Microphone Processing Audio system using STM32 Microcontroller

ANU College of Engineering, Computing and Cybernetics
Australian National University
Canberra ACT 0200 Australia
www.anu.edu.au

+61 2 6125 5254

Student 1 ID	u7114996		
Student 1 Name	Brady Edwards		
Student 2 ID	u7310067		
Student 2 Name	Aedan Jaeson		
Course Code	ENGN4213/6213		
Course Name	Digital Systems and Microprocessors		
Assignment Item	☒ MCU Project Report		
Word Count	2710	Due Date	03 June 2024
Date Submitted	02/06/2024	Extension Granted	

I declare that this work:

☒ upholds the principles of academic integrity, as defined in the University Academic Integrity Rule;

☒ is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the course outline and/or Wattle site;

☒ is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;

☒ gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;

☒ in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

Initials

For group assignments, each student must initial.

B.E

A.J

CONTRIBUTION STATEMENT

We, the undersigned members of ENGN4213/6213 FPGA Project Group No: 27, hereby state our main individual contributions.

- .wav file handling/SD card writing
- LCD setup
- ADC/DMA setup/handling
- UART setup
- Robot effect
- Report writing

Signature 1 with name:

Brady Edwards

Date: 02/06/2024

- FFT/high pass filtering (audio processing)
- FATFS configuration
- Hardware configuration (physical connections)
- UART setup
- Robot effect
- Report writing

Signature 2 with name:

Aedan Jaeson

Date: 02/06/2024

This statement may be considered to assess fair contributions among group members and, if deemed necessary, regulate individual marking.

Contents

Introduction

This report describes the design and implementation of a microphone processing audio system using the STM32F411RE microcontroller. The system is designed to enhance the voice chat experience in a VR MOBA game, providing two operational modes: ‘normal mode’ for standard voice recording and ‘high privacy mode’ for voice modification with a robot effect. The system performs real-time audio processing, including low-frequency noise filtering and frequency spectrum analysis. This report details the system design, hardware and software components and evaluates the effectiveness of the implementation.

