Abdullah Gül University

Department of Electrical and Electronics Engineering

SAPA Capsule Project

Electric Stethoscope

Project Team:
Barış DEMİRCİ agu@338.rocks
Selin Nisa AKGÖL selinnisa.akgol@agu.edu.tr
Nuri İNCEÖZ nuri.inceoz@agu.edu.tr

December 30, 2024

Contents

Chapter 1

Introduction

1.1 Background

The stethoscope, a critical tool in medical diagnostics for nearly two centuries, has been a staple in physicians' practices worldwide. With technological advancements, digital stethoscopes have emerged as modern alternatives, offering enhanced auscultation capabilities. These devices are cost-effective easy to use and capable of capturing both normal and abnormal breath sounds with high accuracy. This project aims to develop an electronic stethoscope system capable of acquiring and processing acoustic signals from various human body parts.

1.2 Objective

The primary goal is to design an electronic stethoscope system that acquires acoustic body sounds, converts them into digital signals, and transmits them to a computer for real-time monitoring and analysis. The system will include hardware and software components for:

- Signal acquisition
- Amplification and filtering
- Digital conversion and transmission
- Real-time tracking via a graphical user interface (GUI)

1.3 Scope

- The project involves designing custom amplification and filtering circuits without using commercial modules.
- Arduino will handle digital conversion and data transmission.
- Python will be used for GUI development, ensuring smooth signal monitoring and analysis.

Chapter 2

Hardware Development

2.1 Circuit Design & Simulation

2.1.1 Electret Microphone Circuit

First, tried to simulate the electrec microphone circuit using LTSpice. The circuit is shown in the figure below.

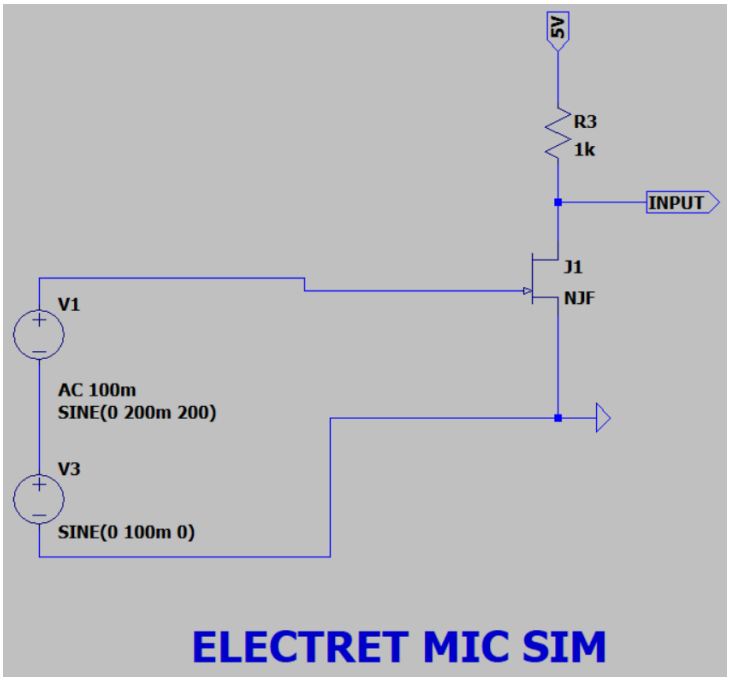

Figure 2.1: Electret Microphone Circuit

A 2-wire electrec microphone consists of one capacitor and one N-channel JFET. Using the circuit shown in Figure 2.1, we we able to acquire the voice signal from the microphone. Signal was too weak and too noisy to be used directly, so we needed to amplify and filter it.

2.1.2 Filter Circuit

High-Pass & DC Filter

Incoming signal was oscillating around 5V and had a lot of noise. We were planning to Arduino Uno for signal processing, so we needed to convert the signal to 0-5V range. We also needed to filter the noise. We designed a filter circuit using LTSpice.

