

## **User Interface Screens**

The Figure 9.2 below shows how the user can interface with the Python and Teensy code:



*Figure 9.2:GUI Diagram*

**GUI framework:**

Henry designed the layout for the GUI (where to place graphs,texts, and buttons)

**Graphs:**

Henry was responsible for updating the graphs.

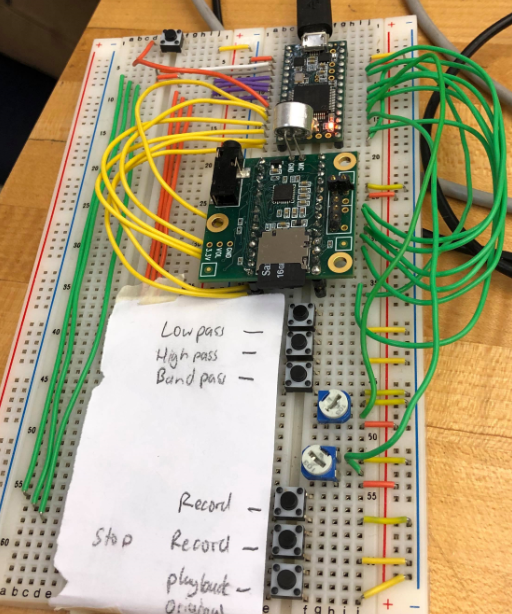
**Buttons:**

Ramiro was responsible for button implementation and text entry implementation.

Ramiro was responsible for scaling functions in graphs (allowed users to zoom into smaller frequencies).

## **User Interface Breadboard**

The following Figure 9.3 below is a picture of the Breadboard with the corresponding and potentiometers the user can use.



*Figure 9.3: User Interface Breadboard Layout*

* The user can set the low, high, or bandpass by pressing the corresponding buttons displayed above.
* The user can set the 1st and 2nd frequency of the filter by twisting the potentiometers. (top-1st frequency used for high, low, and bandpass; bottom-2nd frequency used for bandpass)
* The user can record and stop recording by pressing the “record,” and “stop recording,” buttons shown above
* The user can chose to play the original audio by pressing the “playback original” button
* The user can chose to play the filtered audio by pressing the top left button

# **\* Test Plan**

## **\* Test Design**

The following tests were conducted in order to test the full functionality of DCAFF, and to make sure the product works as intended.

**Test One:**

**Output Original and Filtered Signal**

Objective: The following tests were constructed to make sure the output audio was indeed the expected audio output in terms of frequency.

***Section 1:***

**Output for Original and Filtered Signals and their corresponding graphs**

Experiment Setup: In order to verify that the output signals were indeed the same signals that were being displayed on the GUI, we filtered a signal once and recorded all the graphs on the GUI. The filtered and original signal were then played, recorded, and ran through the filters a second time.

Experiment Data: The graphs and FFT will be printed on to the GUI.

Experiment Results: The graphs of the first iteration and the second iteration should look more or less the same (accounting for noise interference). The FFT should show that the signals were the same.

Analysis/Results: The test proved to be successful. The first iteration showed graphs that were very similar to second iteration graphs. More concisely, the FFT showed that both signals were indeed the same.

***Section 2:***

**Original Input vs. Original Output**

Experiment Setup: In order to test that the original signal picked up by the microphone was indeed the same signal outputted by the Teensy, we used an oscilloscope to measure the frequency of the input and output.

Experiment Data: The graph produced by the oscilloscope and the GUI will be collected and compared to one another.

Experiment Results: Both the frequencies and of the input and output should look the same

Analysis/Results: The test indeed showed the frequency from the output and input are the same.

**Test Two:**

**Button Presses and Potentiometer Readings**

Objective: The following tests were designed to prove that the buttons presses and potentiometer readings on the breadboard are indeed doing their corresponding tasks

***Section One:***

**Button presses for High, Low, and Bandpass (Bread Board)**

Experiment Setup: This test will indicate if the correct filter is applied to a signal when the corresponding buttons for high, low, or bandpass buttons on the breadboard are pressed

Experiment Data: for each time the button is pressed, the output of the filter collected will be recorded

Expected Results: Each button press should update the corresponding filter

Analysis/Results: Whenever the low, high, and bandpass buttons are pressed, the signal is appropriately filtered

***Section Two:***

**Button presses for Record/Stop Recording:**

Experiment Setup: The record and stop recording button will be pressed and the corresponding Teensy will record and stop recording.

Experiment Data: whenever a button is pressed, whether or not the Teensy records or doesn’t should be recorded

Expected Results: Whenever the Record and Stop Recording are pressed, the Teensy should start recording and stop recording audio

Analysis/Results: The test proved to be successful. Audio was recording whenever the record button is pressed, and it stopped recording whenever the stop recording button was pressed

***Section Three:***

**Potentiometer Values**

Experiment Setup: Whenever the potentiometers are set on the breadboard, the frequency values displayed on the Python window should be the same frequencies used on the filter

Experiment Data: The values displayed on the GUI will be recorded, and compared to the frequency set on the filter.

Expected Results: The value printed on the GUI should be the same value as the frequency used on the filter.

