Project SENSE: An Enhanced Hearing Device

ME135 Final Project Report

Team SENSE:
Andrew Moncada   Hubert Liu   Andrew Zhu   Liam McHugh

Professor Anwar

5/10/2023

Table of Contents

Motivation

Active noise-canceling devices have become an integral part of our lives, providing relief from the sensory overload experienced in loud and busy environments. Background noise can be highly distracting, and many people wish they could permanently tune out the chaos around them. However, there are times when we need to pay attention to certain individuals, such as friends, family, or colleagues.
With this in mind, our team set out to create a pair of headphones that offer active noise canceling by default, while also allowing users to select specific spatial regions for "enhanced hearing." In essence, our goal was to develop a device with selective hearing capabilities.

This project posed a significant challenge, as it required the seamless integration of real-time audio sampling and processing, as well as multitasking through timer interrupts for audio collection and transmission. Despite these challenges, we were confident in our team's theoretical knowledge and practical experience.

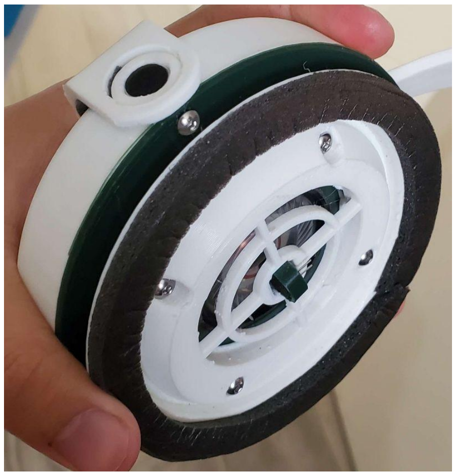


Figure 1: Right Headphone Cup

As a team, this project would pose a difficult challenge to fully implement but we were confident in our theoretical knowledge to produce something that utilized both real-time (audio sampling and processing) and multitasking (timer interrupts for audio collection and transmission). From an individual member standpoint, our team had four members with experience in sound design, microcontroller communication, LabView implementation, and product design.

Physical Design

To achieve effective active noise canceling and spatial recognition, our design incorporated two external microphones, two internal microphones, and two speaker drivers. We developed an ergonomic and comfortable pair of 3D-printed headphones that housed these components, as illustrated in Figure 1.

Figure 2: User Wearing Headphones

To eliminate echoes and reduce unwanted vibrational noise, we integrated foam within the inner cups. This foam not only provided added comfort but also isolated the speaker drivers and microphones from the rigid casing. Additionally, we angled the speaker drivers to minimize the risk of generating unwanted standing waves against the user's head.

Many of our design choices drew inspiration from successful active noise-canceling headphones currently available on the market. One notable aspect of our design was the positioning of the external microphones, which faced the direction of the user's point of view. Theoretically, this arrangement would facilitate more accurate triangulation of specific sound sources when the distance between each microphone is known. This feature is crucial to the overall effectiveness of our selective hearing device.

GUI

We customized LabView's Virtual Instrument Panel (VI) to display the specific goals and tasks of our project. Utilizing ASCII serial protocol for communication, the noise spectrum information for both left and right microphones, along with the five principal frequencies and their corresponding relative intensities, was transmitted to the VI at each timestep. The VI was designed to process each chunk of serial information by parsing it into arrays corresponding to the expected sequence of data and performing the following tasks:

1. **Plot the Fourier Transform** of each microphone for the current timestep, displaying readable information about the surrounding noise environment.
2. Graph a time-buffered frequency vs. intensity **spectrograph**, displaying the most recent 10 sec of data.
3. **Display the 5 Major Sound Signatures** with a bar corresponding to their frequencies. **Allow users to filter back in these major frequencies** by flipping switches corresponding to each MSS (LabView writes an array of booleans back to ESP).
4. **Compute the relative physical positions** of each MSS and plot them on a circular Radar Graph. Positions are calculated by assuming an initial position in front of the user and using the intensity amplitudes from each microphone to compare relative (L/R) time-varying changes in intensity in order to track the sound source's position.