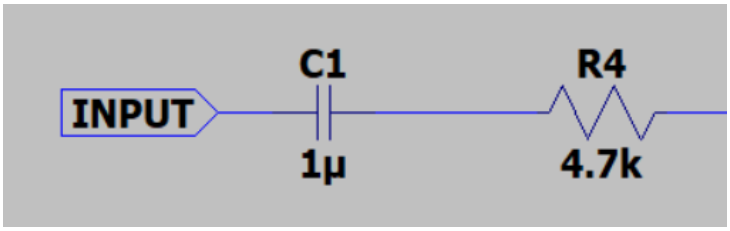

Figure 2.2: High-Pass Filter Circuit

Used a passive high-pass filter setup as shown in Figure 2.2 for both filtering DC offset and low frequency noise. The cutoff frequency can be calculated using the formula:

$$f_c = \frac{1}{2\pi RC} \tag{2.1}$$

In our case, we used R = $4.7k\Omega$ and C = 1μF, so the cutoff frequency was 33.8Hz. This filter was able to filter out the DC offset and low frequency noise.

Low-Pass Filter

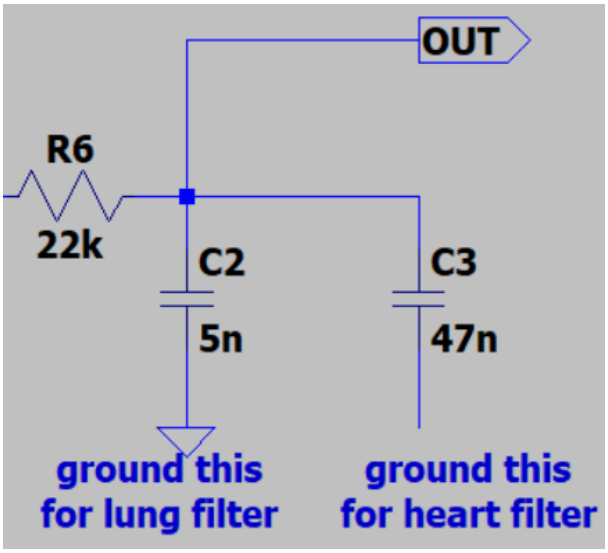

Figure 2.3: Low-Pass Filter Circuit

Again, used a passive low-pass filter setup as shown in Figure 2.3 for filtering high frequency noise. The cutoff frequency can be calculated using the formula 2.1. In our case, we used R = $22k\Omega$ and C = 5nF for lung, C = 47nF for heart, so the cutoff frequency was 153Hz for heart and 1446Hz for lung. This filter was able to filter out the high frequency noise.