Analysis/Results: The test proved to be successful. The values frequencies displayed on the Python window, where the same frequencies used on the filter. We were able to verify this by looking at the FFT after the signal was filtered. The returned frequencies on the fft were within the bounds of the filter set frequencies

***Section Four:***

**Button presses for Play Original and Play Filtered Signal Buttons**

Experiment Setup: Whenever the play original or play filtered signal buttons are pressed, the Original and the filtered signal should be outputted to a speaker

Experiment Data: Whenever a button is pressed, whether or not the output audio is heard will be recorded

Expected Results: Whenever a button is pressed the output signal either be the original or filtered signal

Analysis/Results: The test proved to be successful. Whenever play original and filtered signal buttons were pressed, the corresponding audio was outputted

**Test Three:**

**Button Presses and Input Boxes (GUI)**

Description: The following tests were designed to make sure that all the buttons and input boxes on the GUI are working as intended

***Section 1:***

**Button Presses for Low, High, and Bandpass Filters**

Experiment Setup: The “LPH,” “HPF,” or “BPF” will be pressed numerous times.

Experiment Data: Whenever the buttons above are pressed, whether or not the filters are updated will be recorded.

Expected Result: Whenever the “LHP,” “HPF,” or “BPF” buttons are pressed, the corresponding filter should be applied to audio signal

Analysis/Results: The test proved to be successful. Whenever buttons were pressed, the original audio was filtered with its corresponding filter.

***Section 2:***

**Button Presses for Play Original and Play Filtered**

Description: Whenever the “Play Original” or “Play filtered” buttons are pressed, the original signal or the filtered signal should be outputted via PC speakers

Analysis/Results: The test proved to be successful. Whenever the buttons were pressed, the original or filtered signal was outputted from the PC speakers

***Section 3:***

**Button Presses for Save Original and Save Filtered**

Description: Whenever the “Save Original” and “Save Filtered” buttons are pressed. The original and filtered signal should be saved locally on the PC.

Analysis/Results: We were able to verify that the original and filtered signal saved locally were in fact the corresponding files by comparing the context of the file to that of that context of the original and filtered audio.

***Section 4:***

**Frequency 1 and Frequency 2 Input Boxes**

Experiment Setup: the “Frequency 1” and “Frequency 2” buttons will be pressed many times.

Experiment Data: Whether or not the frequencies are updated will be recorded.

Experiment Results: Whenever a frequency is entered on to the “Frequency 1” and “Frequency 2” input boxes, the corresponding frequencies should be the actual values that the filters used

Analysis/Results: The test proved to be successful. By comparing the the fft of the original and filtered signal, we were able to determine that the frequencies inputted were indeed correct.

***Section 5:***

**Setting the nth order Butterworth Filter Input Box**

Experiment Setup: The order of the butterworth filter will be set using the GUI.

Expected Data: In order to verify that the order set by the user is indeed correct, the poles and zeros of the corresponding butterworth filter were printed out to console and printed to the console.

Experiment Setup: The number of poles and zeros were then compared to the theoretical value of how many poles and zeros the butterworth filter should have

Analysis/Results: The test proved to be successful, where the order set by user matched the expected values and of poles and zeros

***Section 6:***

**Setting range values for Scaled from and Scaled To**

Experiment Setup: The values of the set range box will be inputed and processed.

Experiment Data: In order to verify that the range inputted for each corresponding graph is correct, we compared the graph produced with the corresponding discrete signal saved onto the raw file.

Expected Result: The discrete signal should look like the graph produced

Analysis/Results: The test proved successful. The two graphs looked exactly the same.

**Test Four:**

**Class D Amplifier and Speaker**

The following tests were designed to make sure that the Class D Amplifier and the Speaker are working as intended.

***Section 1:***

**Checking power efficiency**

**Description:** A class D amplifier has at least 90% power efficiency. Power needed to be tested at the beginning of the circuit and at the end of the circuit. In order to find power, current and voltage values were needed at those specific areas. Then calculate the power efficiency by dividing output power by input power.

***Section 2:***  
**Total Harmonic Distortion(THD)**

**Description:** Use a 1kHz sine wave as the audio input. Use an oscilloscope probe to probe the output. Change the oscilloscope to produce FFT of the output. Measure the voltages of each harmonic. Then calculate the THD. The THD of a class D amplifier should be below 10%.

***Section 3:***

**Phase Shift**

**Description:** In order to verify the Amplifier’s audio delay, we use the oscilloscope and probes for the audio input and output connections. The oscilloscope should read two waves with a phase shift of around 180 degrees.

Analysis/Results: The test proved successful. The oscilloscope reads a phase shift of around 190 degrees, indicating that the output signal is released a quarter after the peak of the original input signal.

## **\* Bug Tracking**

Potentiometer Value set too fast Python crashes

-Assigned to Henry to investigate

FFT approximations weren’t the correct frequencies

-Assigned to Ramiro to investigate

Audio was played back too fast (twice the speed)

-Assigned to Henry to investigate

Cutoff frequency wasn’t updating on filters when set on GUI

-Assigned to Henry to investigate

Updates to the GUI placed information in the wrong position

-Assigned to Ramiro to investigate

Python to Arduino Serial communication cutting data

-Assigned to Ramiro to investigate

Microphone wasn’t working sometimes

-Assigned to Henry to investigate

Bandpass filter creating NaN values

-Assigned to Henry to investigate

Audio Signal occasionally fails to connect through to TPA 3122 IC

-Assigned to Jeffrey to fix

Phase Shift has noise at a specific range at the audio input and output

-Assigned to Michael to investigate

## **\* Quality Control**

The following tests were done in order to insure that our final product was working as intended. Each button both on the GUI and on the breadboard were tested individually to ensure the quality of the product.