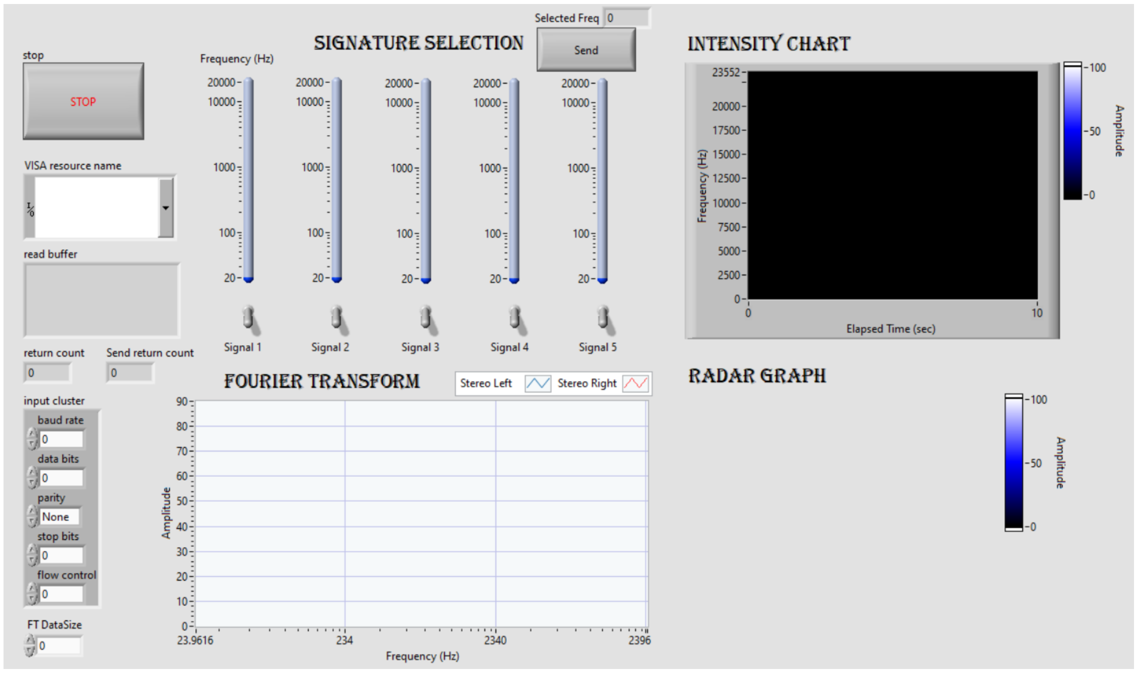


Figure 3: Detailed View of LabView Front Panel

Real-time

The design of noise-canceling headphones necessitates a system that operates at a speed equal to or faster than the speed of sound. This allows for approximately 0.5 ms between when the sounds reach the external microphone and when they arrive at the user's ears. To meet this demanding speed requirement, the system employs a rapid feedforward inverting technique to cancel noise in real-time, along with a slow-updating normalized least mean squared (NLMS) filter to eliminate constant disturbances and compensate for errors. The block diagram illustrating this process can be found in Figure 4 below.