2.1.3 Amplifier Circuit

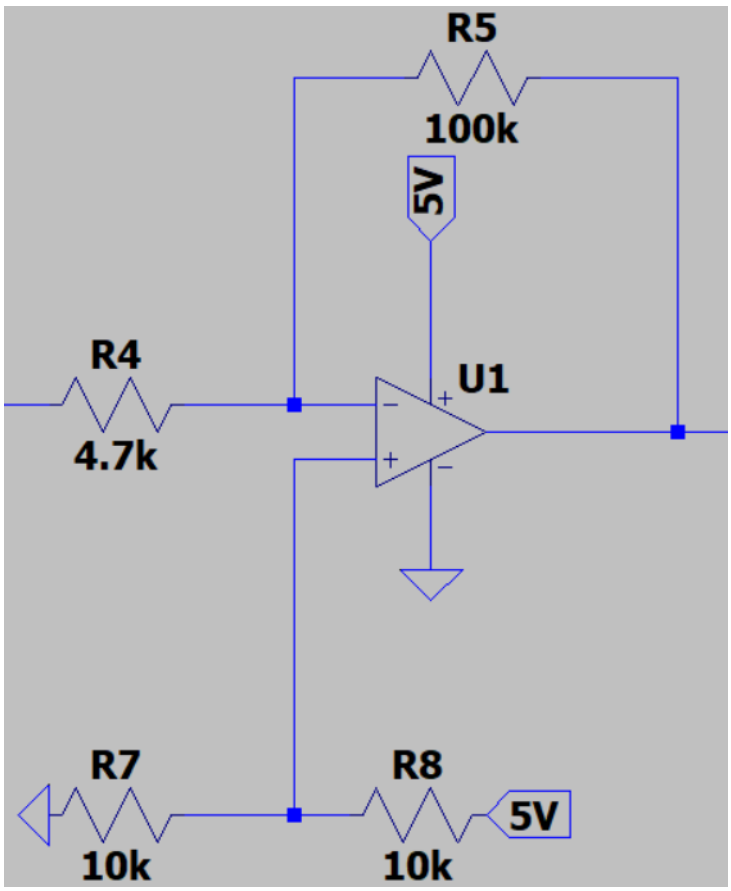

Figure 2.4: Amplifier Circuit

We used an inverting op-amp circuit for amplifying the signal, adding 2.5V DC bias, and inverting the signal to its original state (because the signal was inverted by the electrec mic). The gain of the amplifier can be calculated using the formula:

$$G = -\frac{R_f}{R_1} \tag{2.2}$$

In our case, we used $R_1 = 4.7k\Omega$ and $R_f = 100k\Omega$, so the gain was approximately -20. This amplifier was able to amplify the signal to a usable level. For DC bias, we used a voltage divider ciruit to get 2.5V and connected it to the non-inverting input of the op-amp, which made our signal oscillate around 2.5V.

2.1.4 Final Circuit

After connecting everything together, the final circuit looked like this:

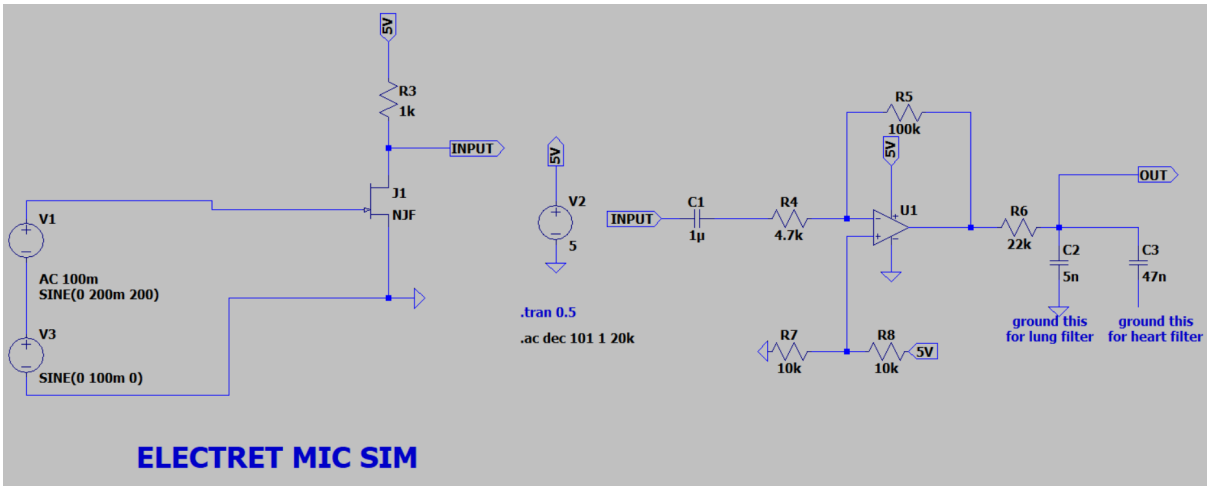

Figure 2.5: Final Circuit

To test the circuit, we used a function generator to simulate the electrec microphone signal. We were able to get the signal from the microphone, amplify it, and filter it successfully. After building the circuit we ran an AC simulation on LTSpice and checked the cutoff frequencies.