**Test One:**

**Output Original and Filtered Signal**

**Test conducted by: Henry and Ramiro**

The graphs outputted by this test were saved to a corresponding folder for each test case. The graphs were then compared to one another in order to verify their accuracy.

**1.1**

The result: FAILED 1/18/20

-Graphs are concatenating from repeated inputs

The result: PASSED 1/25/20

-Graphs are updating correctly

**1.2**

The result: PASSED 1/18/20

-Oscilloscope and first graph are ~95% similar

**Test Two:**

**Button Presses and Potentiometer Readings**

**Test conducted by: Henry and Ramiro**

The tests conducted for each button press were recorded on an excel sheet and whether or not each filter was updated appropriately. Each test case was then analyzed to see if the expected result was achieved. The potentiometer readings were recorded in a similar case, where each reading of the potentiometer was saved and compared to the set filter frequency.

**2.1**

The result: PASSED 1/17/20

-Buttons work as intended, updates real time.

**2.2**

The result:PASSED 1/15/20

-Recording and Stop Recording working as intended

**2.3**

The result: PASSED 1/23/20

-Potentiometer values update real time

**2.4**

The result: PASSED 1/23/20

-Playback works for original signal and filtered signal on Teensy.

**Test Three:**

**Button Presses and Input Boxes (GUI)**

**Test conducted by: Henry and Ramiro**

For every button press, whether or not the button updated the corresponding filter was saved in an excel sheet, and analyzed to make sure each button was working as intended. The frequency set by the potentiometer was also recorded and compared the frequency set for the filter. The poles used for the nth order butterworth filter were also saved when the order was set and compared to the theoretical values for the poles the order of the butterworth filter should be used. As the range for each graph was updated, we saved the picture of each graph and took a picture of the actual signal being graphed, and analyzed each set of graphs to make sure they were the same.

**3.1**

The result: FAILED 2/9/20

-Buttons aren’t updating correctly.

The result: PASSED 2/18/20

-Buttons are working correctly.

**3.2**

The result:PASSED 2/20/20

-Playback for filtered and original are working as intended.

**3.3**

The result: PASSED 2/20/20

-File is saved onto the desktop correctly.

**3.4**

The result: FAILED2/20/20

-Cutoff frequencies not setting correctly.

The result: PASSED 2/28/20

-Cutoff frequencies set correctly.

**3.5**

The result: PASSED /23/20

-Nth order set correctly.

**3.6**

The result: PASSED /23/20

-Graphs for original signal, filtered signal, FFT ,and filtered FFT scale correctly.

**Test Four:**

**Class D Amplifier and Speaker**

**Tests conducted by: Michael and Jeffrey**

4.1

The result: PASSED 2/23/20

-The oscilloscope reads an efficiency of over 90 percent.

4.2

The result: PASSED 2/23/20

-The oscilloscope reads a harmonic distortion of less than five percent of the total signal.

4.3

The result: FAILED 2/23/20

-The oscilloscope reads a phase shift of over 300 degrees.

The result: PASSED 2/27/20

-The oscilloscope reads a phase shift of around 30 degrees.

## **\* Identification of critical components**

**Teensy 3.2**

* Holds all the raw audio data
* Main way to communicate between Python ,SD card, ADC, DAC, and microphone

**SD card**

* Holds all audio raw data for playback
* Holds all filtered raw data for playback

**Python Code**

* All data processing is done here.
* Allows the Teensy buttons and potentiometer to directly communicate with GUI

**GUI**

* Allows visuals for user
* Contains buttons and text inputs for the user to customize filters

**TPA 3122 Amplifier IC**

* Main component that runs the Amplifier and Speaker system

**LM 386 Power Amplifier IC**

* Allows the speaker to reach louder volumes

**4 Ohm, 2 Watt Speaker**

* Used for outputting audio

## **\* Items Not Tested by the Experiments**

**SD Card file size**

The advertised 16gb storage capability was not tested for actual size

**Recording time**

The maximum length of recording time was never measured

# **\* Test Report**

## **\* Test: Signal Processing and GUI (Ramiro Carrillo and Henry Chen)**

**This section shows an overall test of Teensy , GUI, and Python. These tests will show the main features of the final prototype working correctly.**

**The 3 tests will consist of:**

**Test One: Data Acquisition**

* **Output Original and Filtered Signal**
* **Original Input vs Output Signal**

**Test Two: Teensy Functionality**

* **Button presses**
* **Potentiometer**
* **Recording and Stop Recording**
* **Playback**

**Test Three: GUI**

* **Button presses**
* **User Input Cutoff Frequency**
* **Graphs**
* **User Input Nth order**
* **User input graph scaling**
* **Playback**
* **Save Original and Filtered Audio**

**Test Dates:**

**Each test will run through the 3 tests above.**

1. **Total Test 1 (2/24/20)**
2. **Total Test 2 (2/28/20)**
3. **Total Test 3 (3/01/20)**
4. **Total Test 4 ( 3/02/20)**

## **\* Test 1: Data Acquisition**

The following tests were constructed to make sure the output audio was indeed the expected audio output in terms of frequency.