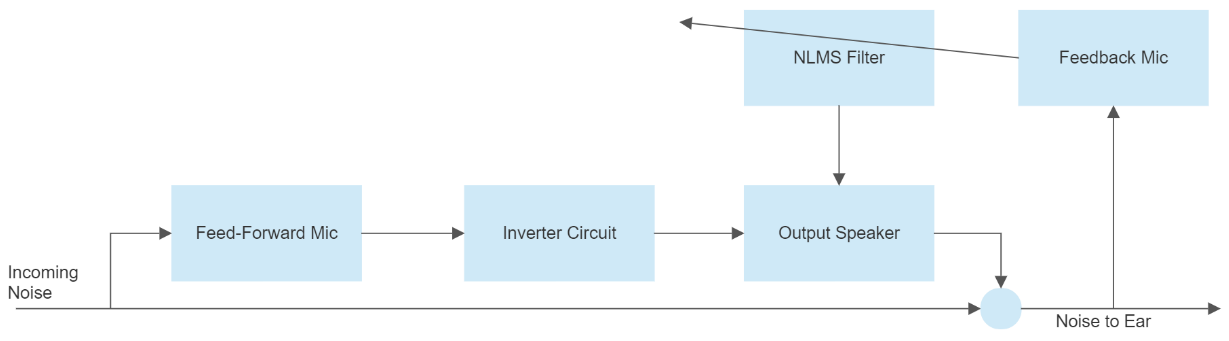


Figure 4: Block Diagram of noise cancelling process

PCB Circuit

Due to hardware constraints, the real-time inverting of audio signals is managed entirely within the circuit. As seen in Figures 5 and 6, we designed and assembled the PCB to handle this inverting process. The circuit takes the microphone signal and, through a series of operational amplifiers (op-amps), inverts it. The board also incorporates an ESP32 and manages all of our microphone data. Separate op-amps provide DC boosting for the microphone data (since the ESP32 cannot handle negative voltages), which the ESP32 reads for all other aspects of the project.

Additionally, the board features a 4-channel, 16-bit digital-to-analog converter (DAC) that communicates with the ESP32 via I2C to output the NLMS filter when updated. The DAC output is combined with the feedforward output before being subjected to a low-pass filter and amplified once more for speaker output. The low-pass filter is essential because, for frequencies above 1000 Hz, the small temporal noise introduced by the circuit may start to generate resonance instead of destructive interference. As a result, these frequencies are entirely disregarded for the active part of our noise-canceling process.

Lastly, the board supports USB-C for power and programming purposes.