System Design

2.1 System Setup

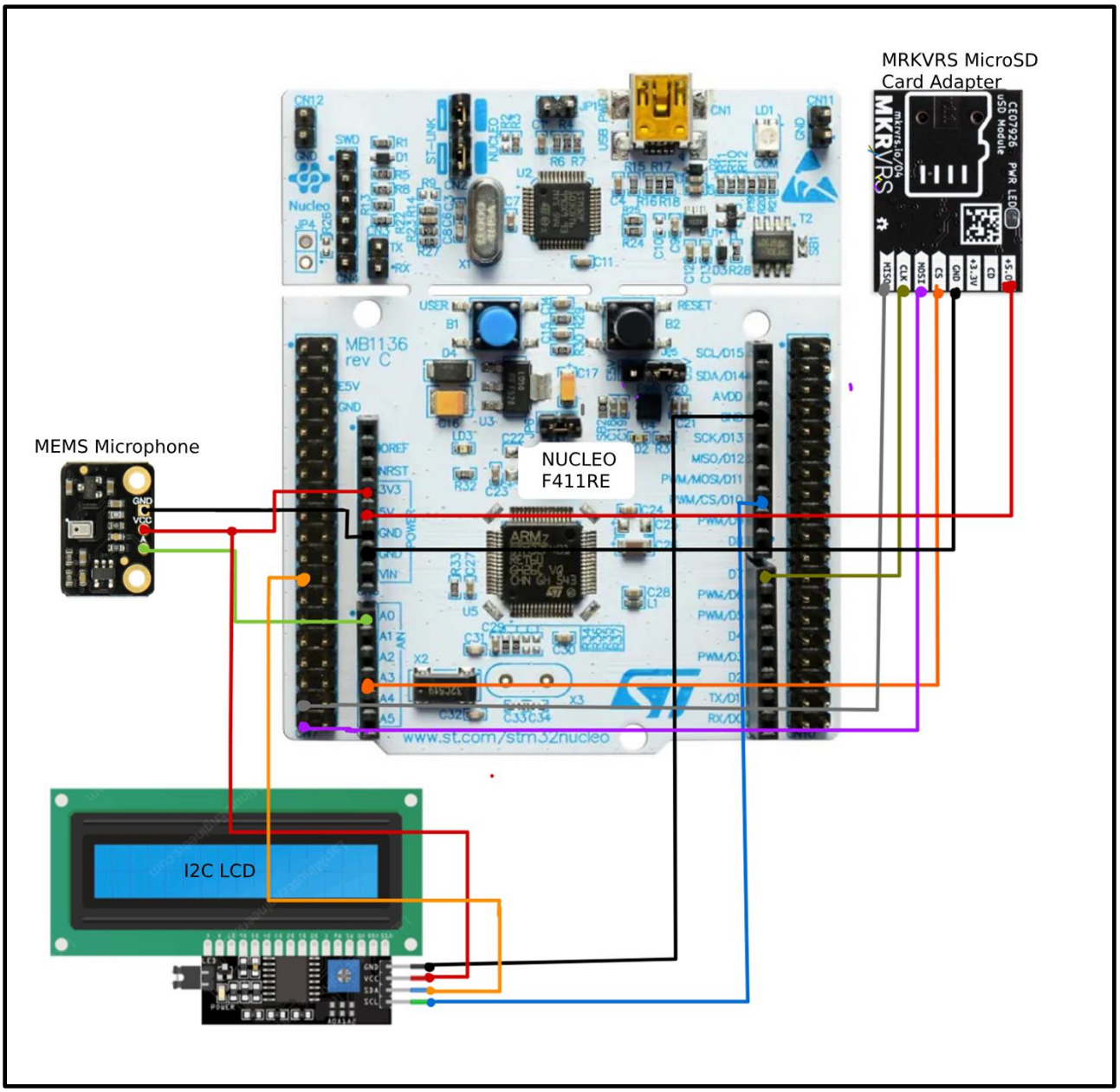

Figure 1: Hardware Connection Schematic

The included hardware components of the system are the NUCLEO-F411RE microcontroller, Makerverse MicroSD card adapter, microSD card, MEMS Microphone Module, I2C LCD, breadboard, interconnecting jumper wires and a USB connection to PC. Audio data is acquired from the MEMS microphone module, a compact microphone that produces an analog signal from picked up sound.

2.1.1 Hardware Components

Component	Component Description
NUCLEO-F411RE Microcontroller	The main processing unit

Makerverse MicroSD Card Adapter	Interface for microSD card
MicroSD Card	Storage medium for audio data
MEMS Microphone Module	Sensor for audio input
I2C LCD	Display for real-time data
Breadboard and Jumper Wires	For circuit connections
USB connection	For power and data transmission to PC

2.1.2 Software Components

Software components include the STM32CubeIDE, SerialPlot, and MATLAB. The primary development and configuration are performed using the STM32CubeIDE, which facilitates:

- SD card file system management: using the FatFS file system libraries.
- Analog to Digital Data Conversion: configuring the ADC for audio signal capture.
- HAL Libraries: Hardware abstraction layer for STM32 peripherals
- Digital Signal Processing: Using CMSIS DSP Library

2.2 MEMS Microphone

2.2.1 Microphone Implementation

The MEMS microphone module captures the audio signals and converts them into analog signals. These analog signals are then fed into the ADC pin of the microcontroller PA0 (See figure 1) for further processing.

2.3 Microprocessor System Design

2.3.1 FATFS

The Generic FAT filesystem (FATFS) module is used to provide a layer of abstraction when writing to the external SD card, such that it can be treated as the local filesystem from the microcontroller's perspective. The module is configured with long filename support (to extend the base 13-byte filename max length) and a sector size range of 512-4096 to support different physical drives (which might have varied sector sizes). However, most SD cards typically have smaller sector sizes.