***Section 1:***

**Output for Original and Filtered Signals and their corresponding graphs**

The Following test was constructed in order to verify that the output audio (original and Filtered) were indeed the same signals that correspond to the graphs displayed on the GUI.

Description: In order to verify that the output signals were indeed the same signals that were being displayed on the GUI, we filtered a signal once and recorded all the graphs on the GUI. The filtered and original signal were then played, recorded, and ran through the filters a second time. The graphs of the first iteration and the second iteration should look more or less the same (accounting for noise interference). The results should be 95% accurate considering noise.

*Tester and Analyzer* : Henry Chen

**Analysis/Results:**

1. The first plot of graphs for the original signal,filtered signal, original FFT, and filtered FFT were correct.
2. The second run of plotting ran into errors. The original signal,filtered signal, original FFT, and filtered FFT were “stretching” as if the previous signals were concatenating themselves.
3. The FFT and filtered FFT bins were not clearing themselves and kept old inputs while binning in new frequencies.

**The result of this section is a FAIL, must fix the graphs to only display the most recent audio signal, not an accumulation of results. Must fix plotting and data acquisition.**

***Section 2:***

**Original Input vs. Original Output**

Description: In order to test that the original signal picked up by the microphone was indeed the same signal outputted by the Teensy, we used an oscilloscope to measure the frequency of the input and output. The frequencies should be more or less the same.

*Tester and Analyzer* : Ramiro Carrillo

**Analysis/Results:**

1. In order to do this test, a tonal generator was used to generate strikingly similar sine waves.
2. The scaled graphs and a snapshot of the oscilloscope yielded the same results.

**The result of this section is a PASS.**

**11.2.2 \* Test 1: Data Acquisition (Repeat:1)**

***Section 1:***

**Output for Original and Filtered Signals and their corresponding graphs**

*Tester and Analyzer* : Henry Chen

**Analysis/Results:**

1. The result should be the most recently filtered graph.
2. The graph is now properly displaying the graphs, the reason the graphs were concatenating was the figures and plots had to be individually cleared (inside the widget and matplotlib figure).

**The result of this section is a PASS**

***Section 2:***

**Original Input vs. Original Output**

*Tester and Analyzer* : Ramiro Carrillo

**The result of this section is a PASS.(Verification:1)**

**11.2.3 \* Test 1: Data Acquisition (Repeat:2)**

***Section 1:***

**Output for Original and Filtered Signals and their corresponding graphs**

*Tester and Analyzer* : Henry Chen

**The result of this section is a PASS.(Verification:1)**

***Section 2:***

**Original Input vs. Original Output**

*Tester and Analyzer* : Ramiro Carrillo

**The result of this section is a PASS.(Verification:2)**

**11.2.4 \* Test 1: Data Acquisition (Repeat:3)**

***Section 1:***

**Output for Original and Filtered Signals and their corresponding graphs**

*Tester and Analyzer* : Henry Chen

**The result of this section is a PASS.(Verification:2)**

***Section 2:***

**Original Input vs. Original Output**

*Tester and Analyzer* : Ramiro Carrillo

**The result of this section is a PASS.(Verification:3)**

## **\* Test 2: Teensy Functionality**

The following tests were constructed to make sure Teensy’s concurrent state machine is working as intended.

***Section 1:***

**Button presses**

Description: In order to make sure the correct inputs were being inputted from the Teensy to the Python code the Teensy must be checked. The test will consist of pressing the lowpass, highpass, and bandpass button on the Teensy to make sure the type of filter is properly updated on the GUI

*Tester and Analyzer* : Henry Chen

**Analysis/Results:**

1. When the 3 buttons (lowpass, highpass, and bandpass) are pressed on the Teensy the serial communication sends a character to the Python code to indicate the mode
2. The Python code read the character reads the character and sets a mode,
3. Inside the mode the filter type is set.

**The results of this section of the test is PASS.**

***Section 2:***

**Potentiometer**

Description: In order to make sure the correct inputs were being inputted from the Teensy to the Python code the Teensy must be checked. The test will consist of turning the potentiometer in many different ways to ensure the value is updated correctly on the GUI.

*Tester and Analyzer* : Ramiro Carrillo

**Analysis/Results:**

1. The potentiometer is changing as planned. The cutoff frequency is being updated on the GUI correctly.
2. A sporadic change in the potentiometer causes a crash. (Turning 180 degrees on the potentiometer in .5 seconds).
3. Need to check hardware connections to ensure wired properly.

**The results of this section of the test is FAIL.**

***Section 3:***

**Recording and Stop recording**

Description: In order to make sure the correct inputs were being inputted from the Teensy to the Python code the Teensy must be checked. The test will consist of recording and audio signal and filtering the signal. The Python code will send back the audio data serially. User will be able to analyze the graphs on the GUI

*Tester and Analyzer* : Henry Chen

**Analysis/Results:**

1. The audio is properly sent to the Python code
2. The graphs analyzed are correctly displaying the data.

**The results of this section of the test is PASS.**

***Section 4:***

**Playback**

Description: In order to make sure the correct inputs were being inputted from the Teensy to the Python code the Teensy must be checked. The test will consist of recording and audio signal and filtering the signal. The Python code will send back the audio data serially. User will press playback original and playback filtered to play audio again.