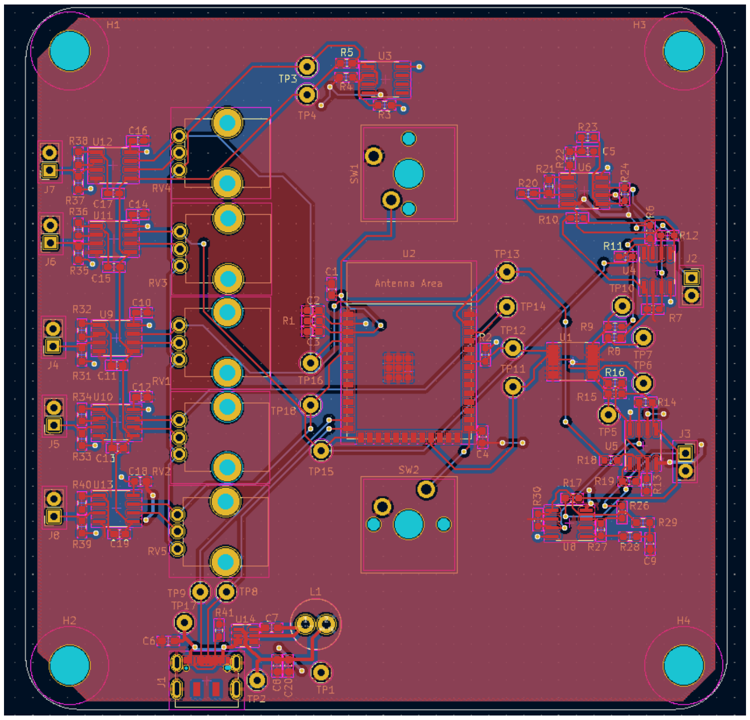


Figure 5: PCB Layout

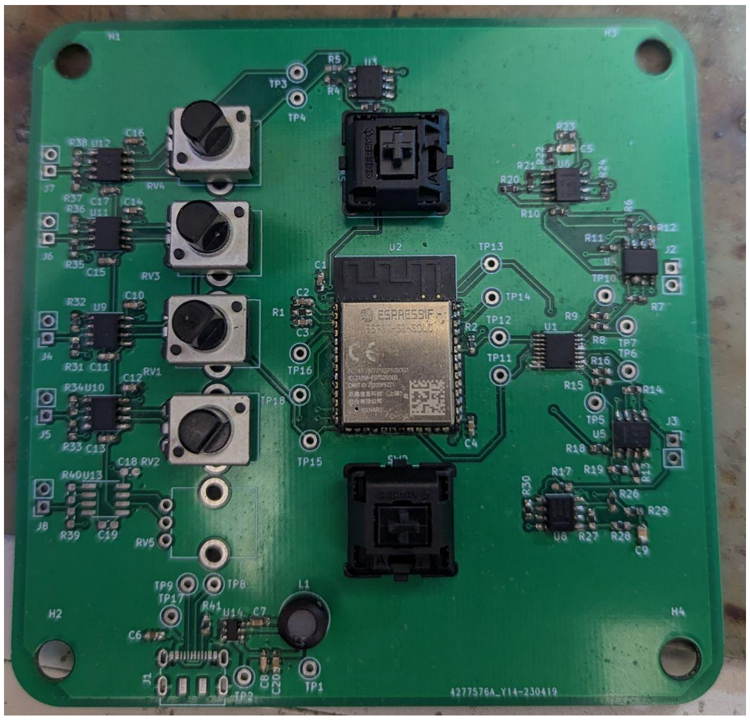
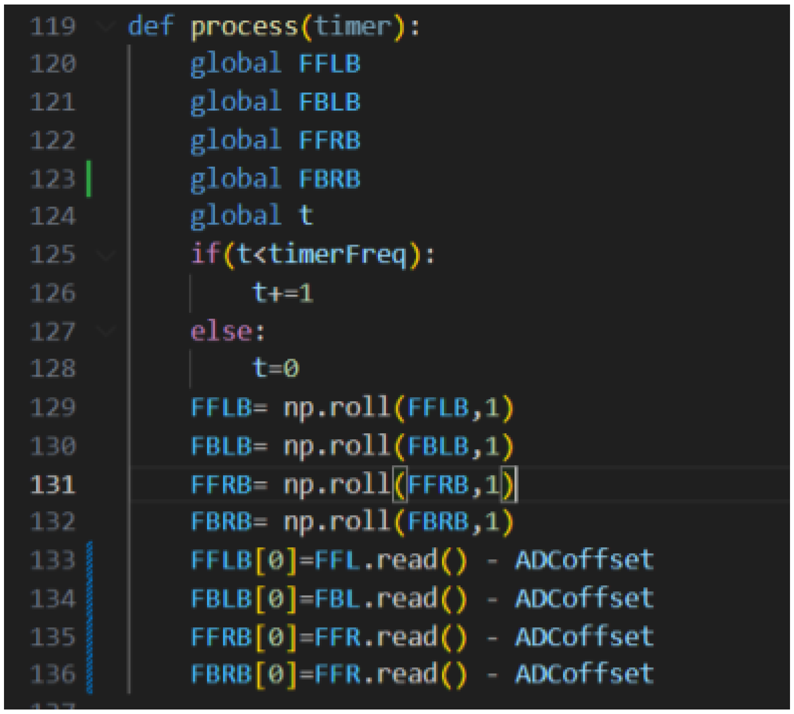


Figure 6: Assembled PCB

Micropython Code

The Micropython code on the ESP32 is responsible for three primary tasks: processing the microphone data, updating and outputting the NLMS filters, and calculating the FFT of the microphone data before sending it to LabView. Each of these tasks is accomplished using a timer (except for the serial output of the FFT, which is an ongoing task). The FFT algorithms and all vector functions utilize the ulab library (https://github.com/v923z/micropython-ulab). We updated the Micropython binary to include this module using CMake, allowing us to leverage its faster functions written in C.

1. The data acquisition timer is set to run at 48 kHz. Achieving this speed with the ESP32 was not entirely feasible, as discussed later. Nevertheless, the timer function in Figure 7 updates ring buffers via the onboard ADC of the ESP32, and these ring buffers are referenced throughout the program for other tasks.