2.3.2 ADC with DMA

Relevant Calculations:

Sampling Frequency: $\frac{\frac{Clock\ frequency}{prescaler}}{(sampling\ time + conversion\ time)} = \frac{\frac{72MHz}{8}}{(15+480)} = 18181Hz$

To process audio signals digitally, the raw audio input from the microphone was passed through an audio to digital converter (ADC). The ADC module was configured to use direct memory access (DMA), such that the digital audio data would be quickly passed from peripheral to memory automatically in the process background. Using ADC with DMA, the analogue audio was processed and converted to digital audio continuously, writing to a circular buffer in memory.

The converted digital audio was processed when the ADC buffer was half-filled, and fully filled. This was done to prevent overwriting the digital audio buffer during signal processing while the ADC module continued to write samples to the buffer.

The ADC buffer size is configured to be 16384 bytes (8192 elements each 2 bytes in size), so that the ADC buffer takes longer to half-fill up and gives the microcontroller more time to process the audio data. This works because despite increasing the number of atomic sector-sized writes to the SD card, other functions such as I2C->LCD do not scale linearly with increases to the ADC buffer. Additionally, the microcontroller only performs frequency analysis on a portion of the audio data received.

The resolution of each ADC sample is 12 bits (15 clock cycle conversion time), with a sampling time of 480 cycles. The clock pre-scaler was set to 8, such that the effective clock frequency was at 9MHz. These values were chosen to set the overall ADC sampling rate at ~18kHz. This value was chosen over the more conventional 44kHz, as all the functionalities required in this project could only be completed within ~22ms before half the ADC buffer was half-filled. So, to preserve all recorded audio data, a lower sampling rate was used.

2.3.3 UART

Universal asynchronous receiver/transmitter (UART) was used to transmit the pre-processed digital audio data to the connected computer for plotting. UART was chosen for this as it is a simple and easy serial communication protocol for establishing communications between computers/laptops and the microcontroller. Additionally, there exists software such as “Hercules” and “SerialPlot” which are very easy to set up with UART. Given that the sampling rate for the digital audio data is 18KHz, a baud rate of 115200b/s was more than enough to ensure that transmission speeds for the audio data were sufficient. This value was chosen to exceed the sampling rate, as further increases to transmission speed give more space for the microcontroller to perform the rest of its required operations (considering it only has a narrow window before the next audio sample needs to be processed).

2.3.4 SPI

The Serial peripheral interface (SPI) protocol is used to establish a serial line between the SD Card and the microcontroller. The Baud rate is 18Mb/s (default for SPI in STM32), which exceeds the required data rate to keep up with the audio data sampling rate when writing the audio data to the SD card. Similar to the UART section, exceeding the required data rate enables the microcontroller to spend more time doing the rest of its required functions before half of the ADC buffer is filled.

SPI is also used for SD card writes because it supports full duplex communication. This is important because any return values from write operations or other functions like f_stat need to be received by the microcontroller. Supporting full duplex communications for file/disk IO operations allows for faster operations.

2.3.5 I2C

For the external liquid crystal display (LCD), the inter-integrated circuit (I2C) serial protocol is used for fast half-duplex communications. The program never reads from the LCD, so data is written in only one direction (master to slave), meaning full-duplex communication is not required. I2C is configured to write at 100kHz, which is more than enough to facilitate semi-frequent (1Hz) writes with low byte counts to the LCD device. Additionally, STM32 provides helpful configurations for easily communicating with the LCD slave device in the STM32 pinout and configuration file.

2.3.6 GPIO

General purpose input/output provides an interface between the microcontroller and external circuitry and devices. GPIO provides a pin interface for all other modules in this project (such as SPI, I2C, and ADC) so that the corresponding external devices can be easily connected to the microcontroller.