	Analytical Cutoff Frequency	**Simulation Result**
High-Pass	33.8Hz	26.7Hz
Low-Pass (Heart)	153Hz	205Hz
Low-Pass (Lung)	1466Hz	1530Hz

Table 2.1: Cutoff Frequencies

According to Debbal (2020), these frequencies are suitable for heartbeat detection and according to Gross et al. (2000), these frequencies are suitable for lung sound detection.

2.2 Prototype Design

2.2.1 Breadboard Prototype

We have built the circuit in Figure 2.5 on a breadboard and measured output voltages using an oscilloscope.

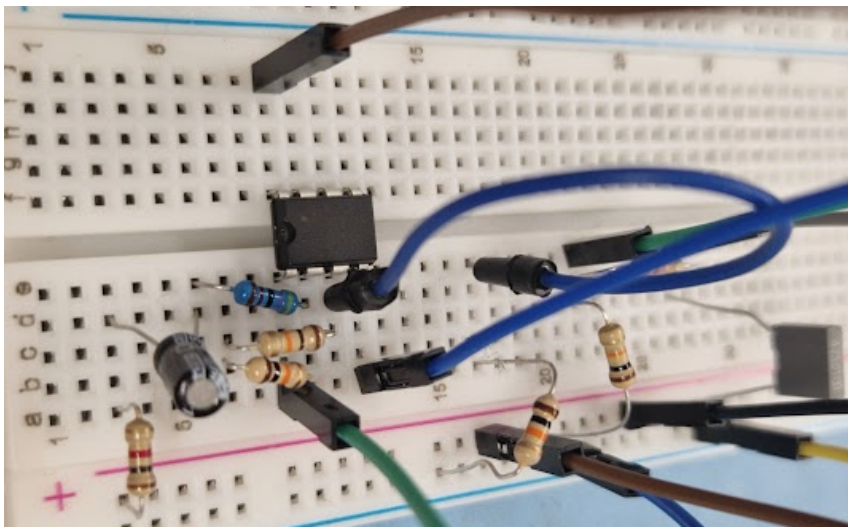

Figure 2.6: Breadboard Prototype

Figure 2.7: Breadboard Prototype Output

Output was exactly as expected. We were able to get the signal from the microphone, amplify it, give 2.5V DC bias, and filter it successfully.

2.2.2 Pertinax Board Design

After making sure that the circuit works as expected, we soldered the circuit parts to perforated pertinax board as shown in Figure 2.8.