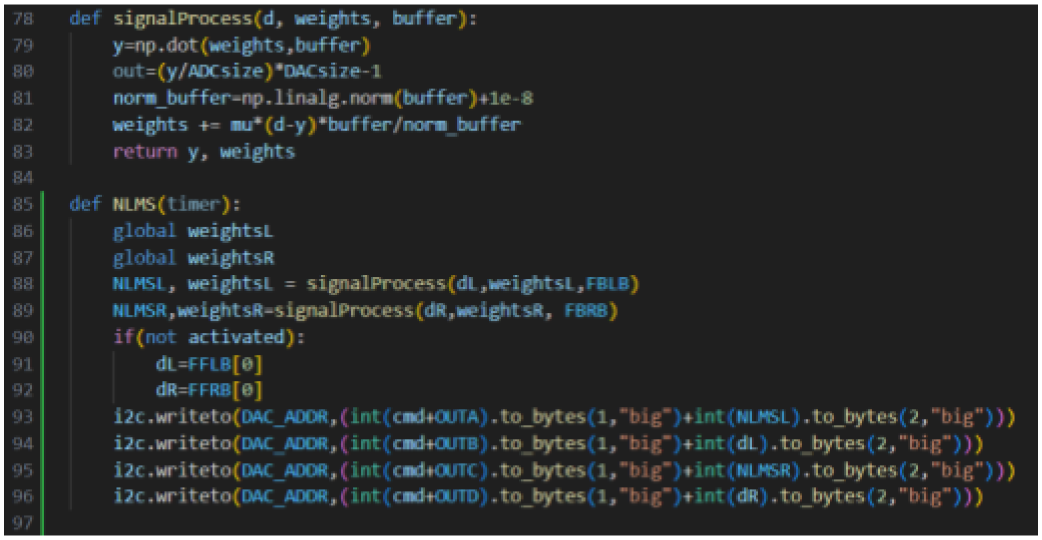


```
def process(timer):
    global FFLB
    global FBLB
    global FFRB
    global FBRB
    global t
    if(t<timerFreq):
        t+=1
    else:
        t=0
    FFLB= np.roll(FFLB,1)
    FBLB= np.roll(FBLB,1)
    FFRB= np.roll(FFRB,1)
    FBRB= np.roll(FBRB,1)
    FFLB[0]=FFL.read() - ADCoffset
    FBLB[0]=FBL.read() - ADCoffset
    FFRB[0]=FFR.read() - ADCoffset
    FBRB[0]=FFR.read() - ADCoffset
```

Figure 7: DAQ timer function

2. The NLMS timer in Figure 8 is set to run at 1 kHz. During testing, this was observed to be the fastest rate the ESP32 could handle, albeit marginally. The timer function processes the ring buffers and past weights from previous NLMS updates, generating new weights and an output for the DAC. This output is then written to the appropriate DAC channels via I2C. The NLMS algorithm itself is relatively simple, utilizing a variable step size based on the distance from the

desired signal to create a signal that converges on eliminating any constant or patterned noises.



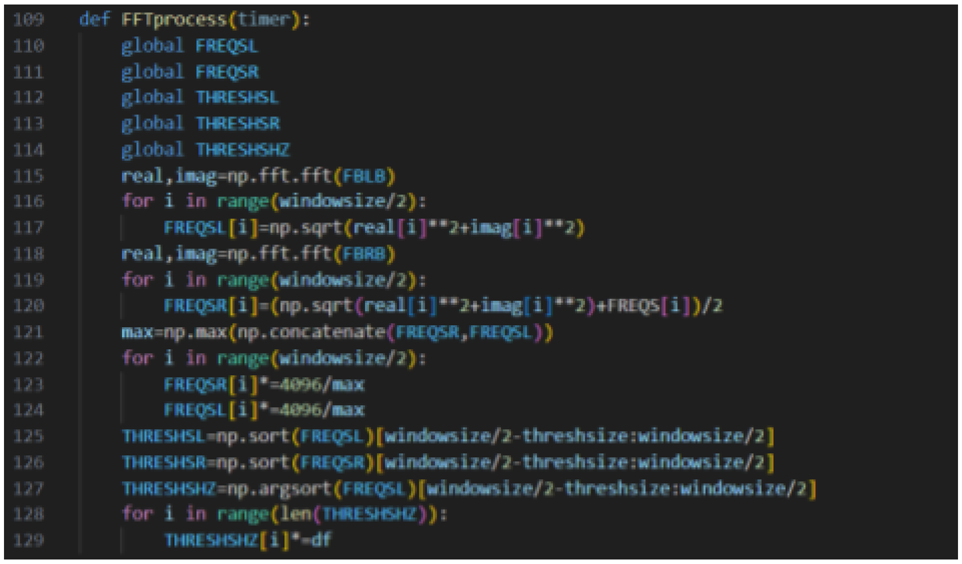
```python
def signalProcess(d, weights, buffer):
    y=np.dot(weights,buffer)
    out=(y/ADCsize)*DACsize-1
    norm_buffer=np.linalg.norm(buffer)+1e-8
    weights += mu*(d-y)*buffer/norm_buffer
    return y, weights

def NLMS(timer):
    global weightsL
    global weightsR
    NLMSL, weightsL = signalProcess(dL,weightsL,FBLB)
    NLMSR,weightsR=signalProcess(dR,weightsR, FBRB)
    if(not activated):
        dL=FFLB[0]
        dR=FFRB[0]
    i2c.writeto(DAC_ADDR,(int(cmd+OUTA).to_bytes(1,"big")+int(NLMSL).to_bytes(2,"big")))
    i2c.writeto(DAC_ADDR,(int(cmd+OUTB).to_bytes(1,"big")+int(dL).to_bytes(2,"big")))
    i2c.writeto(DAC_ADDR,(int(cmd+OUTC).to_bytes(1,"big")+int(NLMSR).to_bytes(2,"big")))
    i2c.writeto(DAC_ADDR,(int(cmd+OUTD).to_bytes(1,"big")+int(dR).to_bytes(2,"big")))