*Tester and Analyzer* : Henry Chen

**Analysis/Results:**

1. The audio signal is properly being sent into the Python code and processed.
2. At this point only the original signal is on the SD card to be played back. Playback on the original works.
3. The filtered data is sent back. Filtered playback button works as intended.

**The results of this section of the test is PASS.**

**11.3.2 \* Test 2: Teensy Functionality(Repeat : 1)**

***Section 1:***

**Button presses**

*Tester and Analyzer* : Henry Chen

**The results of this section of the test is PASS.(Verification:1)**

***Section 2:***

**Potentiometer**

*Tester and Analyzer* : Ramiro Carrillo

**Analysis/Results:**

1. The potentiometer should be able to consistently change and update cutoff frequencies.
2. Potentiometer is crashing each run from iterating over the limit of 40,000 Hz.

**The results of this section of the test is FAIL. This may be an hardware issue, new potentiometer will be inputted.**

***Section 3:***

**Recording and Stop recording**

*Tester and Analyzer* : Henry Chen

**The results of this section of the test is PASS.(Verification:1)**

***Section 4:***

**Playback**

*Tester and Analyzer* : Henry Chen

**The results of this section of the test is PASS.(Verification:1)**

**11.3.3 \* Test 2: Teensy Functionality(Repeat : 2)**

***Section 1:***

**Button presses**

*Tester and Analyzer* : Henry Chen

**The results of this section of the test is PASS.(Verification:2)**

***Section 2:***

**Potentiometer**

*Tester and Analyzer* : Ramiro Carrillo

**Analysis/Results:**

1. The potentiometer should be able to consistently change and update cutoff frequencies.
2. With the new potentiometer the cutoff frequencies are properly set and dont crash when moved.

**The results of this section of the test is PASS.**

***Section 3:***

**Recording and Stop recording**

*Tester and Analyzer* : Henry Chen

**The results of this section of the test is PASS.(Verification:2)**

***Section 4:***

**Playback**

*Tester and Analyzer* : Henry Chen

**The results of this section of the test is PASS.(Verification:2)**

**11.3.4 \* Test 2: Teensy Functionality(Repeat : 3)**

***Section 1:***

**Button presses**

*Tester and Analyzer* : Henry Chen

**The results of this section of the test is PASS.(Verification:3)**

***Section 2:***

**Potentiometer**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS.(Verification:1)**

***Section 3:***

**Recording and Stop recording**

*Tester and Analyzer* : Henry Chen

**The results of this section of the test is PASS.(Verification:3)**

***Section 4:***

**Playback**

*Tester and Analyzer* : Henry Chen

**The results of this section of the test is PASS.(Verification:3)**

* 1. \* **Test 3: GUI**

The following tests were constructed to make sure the GUI’s functionality is working as intended.

***Section 1:***

**Button Presses**

Description: There are several buttons on the GUI. The buttons tested will be the lowpass, highpass, and bandpass.

*Tester and Analyzer* : Henry Chen

**Analysis/Results:**

1. The test is done simply by pressing each button and recording audio.
2. The test is verified by looking at the filter graph.
3. The lowpass filter is showing the correct graph.
4. The highpass filter is showing the correct graph.
5. The bandpass filter is showing the correct graph.

**The results of this section of the test is PASS.**

***Section 2:***

**User Input Cutoff Frequency**

Description: The user is able to set the cutoff frequency for the lowpass,highpass, and bandpass filter. The inputted cutoff frequencies should change the characteristics of the filters to match the user’s input. The first cutoff frequency should change the lowpass , highpass, and the first cutoff of the bandpass filter. The second cutoff frequency sets the second cutoff for the bandpass filter.

*Tester and Analyzer* : Ramiro Carrillo

**Analysis/Results:**

1. The lowpass filter and highpass filter is working as intended.
2. The bandpass filter is having difficulty working. The filter would work half the time. More work needs to be done on fixing the cutoff frequency inputs.

**The results of this section of the test is FAIL.**

***Section 3:***

**Graphs**

Description: The graphs should be able to properly display the original signal, filtered signal, FFT, and Filtered FFT. The graphs should be accurate and notable changes can be seen from comparing the original and filtered results.

*Tester and Analyzer* : Henry Chen

**Analysis/Results:**

1. For the original signal and filtered signal, the results can be seen by looking at the amplitude of the signal.
2. The FFT and filtered FFT seem to be off. Testing by playing a 500hz signal bins the FFT slightly in the range of ~550hz. Needs to be fixed

**The results of this section of the test is FAIL.**

***Section 4:***

**User Input Nth order**

Description: The user is able to input any order into the GUI. The GUI will tell the Python code to update the order of the Butterworth filter. This will be tested by looking at multiple inputs. Higher orders (22nd order) should have a visible steep cutoff frequency. On the other hand, small orders (2nd order) should see a significantly less steep connection from the stopband to the passband.

*Tester and Analyzer* : Ramiro Carrillo

**Analysis/Results:**

1. The default 7th order filter is quite steep. By inputting a 22nd order the filter’s passband and stopband become steeper.
2. By inputting 3rd order the passband and stop band frequencies become further apart.

**The results of this section of the test is PASS.**

***Section 5:***

**User Input Graph Scaling**

Description: The user is able to input 2 numbers to zoom into the graphs of the original signa, filtered signal, FFT signal, and Filtered FFT signal.

