ABDULLAH GÜL ÜNİVERSİTESİ

Department of
Electrical & Electronics Engineering

Abdullah Gül University

Project Report

SAPA Capsule

Submitted on: 02.01.2025

Submitted by: Ayşe Ece Akkurt [redacted]

Group Partner: Ayşe Ece Akkurt
[redacted]

Burak Bekir Çakırer
[redacted]

Ahmet Burak Bilgin
[redacted]

Table of Contents

INTRODUCTION

Over the almost two-century reign, with the coming of digitized technology, this tool began its great advancement: Digital stethoscopes introduced completely new approaches to conventional auscultation and bettered the sound quality that provided accurate detection of physiological normal and abnormal sounds. They basically record acoustic signals of human physiological objects and present them to a clinician for observation of heartbeats and/or lung sounds and the condition of other vital organs.

This project targets the development of an electronic stethoscope system with the ability to capture, amplify, and filter body sounds for perfect diagnostics. The system entails both hardware and software approaches to ensure smooth users' experiences. The developed system is integrated with custom-designed amplification and filtration circuits that yield noise-free sound acquisition, excluding the usage of pre-developed modules hindering original designs and innovations. This uses Arduino or any other similar platforms for the digitization process.

The analysis of the signal is done on a GUI-based MATLAB or Python, which can visualize the signals in real time and change the filter parameters. It provides an interface that allows users to monitor key health metrics, including heart rate and respiration rate, efficiently.

1. OBJECTIVE

The goal of this project was the design and building of an electronic stethoscope that could implement modern technologies into the functions of a traditional one. The device should be able to capture, amplify, and filter body sounds-like heartbeats and lungs' activity-while reducing noise and maintaining signal quality.

Objective The project seeks to design a digital stethoscope circuit that will capture and process auscultatory sounds of high fidelity to provide better diagnoses, rather than those realized from using a traditional stethoscope. Such a digital stethoscope shall be able to do the following:

Record heart and lung sounds using a highly sensitive microphone. Use advanced signal processing methods for eliminating background noise, enhancement, and noise canceling for auscultatory sound clarification. Provide real-time visualization of the waveforms of the sound for better analysis and interpretation. Allow recording and sharing of data for further diagnostic purposes, or even for telemedicine applications. A user-friendly, portable device for medical professionals and healthcare researchers. This project will bridge the gap between old analog auscultation tools and modern digital solutions, enabling the medical practitioners with a more accurate and versatile tool for clinical diagnosis.

2. BACKGROUND

Fourier Transform : The Fourier transform (FT), an integral transform in mathematics, takes a function as input and produces a different function that indicates how much of each frequency is contained in the original function.

Frequency Domain (s-Domain): Frequency domain decomposition is a popular output-only system identification technique in civil engineering, especially in structural health monitoring. Being an output-only algorithm, it is useful when the input data is unknown. FDD is a modal analysis technique that generates a system realization using the frequency response given multi-output data.

LTspice: LTspice is a powerful, fast, and free SPICE simulator software; schematic capture; and waveform viewer, with enhancements and models to improve the simulation of analogue circuits. This graphical schematic capture interface lets you explore schematics and produce simulation results that can be explored further through its built-in waveform viewer.

KCL-KVL: In the lumped-element model of electrical circuits, Kirchhoff's circuit laws are two equivalencies that deal with current and potential difference, also referred to as voltage. In 1845, German scientist Gustav Kirchhoff published the first description of them.[1] This came before James Clerk Maxwell's work and generalized Georg Ohm's work. These are sometimes called Kirchhoff's rules or simply Kirchhoff's laws, and are standard in electrical engineering. The rules are usually important for the nodal analysis and mesh analysis. Both can be used in time domain and frequency domain. In the low-frequency limit both of Kirchhoff's laws can be explained directly from Maxwell's equations.

Fast Fourier Transform: A variety of algorithm which computes DFT or IDFT of a sequence is called a Fast Fourier transform. The Fourier transform is a mathematical procedure of transforming a function of one independent variable into another function of frequency, its inverse.