```

Figure 8: NLMS timer function and supporting filter function

3. The FFT timer in Figure 9 is set to run at 10 Hz. This low frequency results from the data throughput limitations of our serial implementation. The function processes the two feedback ring buffers, performing an FFT on them. The amplitude components are then normalized by the maximum amplitude, and the frequencies of the five greatest amplitudes are also stored.



```python
def FFTprocess(timer):
    global FREQSL
    global FREQSR
    global THRESHSL
    global THRESHSR
    global THRESHSHZ
    real,imag=np.fft.fft(FBLB)
    for i in range(windowsize/2):
        FREQSL[i]=np.sqrt(real[i]**2+imag[i]**2)
    real,imag=np.fft.fft(FBRB)
    for i in range(windowsize/2):
        FREQSR[i]=(np.sqrt(real[i]**2+imag[i]**2)+FREQS[i])/2
    max=np.max(np.concatenate(FREQSR,FREQSL))
    for i in range(windowsize/2):
        FREQSR[i]*=4096/max
        FREQSL[i]*=4096/max
    THRESHSL=np.sort(FREQSL)[windowsize/2-threshsize:windowsize/2]
    THRESHSR=np.sort(FREQSR)[windowsize/2-threshsize:windowsize/2]
    THRESHSHZ=np.argsort(FREQSL)[windowsize/2-threshsize:windowsize/2]
    for i in range(len(THRESHSHZ)):
        THRESHSHZ[i]*=df
```

Figure 9: FFT timer function

Multitasking

In our final project, we designed the code to process audio signals in real-time, which requires the efficient handling of tasks concurrently. To achieve this, we have incorporated various multitasking mechanisms into our code, including interrupts, timers, and task prioritization. These mechanisms allow our system to quickly respond to events and ensure smooth operation.

1. Timers and interrupts: Timers are essential for maintaining accurate timing and scheduling tasks at regular intervals. In our code, we employed three timers (t1, t2, and t3) to ensure that specific tasks, such as audio processing, FFT processing, and NLMS processing, were executed at the desired rate. This approach helps maintain consistent performance and prevents the system from being overwhelmed by sporadic task execution.
2. Task Preemption: In our code, we allowed for task preemption, ensuring that higher-priority tasks, such as audio processing, could be executed without waiting for lower-priority tasks to complete. This capability helps maintain consistent real-time performance by ensuring that time-sensitive tasks are not delayed by less critical tasks. However, we realized that sampling at 40kHz was very demanding on the microcontroller we used (ESP32-S2). As a result, data logging and transmission became significantly slower, with serial transmission taking roughly 40 seconds to transfer 1000 data points.

Throughout the project, we faced several challenges related to the real-time processing of audio signals and the implementation of multitasking mechanisms. These challenges provided valuable learning experiences and prompted us to find innovative solutions to overcome them.

1. Resource constraints: Real-time systems often face resource limitations, such as memory, processing power, and communication bandwidth. Balancing the resource requirements of our code while maintaining real-time performance was a significant challenge. During development, we had to optimize memory in the number of bytes due to the limited memory onboard. We were able to address the issue by reducing the sample buffer size and decreasing the sampling rate, but it was only a temporary fix, and will not be used in the future should we pursue further development.
2. Handling edge cases and error conditions: Real-time systems must be able to gracefully handle edge cases and error conditions to ensure reliable operation. Identifying and handling these situations in our code required thorough analysis and the implementation of appropriate error-handling mechanisms. We had to

ensure that our system could recover from errors and continue to operate effectively without compromising its real-time performance.