*Tester and Analyzer* : Ramiro Carrillo

**Analysis/Results:**

1. An audio signal is recorded. Scale for the original signal graph works as intended.
2. An audio signal is recorded. Scale for the filtered signal graph works as intended.
3. A sinusoidal signal at 500 Hz is recorded The FFT graph is scaled from 0 to 1000 to allow the user to see the most binned samples at 500Hz.

**The results of this section of the test is PASS.**

***Section 6:***

**Playback**

Description: By pressing “Playback Original” the recorded audio will play the audio locally on the PC.By pressing “Playback Filtered” the recorded audio will play the audio locally on the PC.

*Tester and Analyzer* : Ramiro Carrillo

**Analysis/Results:**

1. The original audio plays locally from PC as intended.
2. The filtered audio plays locally from PC as intended.

**The results of this section of the test is PASS.**

***Section 7:***

**Save Original and Filtered Audio**

Description: By pressing “Save Original” or “Saved Filtered” the respected audio file is saved onto the desktop as an audio file.

*Tester and Analyzer* : Henry Chen

**Analysis/Results:**

1. The audio is saved onto the desktop correctly.
2. To test the playback, using windows media player and Audacity, the audio is as recorded from the microphone.

**The results of this section of the test is PASS.**

**11.4.2 \* Test 2: GUI(Repeat : 1)**

***Section 1:***

**Button Presses**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS. (Verification:1)**

***Section 2:***

**User Input Cutoff Frequency**

*Tester and Analyzer* : Henry Chen

**Analysis/Results:**

1. The lowpass, highpass, and bandpass should consistently update cutoff frequencies inputted from the user.
2. The lowpass and highpass are working fine.
3. The bandpass cutoff frequencies are set properly using the potentiometers,but the GUI input isn’t correctly updating.

**The results of this section of the test is FAIL. (Debug GUI code to fix bandpass cutoff frequencies)**

***Section 3:***

**Graphs**

*Tester and Analyzer* : Ramiro Carrillo

**Analysis/Results:**

1. For the original signal and filtered signal, the results can be seen by looking at the amplitude of the signal.
2. The FFT and Filtered FFT is approximated properly, this was fixed by setting a reference sample(in this case 44,100 Hz) in order for the program to limit the bins in the FFT.

**The results of this section of the test is PASS.**

***Section 4:***

**User Input Nth order**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS. (Verification:1)**

***Section 5:***

**User Input Graph Scaling**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS. (Verification:1)**

***Section 6:***

**Playback**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS. (Verification:1)**

***Section 7:***

**Save Original and Filtered Audio**

*Tester and Analyzer* : Henry Chen

**The results of this section of the test is PASS. (Verification:1)**

**11.4.3 \* Test 2: GUI(Repeat : 2)**

***Section 1:***

**Button Presses**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS. (Verification:2)**

***Section 2:***

**User Input Cutoff Frequency**

*Tester and Analyzer* : Henry Chen

**Analysis/Results:**

1. The lowpass, highpass, and bandpass should consistently update cutoff frequencies inputted from the user.
2. The lowpass and highpass are working fine.
3. The GUI is finally able to properly update the cutoff frequencies. This was because of a variable assignment error.

**The results of this section of the test is PASS.**

***Section 3:***

**Graphs**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS.(Verification:1)**

***Section 4:***

**User Input Nth order**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS. (Verification:2)**

***Section 5:***

**User Input Graph Scaling**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS. (Verification:2)**

***Section 6:***

**Playback**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS. (Verification:2)**

***Section 7:***

**Save Original and Filtered Audio**

*Tester and Analyzer* : Henry Chen

**The results of this section of the test is PASS. (Verification:2)**

**11.4.4 \* Test 2: GUI(Repeat : 3)**

***Section 1:***

**Button Presses**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS. (Verification:3)**

***Section 2:***

**User Input Cutoff Frequency**

*Tester and Analyzer* : Henry Chen

**The results of this section of the test is PASS(Verification:1).**

***Section 3:***

**Graphs**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS.(Verification:2)**

***Section 4:***

**User Input Nth order**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS. (Verification:3)**

***Section 5:***

**User Input Graph Scaling**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS. (Verification:3)**

***Section 6:***

**Playback**

*Tester and Analyzer* : Ramiro Carrillo

**The results of this section of the test is PASS. (Verification:3)**

***Section 7:***

**Save Original and Filtered Audio**

*Tester and Analyzer* : Henry Chen

**The results of this section of the test is PASS. (Verification:3)**

## **11.5 \* Class-D Amplifier and Speaker**

**This section shows an overall test for the Amplifier and Speaker. These tests will show the main features of the final prototype working correctly.**

**The 3 tests will consist of:**

**Test One: Power Efficiency**

**Test Two: Total Harmonic Distortion**

**Test Dates:**

1. **Total Test 1 (2/23/20)**
2. **Total Test 2 (2/27/20)**

## **11.6.1 \* Power Efficiency**

***Section 1:***

**Power Efficiency**

The following test was constructed in order to verify that the Class-D Amplifier is indeed Class D by proving the amplifier’s unique characteristic: high efficiency.

Description: Find input power and output power via voltage and current. Calculate power efficiency through the equation: output power/input power \* 100%. The Class D Amplifier was tested with a signal generator and a smartphone on two separate tests.