This pin interface is also used to configure some components on the physical board that houses the microcontroller, such as the blue push-button which toggles the “privacy mode” feature.

2.3.7 Interrupts

4 user-defined interrupts were used in the program:

1. An interrupt triggered when half of the ADC buffer is filled, such that the first half of the buffer can be processed while the second half is being filled (prevents overwriting data that hasn’t been processed yet).
2. A second corresponding interrupt triggered when the ADC buffer is fully filled (same reasoning as above).
3. A timer interrupt set to trigger every second (1Hz) for updating the LCD screen.
4. A GPIO push-button interrupt to trigger whenever the blue push-button is pressed, to toggle the privacy mode.

The priority for these interrupts was configured such that the blue pushbutton has highest priority (so that it doesn’t get smothered by the ADC interrupts), followed by the timer and ADC interrupts. This was done because the ADC interrupts require the most amount of time to complete (the FFT/high-pass filter and SD card writes are done during these interrupts), so should have the lowest priority to prevent the other interrupts from being smothered.

2.4 Microphone Audio Processing

The microphone audio processing is divided into two main modes: Normal Mode and Privacy Mode. Switching between modes is triggered by a GPIO push-button interrupt, which toggles the privacy mode on or off. The processing of audio data is handled in the DMA half and full callback functions, which allows for real-time processing.

2.4.1 Normal Mode

In Normal Mode, the audio data undergoes high-pass filtering to remove low-frequency noise. The 'apply_high_pass_filter' function processes the audio samples to enhance the clarity of the signal. The high pass filter helps in eliminating frequencies below 250 Hz, which typically consist of background noise or low-frequency hum.

2.4.2 Privacy Mode

Privacy mode introduces the robot voice effect, which modifies the audio signal to mask the original voice. The robot effect is created by adding delayed versions of the original audio signal to itself. Initially, the audio data undergoes high-pass filtering to remove low-frequency noise. The robot effect function then adds delayed versions of the original signal to itself, creating the robotic effect by combining the original and delayed signals at specific offsets (200 and 400 sample delays). After applying the robot effect, the signal undergoes a second instance of high-pass filtering to ensure clarity and to make it harder to find out who is speaking.

2.4.3 SD card writing

The following files are written to the SD card while the program is running:

- Unprocessed audio file in .wav file format
- Processed audio file in .wav file format

The "wav_helper" source and header files handle creating the valid .wav file header and appending data to the file (as metadata needs to be updated based on the overall file length).

2.5 Audio Filtering and Frequency Spectrum Analysis

Audio filtering and frequency spectrum analysis ensures that the audio data captured by microphone is processed efficiently and accurately.

2.5.1 High Pass Filter

A high-pass filter (HPF) is used to remove low-frequency noise from the audio signal, allowing higher frequencies to pass through. The design of our HPF is based on the finite impulse response (FIR) filter, which was implemented using the 'firwin' function from the Python 'scipy.signal' library.

The script for generating the filter coefficients used the following code:

```
from scipy.signal import firwin

fs = 18181
cutoff = 250
numtaps = 5

coeffs = firwin(numtaps, cutoff, fs=fs, pass_zero=False)
print(coeffs)
```

The firwin functions take the parameters of our sampling frequency, desired cutoff frequency, and the number of taps (coefficients). Setting the argument ‘pass_zero’ to false specifies that the filter is designed to not pass the 0 frequency, effectively blocking low-frequency components, and turning it into a high-pass filter. These coefficients were then generated by the code:

```
-0.00219403, -0.01486513,  0.9746578,  -0.01486513, -0.00219403
```

The choice of using 5 taps for the filter was a trade-off between filter performance and computational efficiency. A lower number of taps results in fewer computations per sample, which is essential for real-time processing with limited computational resources. Increasing the number of taps can improve the performance by providing a sharper cutoff and better attenuation but also increases the computational load. After testing

different configurations, 5 taps were found to provide an adequate balance.