By addressing these challenges and attempting to develop a robust and efficient real-time audio processing system that met our project's requirements, we have gained valuable insight into working with real-time systems and multitasking environments.

Reflection

Physical Design Future Enhancements

1. **Circuitry Integration:** Developing a custom circuit board with efficient design and communication would allow for a better fit within one of the headphone cups.
2. **Additional microphones:** While two external microphones are sufficient, their positioning biases them towards sampling sound primarily in front of the user. Incorporating two additional external microphones facing the rear would eliminate this bias and provide a more accurate environmental sampling.
3. **Distance between Microphone Detection:** Attaching a flexible strain gauge to the headband could potentially enable automatic detection of the headband's stretch and, consequently, the distance between the external microphones for triangulation.

GUI Future Enhancements

1. **Improved Data Communication:** Attempting to sample high-frequency data while writing through ASCII serial protocol proved very difficult for our processors, resulting in significant amounts of lost data upon sending it to LabView. In the future, we'd tailor our communication protocol to use the lightest-possible communication protocol to send a very carefully slimmed package of information to LabView.
2. **More Robust Data Parsing:** Due to data communication problems, the current method of array size-based data parsing proved insufficient. Organizing data and sending smaller packages (MSS information) before sending the slimmed FTs would enhance robustness.
3. **Better debugging:** Since our primary issues with the project lie in the microprocessor sampling and communication, we could have developed a debugging platform to simulate serial communication with LabView and more efficiently test both project facets in parallel.

Real-Time Future Enhancements

One of the primary challenges our project faced was speed. Initially, a 48kHz timer interrupt was beyond the capabilities of the ESP32 or Micropython. The solution to this issue can be found in hardware, software, and design. On the hardware side, a faster microcontroller, such as an STM32 with its ARM architecture, would have alleviated some of the speed concerns, allowing for a more efficient system overall. In terms of software, switching from Micropython to C, while labor-intensive, would have resulted in a much faster program.

From a design perspective, the 48kHz timer could have been reduced to a much slower one with a more intentional circuit design. The 48kHz timer was necessary to ensure that when the FFT of the data was taken, frequencies up to 24kHz (human hearing) were preserved. However, as our professor pointed out, real systems avoid this issue by leveraging Nyquist folding. By using band filters, data could be sampled at a much lower frequency (e.g., 5kHz), and higher-frequency components could be read from their folding over in the FFT. Implementing these three changes would result in a faster processor, more efficient code, and reduced workload for the system.

Another issue worth noting is the speed of our serial data communications. In our current implementation, we used strings to send integers, which could have been significantly improved if the serial was configured to send bytes instead. Additionally, serial communication is inherently slow, particularly for the volume of data our system gathers. By creatively using USB drivers, the D+ and D- pins on the USB input could be configured like SDA and SCL pins for I2C. While this change would move us away from the general applicability of serial communication, it would allow for much faster data acquisition transfer rates.

Multitasking Future Enhancements

To develop multi-tasking further, we must consider several factors, such as processing power, multi-core microcontrollers, and optimizing both the audio collection method and communication protocols.

Upgrading to a more powerful microcontroller would allow for better performance in multitasking. Multi-core microcontrollers can be employed to address the various tasks concurrently. For example, one core could be dedicated to audio sample collection, while the other prioritizes data transfer. This would result in more efficient and faster processing, ultimately improving the overall functionality of the selective hearing device.

In addition to upgrading the hardware, it is essential to optimize the audio collection method and algorithm. This involves refining the sampling rate, enhancing the noise-canceling techniques, and improving the selective hearing algorithm. By doing so, the system can process sound more effectively, providing the user with a better experience.

Another crucial aspect to consider is the optimization of communication protocols. As mentioned earlier, transitioning from serial communication to a faster, more efficient protocol such as I2C would greatly enhance data transfer rates. This would be particularly beneficial for the system as it would reduce data loss and improve real-time processing of audio signals.