*Tester and Analyzer* : Jeffrey Lee

**Analysis/Results:**

1. It was calculated that there was a power efficiency of around 95 percent. A small error percentile to the expected 96 percent.

**The result of this section is a PASS, TPA 3122 IC is indeed a Class D Amplifier.**

## **11.6.2 \* Power Efficiency(Repeat:1)**

***Section 1:***

**Power Efficiency**

*Tester and Analyzer* : Jeffrey Lee

**The results of this section of the test is PASS. (Verification: 1)**

## **11.7.1 \* Total Harmonic Distortion**

The Following test was constructed in order to determine the amount of audio distortion occurring in the Class D Amplifier.

Description: Use a 1kHz sine wave as the input. On the oscilloscope, change the display to show the FFT. Find the harmonics and calculate the total harmonic distortion.

*Tester and Analyzer* : Jeffrey Lee

**Analysis/Results:**

1. The oscilloscope displays one harmonic frequency which results in a total harmonic distortion of less than 5 percent.

**The result of this section is a PASS, TPA 3122 IC circuit has small audio distortion.**

## **11.7.2 \* Total Harmonic Distortion(Repeat:1)**

***Section 2:***

**Audio Distortion**

*Tester and Analyzer* : Jeffrey Lee

See Test 1

**The results of this section of the test is PASS. (Verification: 1)**

# **\* Conclusion and Future Work**

## **\* Conclusion**

The signal processing side of DCAFF works as intended. The GUI interface made for DCAFF can be repeatedly used without crashing. DCAFF is able to use various filters to convolve with the recorded audio. The FFT for the original signals are correctly calculated. The buttons and GUI can also be interacted with real time and does not crash as long as the user inputs the correct values. The only issue was the signal processing system could not be integrated well with the amplifier. For future work the amplifier could be fixed to properly have a large enough amplification to output any signal from the other modules.

**Ramiro:**

This project has taught me Python coding. It has enhanced my arduino programming. It has taught me about filter construction. Filtering in general. ADC and DAC conversion. Serial/SPI/I2C Communication. Audio formatting. Little-endian raw audio formatting, and how to extract a discrete signal from it. Gain produced from a butterworth filter, and how each frequency has a different gain. Taking the fft of a discrete signal. How to make a GUI and user interface. Simultaneous looping between software architectures. Sampling a continuous signal. How to properly implement tests and experiments.

**Henry:**

This project has taught me a lot about signal processing. This project allowed me to learn Python and implement some real life applications of audio processing. This project allowed me to learn more about audio formatting (converting raw PCM values into audible waves). I also learned a lot about how audio is formatted digitally. Furthermore, the importance of checking if your system is LTI. DCAFF has given me a deeper understanding of FFT and analyzing signals. DCAFF has also taught me a lot about implementing state machines. Teensy, PC ,and GUI proved to be a good challenge to learn how to run 3 asynchronous state machines in one. DCAFF has also taught me alot about the importance of having a backup plan in case something goes wrong. Systems Engineering taught me a lot about the way projects are supposed to run structurally.

**Michael**:

This project and especially people such as Gaalaass, Chirila, and Afrotechmods, teaches me about the concept, theory, and assembly of a Class D Amplifier. To a lesser extent, I also learned more about audio processing from my partners Ramiro and Henry and how DCAFF is a combination of embedded systems, audio processing, and digital audio filtering. Project DCAFF is the first project that I have done that uses Systems Engineering. The Systems Engineering process had multiple steps that added onto my learning experience in this project and I am left with a better idea of how to manage future projects professionally. Jeffrey and my Class D Amplifier succeeded in producing lower power loss as well as outputting audible sound from DCAFF’s other modules and from other electronics such as smartphones and personal computers. I succeed in developing a schematic, creating a circuit from the schematic, and troubleshooting component and system problems with the utmost safety. Jeffrey and I have failed to have the Amplifier and Speaker to output louder audio from the signal processing system. I suspect that the TPA 3122 IC and the speaker needs to be connected together differently. The speaker is connected to one channel of the TPA 3122 IC which only outputs around 10 watts per channel. If Jeffrey and I used the Mono Bridge Tied Load (where one speaker goes to both channels) then the speaker would be able to produce a louder audio signal.

**Jeffrey:**

This project has taught me about the theoretical and conceptual aspects of a class D amplifier. I gained knowledge on how to do some basic soldering and basic Python coding. If I were to continue to research and try these things out then these skills could help me in the future. I learned how to read multiple datasheets and find the information needed from them. I also learned how to research information when tests do not produce the desired results. Also I learned that I need to take charge sometimes and not always wait for someone to tell me what to do.

## **Future Work**

-Currently the user needs to install Python and a number of libraries in order for the product to work. There is a program called “pyinstaller” that makes any Python code into a stand alone executable program; no libraries or Python need to be installed. The user user would simply double click the executable, and the GUI would pop open. We were unable to get this to work due to time constraints.

-DCAFF was originally intended to filter live audio and output the audio as it is being recorded. The original and filtered audio graph, along with the FFT and the power density graphs were all supposed to update real time. Due to time constraints, we were unable to make this happen, and decided to process the audio data one step at a time: record audio, filter it, output it.