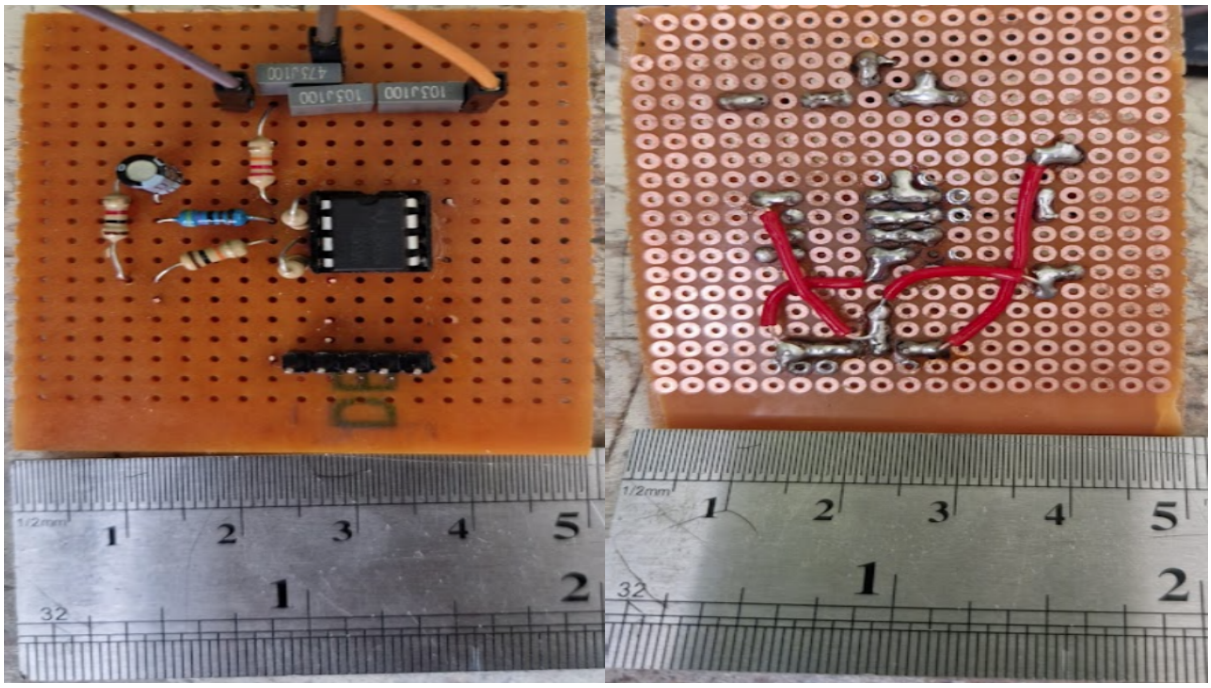

Figure 2.8: Pertinax Board Design

2.2.3 Casing Design

We designed a casing for the prototype using SolidWorks. Casing has required pin headers for microphone, Arduino, and power supply connections.

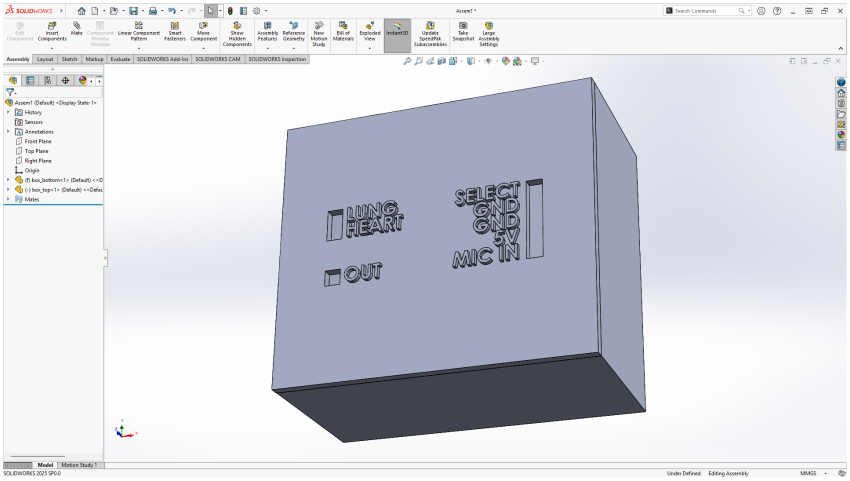

Figure 2.9: Casing Design

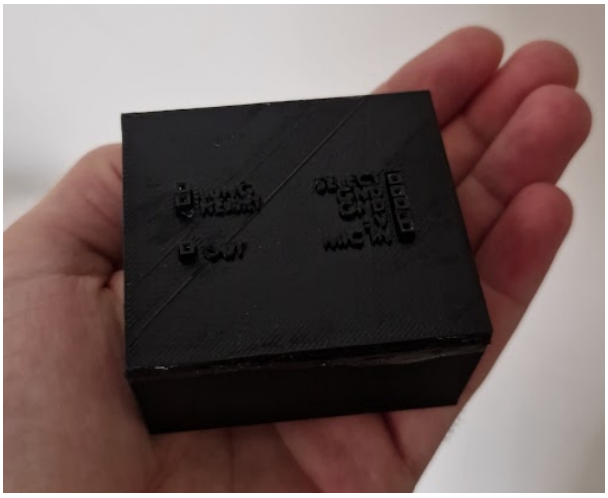

Figure 2.10: Casing Design Printed

Our first prototype is worked as expected so we decided to use it as our final prototype. By putting the prototype in the casing, we completed the hardware development phase.