Additional Comments

It is important to note that our project faced significant challenges due to supply chain issues. Almost all of our electrical hardware components took over four weeks to ship, arriving only after our project showcase. This severely limited our ability to fully debug and refine our project within the given timeframe. The parts were ordered from DigiKey, but as a workaround, we ordered essential components from Mouser a week and a half before the showcase, based on a recommendation.

In response to these challenges, we developed code that simulated the data collection and processing operation. We also designed a detailed LabView VI and physical headphones to present our work despite the setbacks. Although the supply chain issue was beyond our control, exploring alternative shipping options earlier might have helped mitigate some delays.

In future projects, it will be crucial to consider potential supply chain disruptions and take proactive measures to minimize their impact on project timelines and overall success. This may involve sourcing components from multiple suppliers, anticipating potential delays, and developing contingency plans to ensure continuous progress despite unforeseen challenges.

Appendix

Cad and Pictures Model

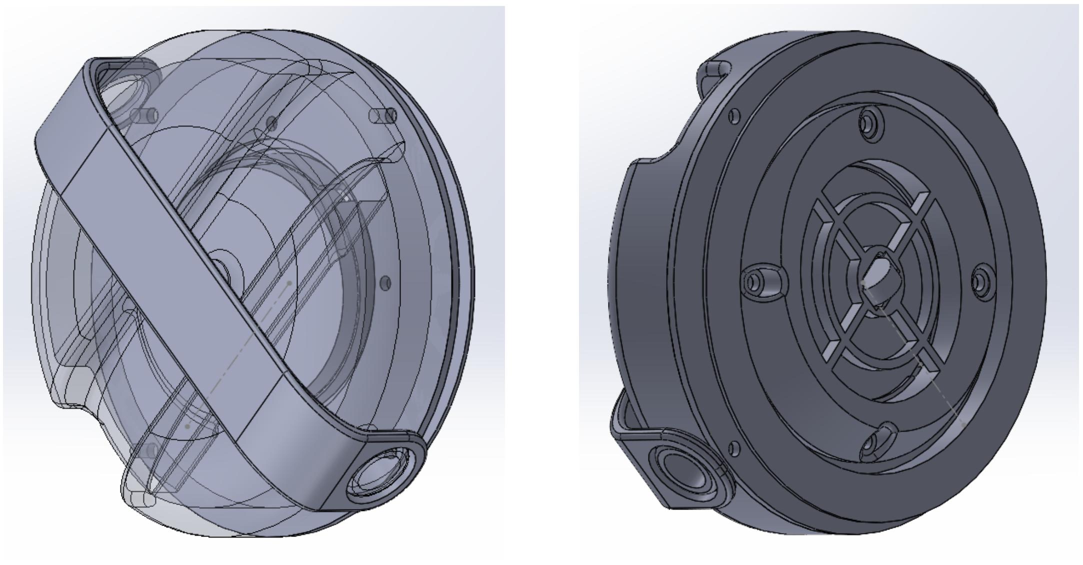


Figure A: Isometric Views of Headphone Cup

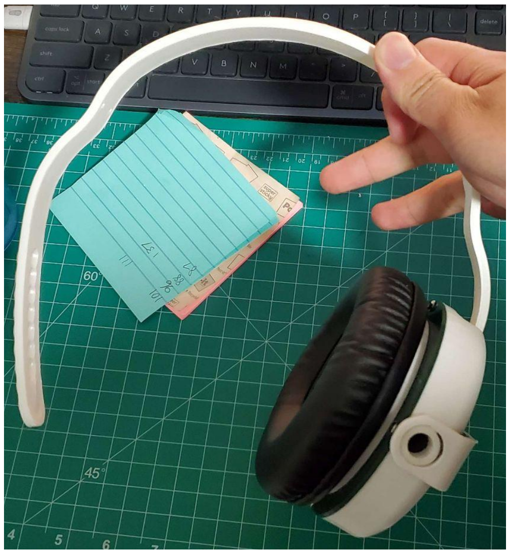


Figure B: Headband Integrated with Earcups

Wiring Diagram

We have attached the wiring diagrams as additional pdfs. Please check out PCB_diagram.pdf for the PCB design, and wiring_diagrams.pdf for other electronic components, including the ESP, DACs, and Mics.