3.ANALYTICAL AND SIMULATION PROCEDURES

3.1. CIRCUIT

Two different circuits were developed by the team participants to address their audio signal processing needs. The circuit was built using a total of three breadboards. The first breadboard consists of the microphone circuit, the second breadboard consists of the heart and lung filter, and the third breadboard consists of the speaker circuit. The microphone circuit was installed on the first breadboard with a 3.3kΩ resistor and 330nF capacitor. The circuit on the second breadboard includes TL072 op-amps and low-pass and high-pass filters of the heart and lung. On the last breadboard, a speaker circuit was created using TL072 op-amp and transistor.

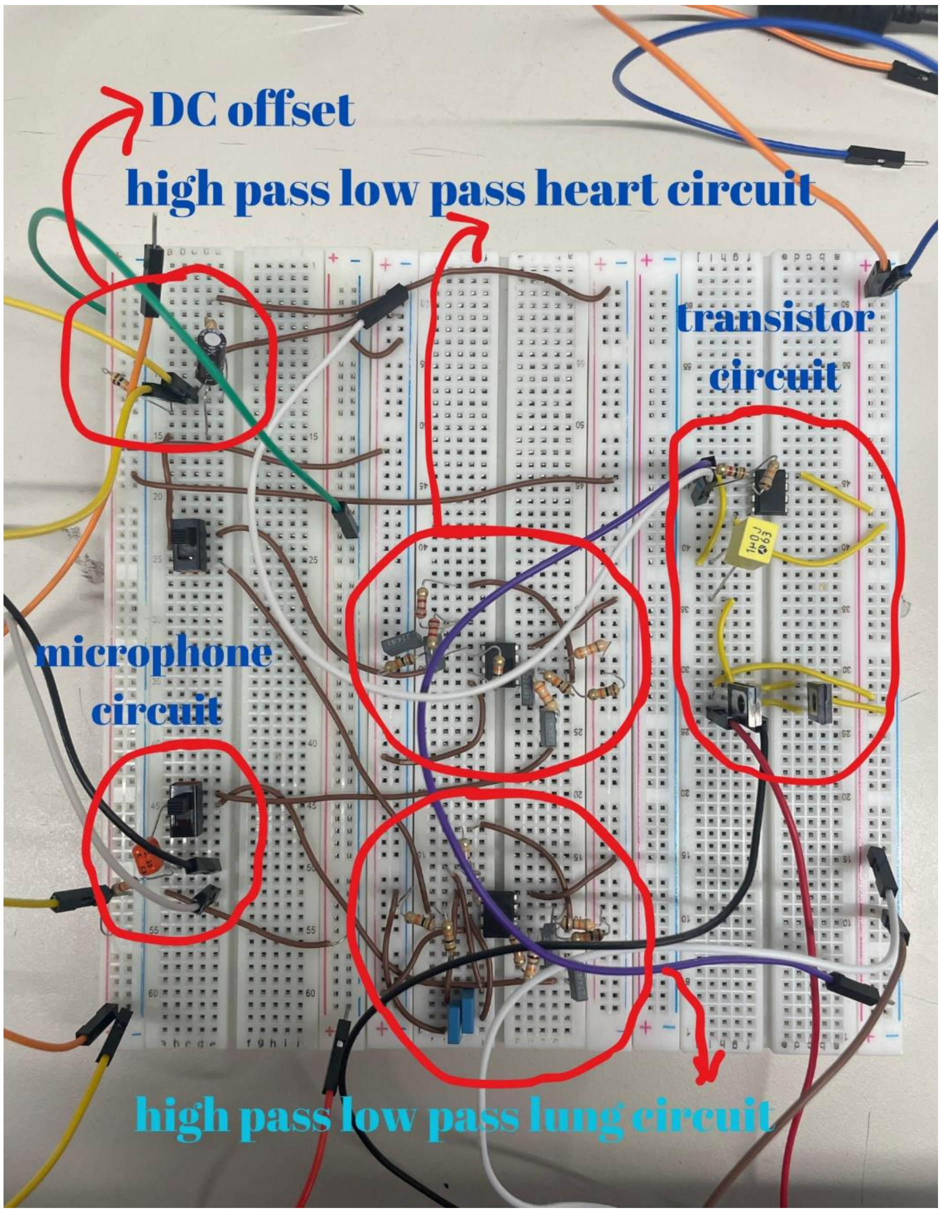

DC offset
high pass low pass heart circuit
transistor circuit
microphone circuit
high pass low pass lung circuit

Figure 1

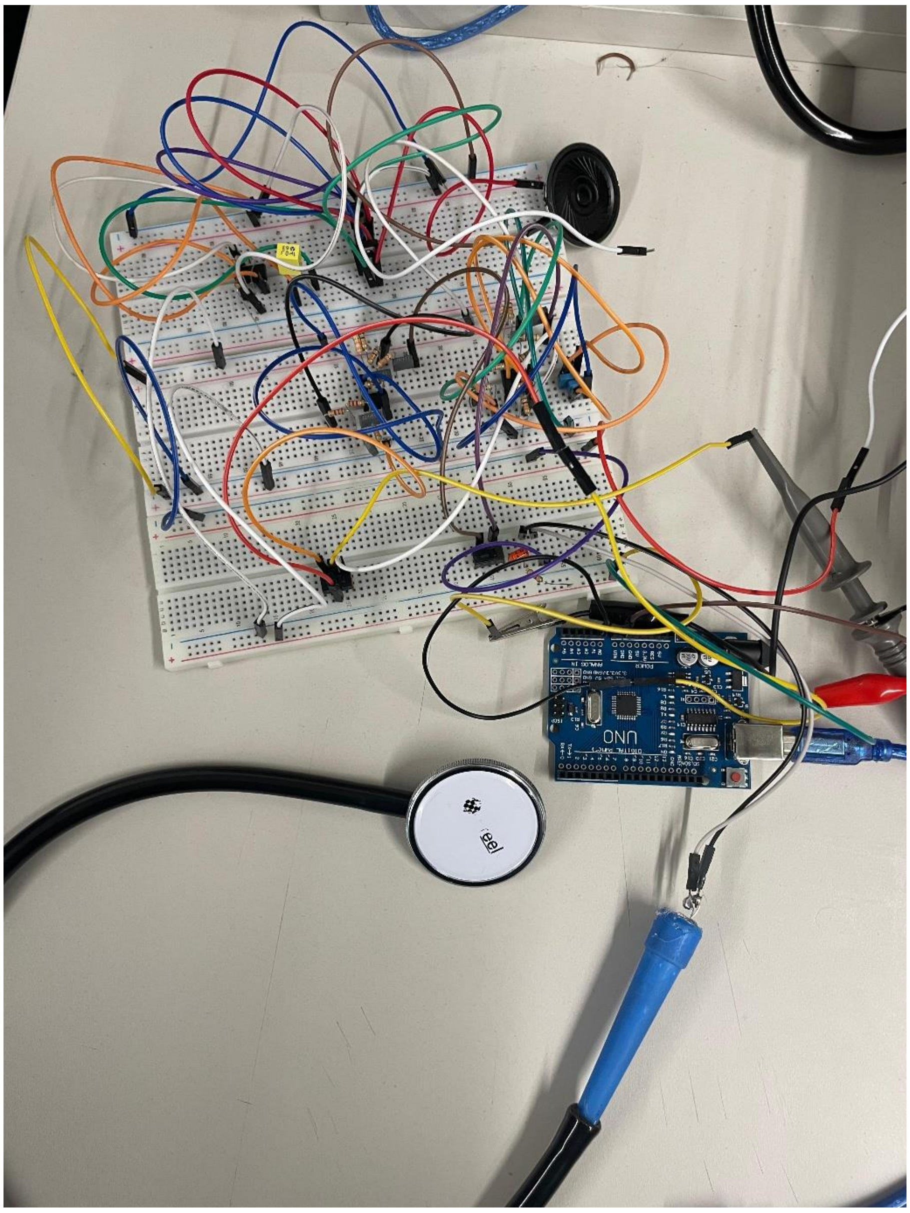


Figure 2

3.2 Filtering Requirement

These filters are realized using specially selected TL072 operational amplifiers because of their very low noise factor and high sensitivity.