Chapter 3

Software Development

3.1 Arduino Code

We used an Arduino Uno for signal transmission. It already has its own 10-bit 0-5V ADC for analog signal transmission. We already made the necessary signal processing on the hardware side, our signal oscillates around 2.5V and has a range of 0-5V. We can start to read the signal using Arduino without any problem or any signal loss.

Before starting to measure the analog signal, we measured the reading speed using the following benchmark code in order to decide what sampling rate we should use:

Listing 3.1: Analog Read Benchmark Code

```
void loop() {
    unsigned long start = micros();
    analogRead(A0);
    unsigned long end = micros();
    Serial.println(end - start);
}
```

The code reads the analog signal from pin A0 and prints the time taken to read the signal in microseconds. We used this code to measure the analog signal reading speed of the Arduino Uno. The average time taken to read the analog signal was 112μs. This means that the Arduino Uno can read the analog signal at a rate of 8.9kHz. This is more than enough for our project, as the maximum frequency of the signal we are trying to measure is 2kHz.

According to Wikipedia (2024), Nyquist-Shannon Sampling Theorem says the minimum sampling rate should be twice the maximum frequency of the signal. So, we need a minimum sampling rate of 4kHz, but more is better.

Arduino Uno can read the analog signal at a rate of 8.9kHz, which is more than enough for our project but for stability (because read speed vary around 108μs to 116μs according to our benchmarks), we decided to use a sampling rate of 8kHz (125μs, more than benchmark average). To achieve this, we used the following code:

Listing 3.2: Arduino Code for 8kHz Sampling Rate

```
void loop() {
    unsigned long startedAt = micros();

    // Send the sensor value to the serial port
    int sensorValue = analogRead(analogPin);
    Serial.println(sensorValue);

    unsigned long endedAt = micros();
    unsigned long timeTook = endedAt - startedAt;

    // 125us = 8kHz sampling rate
    unsigned long timeRemaining = 125 - timeTook;

    // Wait for the remaining time
    // this approach will give us 8kHz sampling rate in theory
    // But in practice, it may not be exactly 8kHz
    // but it is good enough for our purposes
    if (timeRemaining > 0) {
        delayMicroseconds(timeRemaining);
    }
}
```

This code reads the analog signal from pin A0, prints the signal value to the serial port, and waits for the remaining time to achieve an 8kHz sampling rate. This code will give us an 8kHz sampling rate in theory, but in practice, it may not be exactly 8kHz. However, it is good enough for our purposes.

While this code prints the analog signal value to the serial port at 8kHz sampling rate, we used a Python script to read the serial port. Sometimes, Python cannot read the serial data correctly because this process is an asynchronous process. Maybe Arduino sends half of the data before Python is ready to read it. To prevent this, we can use starting and ending characters for the serial communication and keep a buffer in Python. We used the following code for this purpose:

Listing 3.3: Arduino Code with Start and End Characters

```
const int analogPin = A0;

// Define the starting and ending characters
// for the serial communication.
// Sometimes python cannot read the serial data incorrectly.
// To prevent this, we can use starting and ending characters.
const String startingChar = "S";
const String endingCar = "E";

void loop() {
    unsigned long startedAt = micros();

    int sensorValue = analogRead(analogPin);

    // Print the value with starting and ending characters
    Serial.print(startingChar);
    Serial.print(sensorValue);
    Serial.println(endingCar);

    // On serial port:
    // SxxxxE -> S: Start, E: End, xxxx: Sensor Value
    // Example
    // S338E -> S: Start, E: End, 338: Sensor Value

    unsigned long endedAt = micros();
    unsigned long timeTook = endedAt - startedAt;

    // 125us = 8kHz sampling rate
    unsigned long timeRemaining = 125 - timeTook;

    // Wait for the remaining time
    // this approach will give us 8kHz sampling rate in theory
    // But in practice, it may not be exactly 8kHz
    // but it is good enough for our purposes
    if (timeRemaining > 0) {
        delayMicroseconds(timeRemaining);
    }
}
```