-While the original audio is transferred between the Teensy to pc and back from pc to Teensy, the code for the pc and the Teensy enter a line of code, which they don’t exit until the audio is completely transferred. While this does not affect performability, the user cannot interact with the GUI during this time. We would like to improve the code to where it keeps on looping while the audio is being transferred.

-Currently, the Teensy and Python need to be connected in order to work. We would like to make the Teensy able to record on its own. When the user wants to analyze the audio, then he would connect the Teensy to Python.

-We would also like to move the audio filters on to the Teensy. This would make it so the Teensy operates as a stand alone product, where it is able to filter audio more portably. And, if the user needs to analyze the filtered audio, he then would be able to connect it to Python.

-Currently, the nth order butterworth filter graph is updated after the audio is processed. We would like to have the nth order butterworth filter update as the frequency values are being set.

-We would like to increase the speed of audio transfer between Teensy and pc. This could be done by compressing the audio file being sent.

-We will change the amplifier and speaker’s power supply system into a battery and usb based power charging system to remove the large and expensive HP power supply system.

-We will expand the line-up of amplifiers with the inclusion, but not all, of Class A, Class B, Class AB, Class C, and Class T variants.

-We will include a seperate output audio jack to allow connections to external speakers. A button or switch will change the output between the in-built speaker and the external speaker.

## **\* Acknowledgement**

Prof. Roman Chomko, Department of Electrical Engineer

Prof. Hossny El-Sherif, Department of Electrical Engineer

Prof. Ertem Tuncel, Department of Electrical Engineer

Paul Stoffregen, Creator of Teensy

# **\* References**

[1]Oppenheim, A. V., Willsky, A. S., & Young, I. T. (1983). *Signals and systems.* Englewood Cliffs, N.J: Prentice-Hall.

[2]P. Stoffregen, *Advanced Microcontroller Audio Workshop.* March 2013*.*[Online]. Available: <https://cdn.hackaday.io/files/8292354764928/workshop.pdf>. [Accessed Mar. 15, 2020]

For Amplifier Design:

First Prototype Model:

[3]7HC04 Hex Inverter IC Datasheet: <https://assets.nexperia.com/documents/data-sheet/74HC_HCT04.pdf>

[4]E. Gaalaass. *Analog.com: Class D Audio Amplifiers: What, Why, and How.* July 2006. [Online]. Available: <https://www.analog.com/en/analog-dialogue/articles/class-d-audio-amplifiers.html#>. [Accessed Oct. 24, 2019].

[5]C. Chirila. *All About Circuits: How to Build a Class-D Power Amp*. August 2018. [Online]. Available: <https://www.allaboutcircuits.com/projects/how-to-build-a-class-d-power-amplifier/>. [Accessed Nov. 12, 2019].

[6]GreatScottLab. *Instructables: DIY Class D Audio Amplifier.* June 2017. [Online]. Available: <https://www.instructables.com/id/DIY-Class-D-Audio-Amplifier/>. [Accessed Nov. 13, 2019]

[7]IRLZ44N Mosfet Driver IC Datasheet: <http://www.irf.com/product-info/datasheets/data/irlz44n.pdf>

[8]LM 393-N Comparator IC Datasheet: <http://www.ti.com/lit/ds/symlink/lm393-n.pdf>

[9]LM 7805 Voltage Regulator IC Datasheet: <http://www.ti.com/lit/ds/symlink/lm340.pdf>

[10]LM7812 Voltage Regulator IC Datasheet: <https://datasheet.octopart.com/LM7812-Inchange-Semiconductor-datasheet-15981488.pdf>

[11]TLC 555 Timer IC Datasheet: <https://www.instructables.com/id/DIY-Class-D-Audio-Amplifier/>

Second Prototype Model:

[12]Afrotechmods. "Class D Amplifier Tutorial!" **YouTube**, June 1, 2014. [**Video File Uncertain**]. Available: <https://www.youtube.com/watch?v=O1UagNkcxi4> [Accessed: Feb 27, 2020].

[15]LM 386 Power Amplifier IC Datasheet: <http://www.ti.com/lit/an/sloa068a/sloa068a.pdf>

[16]TPA 3122 Stereo Power Amplifier IC Datasheet: <http://www.ti.com/lit/ds/symlink/tpa3122d2.pdf>

# **\* Appendices**

**Appendix A: Parts List**

Parts list:

Teensy 3.2

SGTL 5000 Audio Shield

2x105 Potentiometer

7xButtons

Electric microphone

16 gb sd card

Breadboard

Class D Amplifier Kit

* 0.1 ohm, 1 ohm, 10 ohm, 1000 ohm capacitors (ceramic and electrolytic)
* 33 uH Shield Inductors
* U4007 Diodes
* TPA3122, LM 386, LM393, TLC555, IRLZ44N, LM 7805, LM 7812, 7HC04, ICs

4 Ohm, 3 Watt Speaker

**Appendix B: Equipment List**

PC/Laptop

DC Power Supply

Oscilloscope

Signal Generator

**Appendix C: Software List**

Python 2.7

Arduino IDE

Audacity

Github

Fritzing

Draw.io

Text-compare

**Appendix D: Special Resources**

IEEE Solarium

IEEE Lab Rooms

**Appendix E: User's Manual**

To Implement Code: <https://github.com/henryc386/SeniorDesignVersions/tree/master/175prototypeworking_5.0>

Upload Arduino Code and run Python script