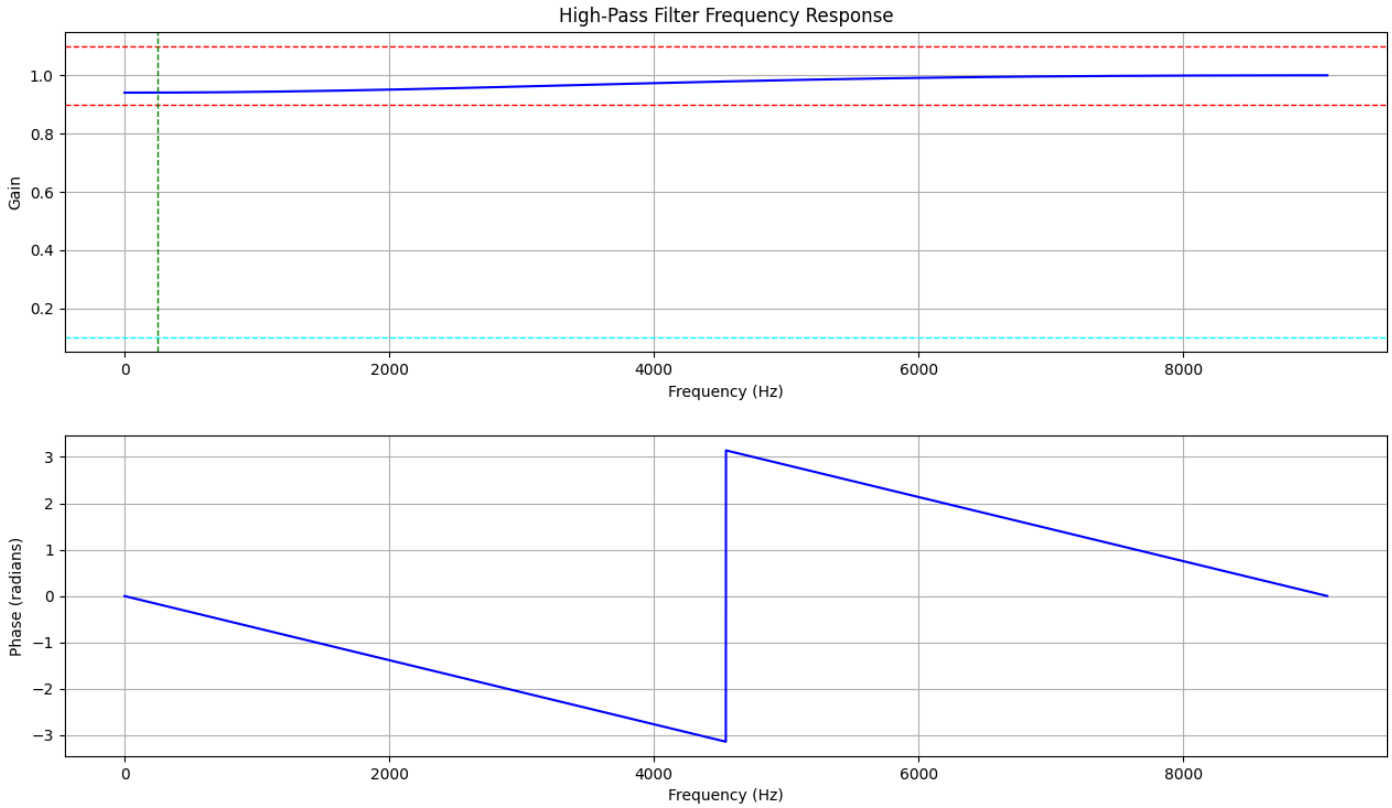

Figure 2: Frequency Response Plot

To verify the effectiveness of the HPF, we plotted the frequency response and as shown in figure 3, frequencies below the 250 Hz cutoff are significantly attenuated, confirming that the filter blocks low-frequency noise. The gain is near 1 for frequencies higher than the cutoff frequency, indicating that these frequencies pass through with minimal attenuation.