This code prints the analog signal value to the serial port with starting and ending characters. This way, we can prevent Python from reading the serial data incorrectly.

3.2 Python Code

We used Python to read the serial port, process the analog signal and show the real-time signal on a GUI. As we mentioned in Section 3.1, Arduino Code, we used starting and ending characters for the serial communication. We used these characters to prevent Python from reading the serial data incorrectly. We used the following code to read the serial port:

Listing 3.4: Python Code to Read Serial Port

```
# Read data from serial port. Read Arduino code for more details
data_buffer = ""
while ser.in_waiting > 0:
    try:
        char = ser.read().decode("utf-8")
        data_buffer += char

        if "E" in data_buffer and "S" in data_buffer:
            try:
                start_index = data_buffer.index("S")
                end_index = data_buffer.index("E")
                value_str = data_buffer[start_index + 1 : end_index]
                if value_str.isdigit():
                    analog_value = int(value_str)

                    # Normalize and zero center because
                    # most of the dsp functions
                    # require zero centered data
                    normalized = (
                        (analog_value / 1024.0) * 5.0
                    ) - 2.5

                    # Append the normalized value to the list
                    # So that we can plot it later
                    analog_data.append(normalized)
                    time_data.append(time.time())

                    while time_data and (time_data[-1] - time_data[0] > 5):
                        time_data.popleft()
                        analog_data.popleft()
            except ValueError:
                pass
            data_buffer = ""

    except UnicodeDecodeError:
        data_buffer = ""
    except Exception:
        data_buffer = ""
```

With this code, we read the serial port and process the analog signal. We normalized the analog signal to be zero-centered because most of the digital signal processing functions require zero-centered data. We also kept the last 5 seconds of the data in the list to plot it later.

While these codes are the most important parts of the software development, we also developed a GUI to show the real-time signal which we are not mentioning the code here. The GUI is developed using the tkinter and matplotlib library. You can always check the full code from the GitHub repository of the project, which is given in Appendix 4.