Noise can mask the clarity and precision of the data in any signal acquisition system so easily. For example, high-frequency noise, similar to the hum of an electrical appliance, masks the subtlety of the acoustic details like heartbeats, while low-frequency noise, much like machinery rumble, distorts the waveform. Events against these issues use project custom-designed active filters:

Low-Pass Filter: It allows the low-frequency signals, that is, the heartbeats, to pass and attenuate the high-frequency noise. This ensures the critical details in both the cardiac and respiratory signals are preserved.
High-Pass Filter: This filter removes the low-frequency artifacts, which is baseline drifts or motion-induced interference in ECG signals, thus isolating mid-range frequencies, very crucial for signal interpretation.

3.3 Signal Processing and MATLAB

The project is based on the main platform of MATLAB due to its powerful analytical capability and user-friendly interface, while the signal processing and visualization can be done in real time. A Graphical User Interface has been developed for observing and analyzing real-time signals. The main key features provided by GUI are as follows:

Real-time Visualization: The system displays raw and filtered signals, thereby presenting, in real time, the results of filtering to the users.
Dynamic Filter Adjustment: This GUI will enable interactive changes in filter parameters; hence, the system will dynamically adjust to time-varying noise conditions and signal characteristics.
Health Metric Monitoring: It is used for the extraction and display of important metrics, including heart rate and respiration rate.

3.4 Heart Circuit

3.4.1 Active Low-Pass Filter

An active low-pass filter is a circuit arrangement that damps all sounds whose frequencies are higher than the cut-off while allowing those below the cut-off to pass through. Its design actively amplifies and filters using an operational amplifier, or op-amp which is TL072 is on the circuit tested.

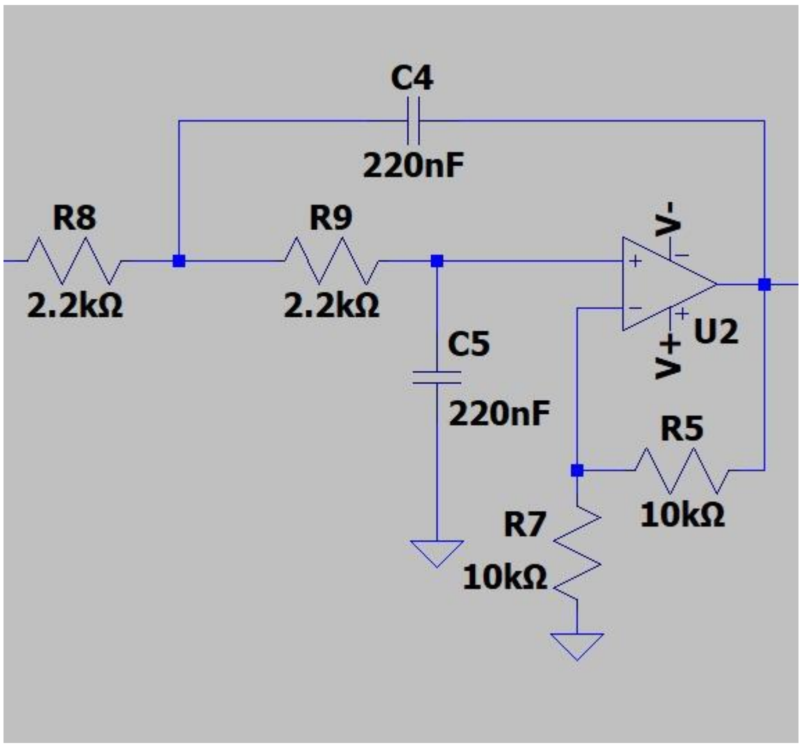


Figure 3

Calculations for Cutoff Frequency :

$$f_C = \frac{1}{2\pi RC}$$

R = 2,2.10^3 ohm

C = 220 nF

$$f_c = \frac{1}{2 \cdot \pi \cdot 2{,}2.10^3 \cdot 220.10^{-9}} = 328{,}83 \text{ hz}$$