The coefficients generated were then used in our 'apply_high_pass_filter' function, which processes the audio samples to enhance the clarity of the signal by removing frequencies below 250Hz. In the implementation, we utilized the 'arm_fir_f32' function which applies the filter to the audio samples. This ensures that the high pass filtering is performed with optimal performance on the ARM processor.

2.5.2 FFT and Frequency Spectrum Analysis

The Fast Fourier Transform (FFT) is used to analyse the frequency components of a signal. In our system, the FFT is implemented using the CMSIS DSP library, which provides optimized functions for the ARM Cortex-M processors. The 'apply_fft' function uses the 'arm_rfft_fast_f32' function to perform the FFT on our audio data.

The 'apply_fft' function starts by converting the input ADC data to 'float32' format to be compatible with the CMSIS DSP functions, which is essential as the FFT algorithm operates on floating-point data. The function

then performs the FFT, transforming the signal from the time domain to the frequency domain. The magnitudes of the frequency components are calculated by taking the square root of the sum of the squares of the real and imaginary parts. The index of the maximum magnitude is identified, corresponding to the dominant frequency.

The dominant frequency is calculated using the following formula:

$$Dominant\ Frequency = \frac{Sampling\ Rate}{FFT\ Size} * Index\ of\ Max\ Magnitude$$

This calculation provides the frequency in Hertz, allowing us to analyse the main components of the audio signal. An FFT size of 512 ensures that we have sufficient frequency resolution to identify significant components while maintaining computational efficiency.

The FFT spectrum analysis processing through the microphone was verified through a phone app called 'tone gen' which would generate tones based on the given frequency, and this was compared to the frequency reading on the LCD.

2.6 Limitations

The primary limitation of the program is sending the audio data to the connect computer through UART, processing the audio data (high pass filter and FFT) and writing both the raw and processed audio data to the SD card every time the ADC half-filled or fully filled buffer interrupts trigger. To mitigate this issue, the sampling rate, FFT block size, and LCD update frequency were all reduced to decrease the computational complexity of the functions the microprocessor completes each time the ADC interrupt triggers.

Without these mitigations, the output audio data sounds truncated, since there is audio data not being written to the .wav files (the microcontroller is still processing the previous audio block by the time the next half of the ADC buffer is ready).

Currently, the audio data sounds slightly truncated when writing both the raw and modified audio .wav files. Where the program runs seamlessly only writing the modified audio .wav file. To only write the modified .wav file for crisper audio, set the "write_only_mod" flag in "main.c" to 1.