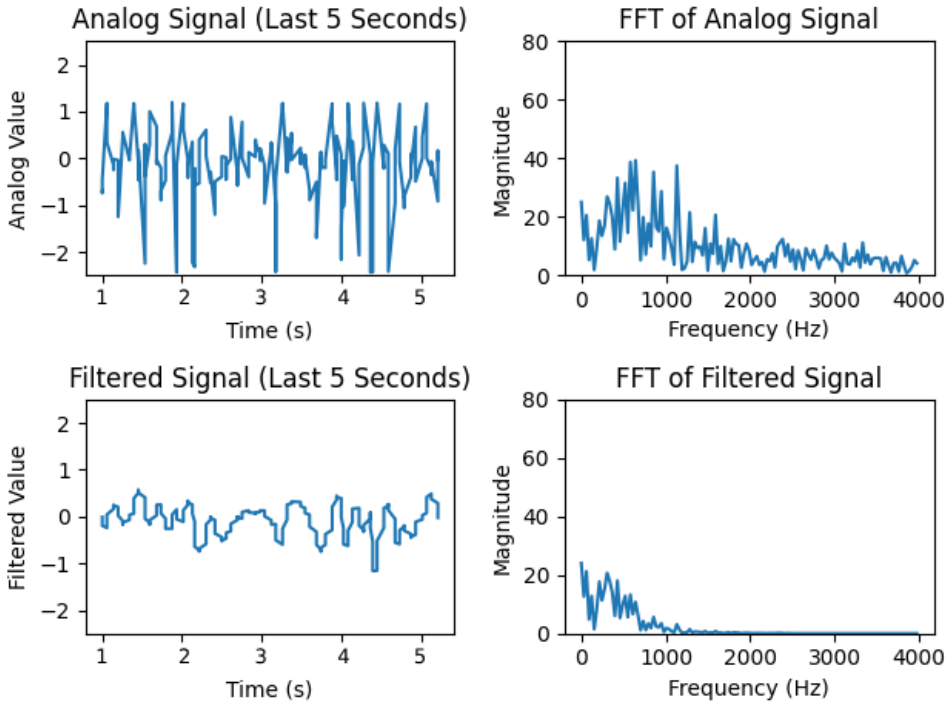

Figure 3.1: GUI of the Project

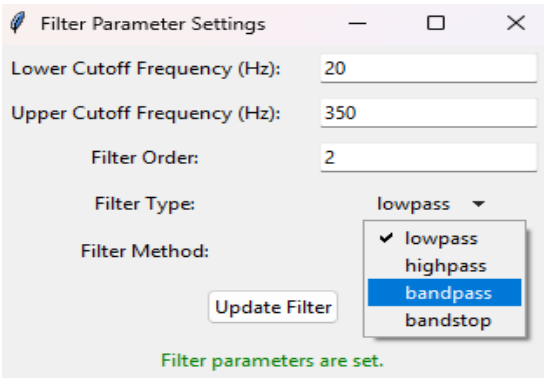

Figure 3.2: Filter Type Selector

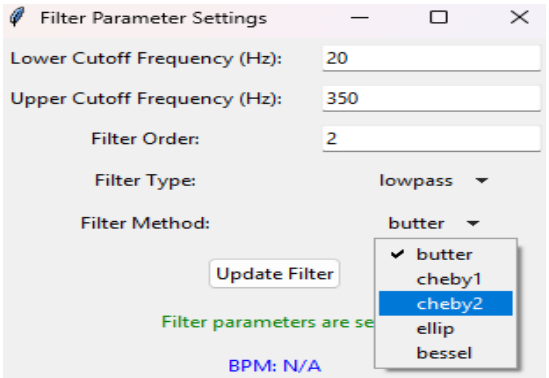

Figure 3.3: Filter Method Selector

Chapter 4

Conclusion

The development of the electronic stethoscope demonstrates the integration of electronics, software, and medical knowledge to create a modern diagnostic tool. With its ability to capture, process, and analyze body sounds, this device provides a valuable addition to healthcare diagnostics. Further improvements in signal processing and hardware miniaturization could make this tool indispensable for clinicians.

This project demonstrated how easy to design a digital stethoscope using off-the-shelf components. The device was able to capture and process body sounds, and the results were displayed on a computer screen. With further development, the device could be used for real-time diagnosis and monitoring of patients.

Appendix

Appendix A: Github Repositoy

https://github.com/dotrocks/sapa-project-electric-stethoscope

Bibliography

Debbal, S. M. E. A. (2020). Analysis of the four heart sounds statistical study and spectro-temporal characteristics. *Clinical Case Reports Journal*, *1*(4), 1–13. https://clinicalcasereportsjournal.com/article/1000046/analysis-of-the-four-heart-sounds-statistical-study-and-spectro-temporal-characteristics

Gross, V., Dittmar, A., Penzel, T., Schüttler, F., & von Wichert, P. (2000). The relationship between normal lung sounds, age, and gender. *American Journal of Respiratory and Critical Care Medicine*, *162*(3), 905–909. https://doi.org/10.1164/ajrccm.162.3.9905104

Wikipedia. (2024). Nyquist-Shannon sampling theorem — Wikipedia, the free encyclopedia [Online; accessed 29-December-2024].