3.4.2 Active High-Pass Filter

An active high-pass filter is an electronic circuit that attenuates the lower-frequency components, while allowing signals having frequencies above a certain cut-off frequency to pass through. The normal active components in such a filter are operational amplifiers, commonly known as op-amps, which actively shape and amplify the input signals. The TL072 will be used to provide an operational amplifier for the active low-pass filter part.

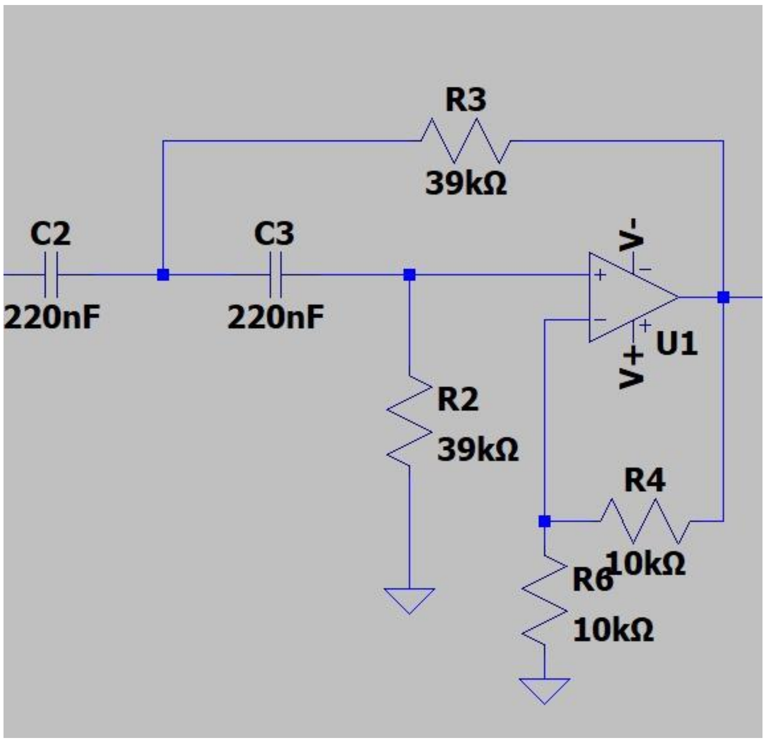


Figure 4

$$f_C = \frac{1}{2\pi RC}$$

R = 39 000 ohm

C = 220 nF

$$f_c = \frac{1}{2 \cdot \pi \cdot 39.10^3 \cdot 220.10^{-9}} = 18{,}54\ hz$$

3.4.3 Amplificator

U3 in the circuit provides the last stage of processing and amplification for the digital stethoscope system, such that the processed signal will be clear, powerful, and useful. The U3 amplifies the low-pass-filtered signal at the input to rid it of distortion and make it suitable for the output system. The U3 also provides impedance matching so as not to lose any signal and also to stabilize it. This process is especially vital in clear hearing and analysis of heart and lung sounds.