2.7 External Display

2.7.1 Audio Signal Display (SerialPlot)

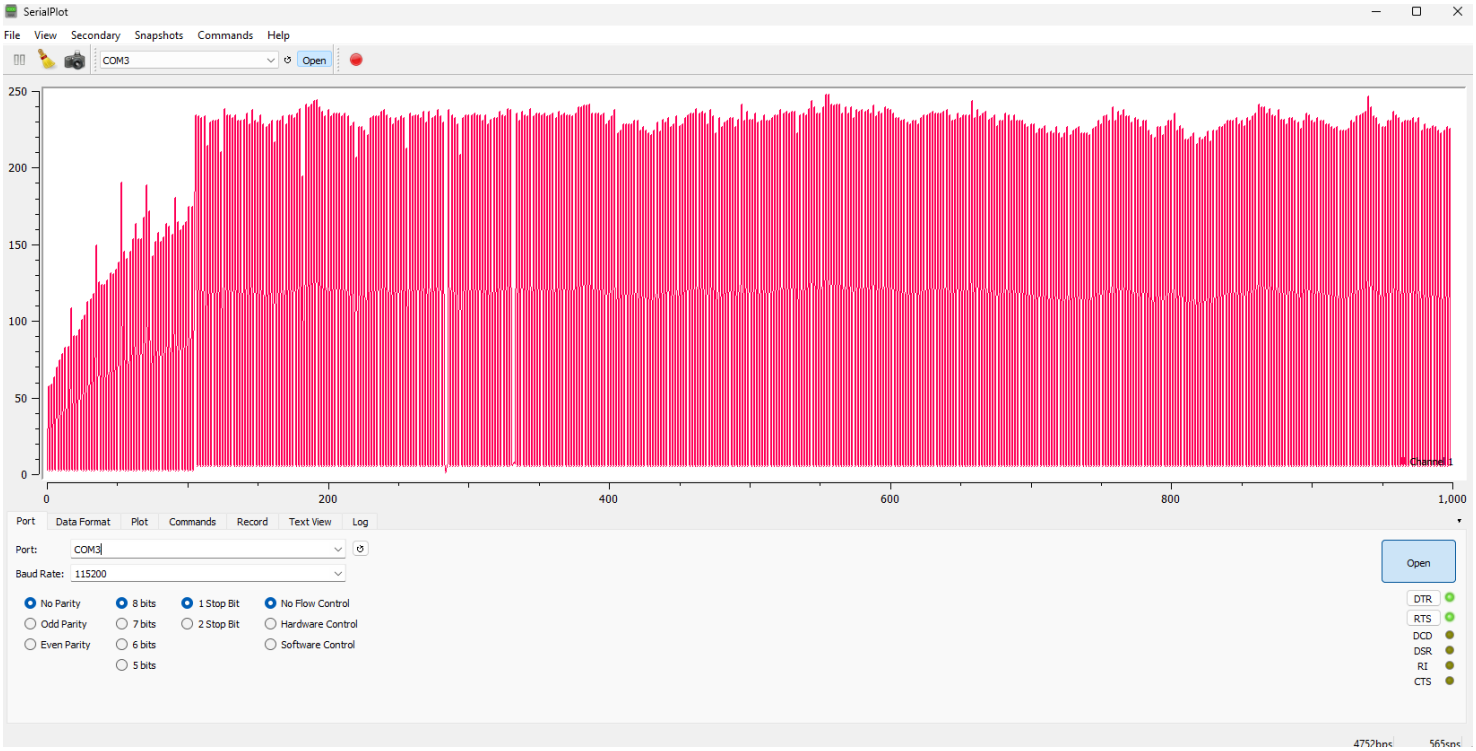

Figure 3: Plotting Audio Data with SerialPlot

The serial plot window shown in Figure 4 captures and displays an audio signal in real-time. The system is connected through port COM3, with a baud rate of 115200, which was the data transfer needed for efficient transfer and real-time plotting.

The graph displays a continuous stream of data representing the audio signal. The amplitude of the signal is represented on the Y-axis, while the time or sample index is on the X axis.

2.7.2 Advanced Feature Implementation

When no significant microphone data is picked up, the system detects the absence of data and ceases output, preventing unnecessary data transmission and ensuring efficient use of system resources. If the amplitude falls below a specified threshold of 1850, indicating no significant audio signal, the system refrains from sending data to SerialPlot.

2.7.3 Frequency Spectrum and Mode Display (LCD)

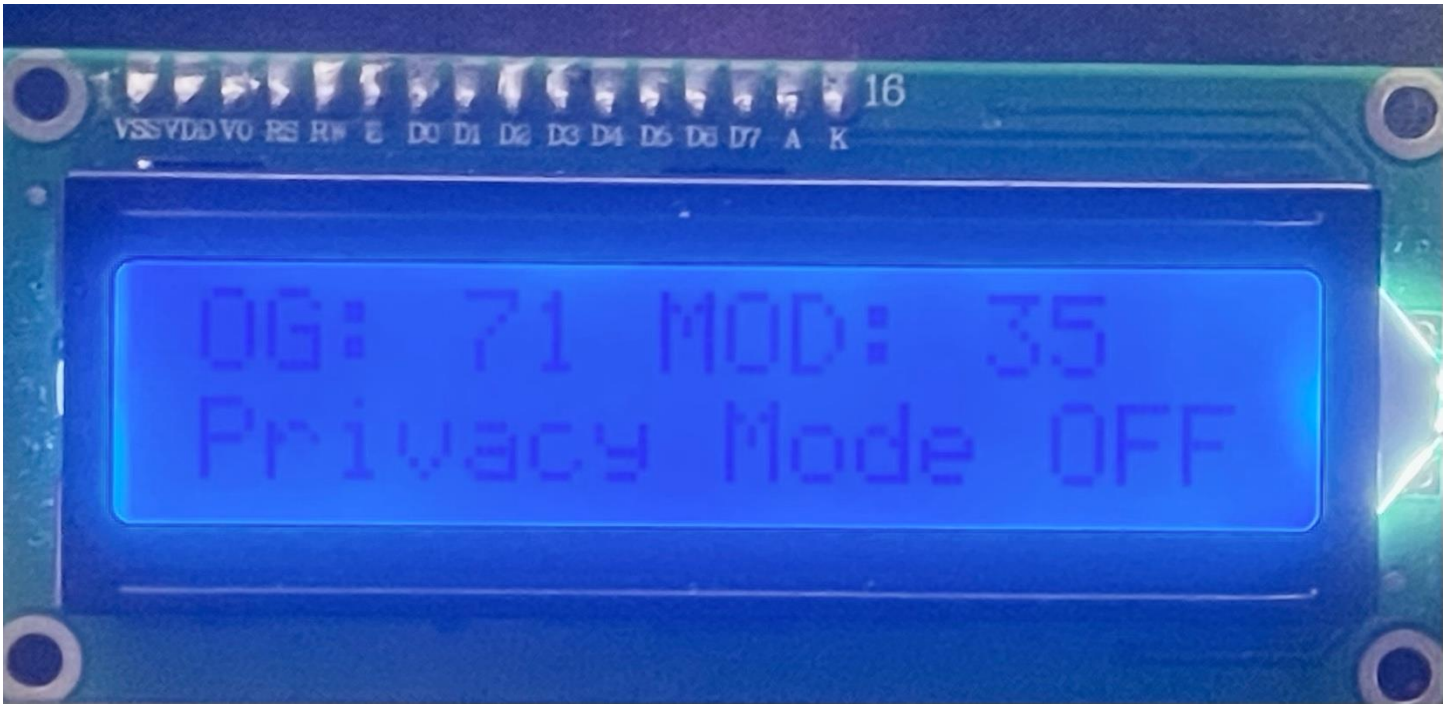

Figure 4: System LCD Screen

Figure 5 displays the frequency spectrum and mode settings on the I2C LCD screen. What is being displayed is the OG (Original Signal Frequency) and MOD (Modified Signal Frequency), these labels were shortened to fit on the max 16 characters of the LCD screen. The OG signal is the audio signal without the high pass filter of 250hz, and the MOD signal is the filtered signal. The values being displayed are the frequency spectrums.

In the HAL ADC conversion callbacks, the FFT processing functions are invoked to compute the dominant frequencies of both the original and filtered signals.

References

Standford, “WAVE PCM soundfile format”, https://ccrma.stanford.edu/courses/422-winter-2014/projects/WaveFormat/

scipy.signal.firwin, [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.firwin.html.

ARM, "CMSIS-DSP Software Library," [Online]. Available: https://www.keil.com/pack/doc/CMSIS/DSP/html/index.html.

E. Chan, "FATFS Generic FAT File System Module," [Online]. Available: http://elm-chan.org/fsw/ff/00index_e.html.

Digikey, “Getting Started with STM32 - Working with ADC and DMA”, https://www.digikey.com.au/en/maker/projects/getting-started-with-stm32-working-with-adc-and-dma/f5009db3a3ed4370acaf545a3370c30c