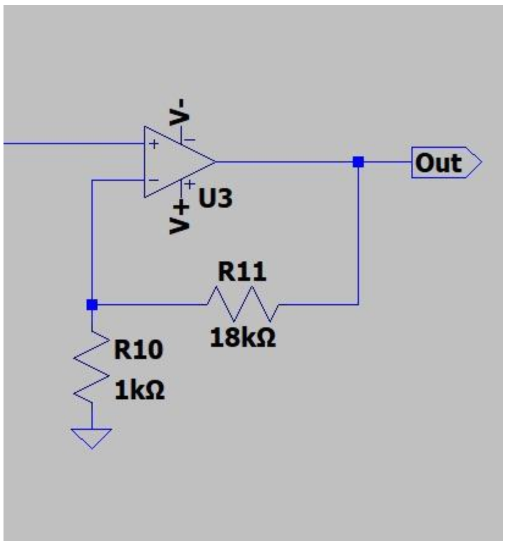


Figure 5

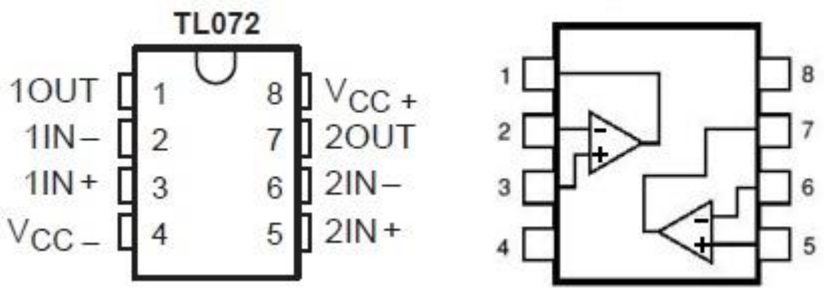


Figure 6

A low-pass filter is one of the basic electronic circuits that allows signals with a frequency lower than the designed cutoff frequency to pass while, at the same time, attenuating those beyond this cutoff frequency. That makes low-pass filters be very vital in removing high-frequency noise or unwanted signal components and ensuring that the desired low-frequency signal remains intact.

In many circuits using active low-pass filters, amplifiers like the one used in this project—a TL072—are often partnered with resistive and capacitive elements. Active designs offer many advantages over passive designs for building a filter, including amplification of the signal as well as improved performance working with low-frequency signals. The cutoff frequency of the filter, which determines the point at which signal attenuation begins, is defined by the circuit's resistance and capacitance values, according to the formula:

$$f_C = \frac{1}{2\pi RC}$$

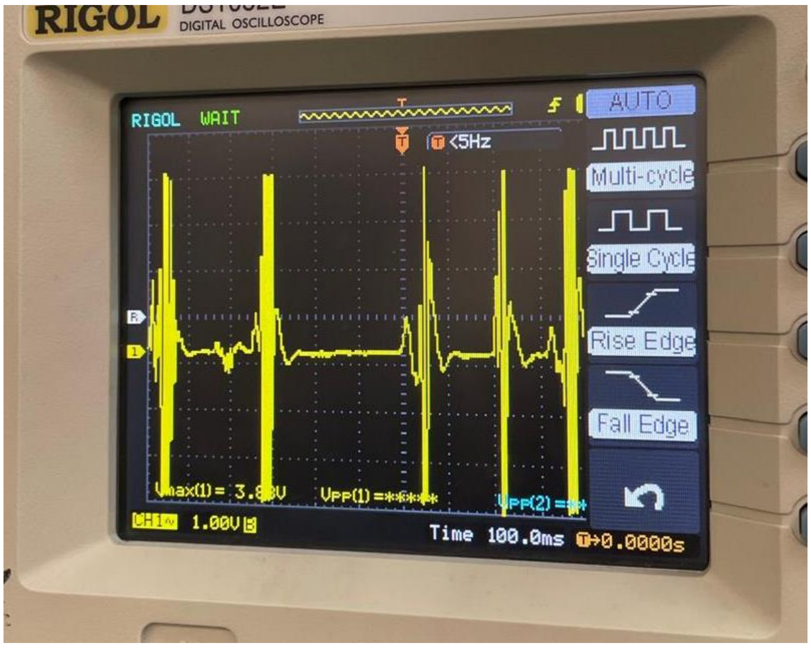
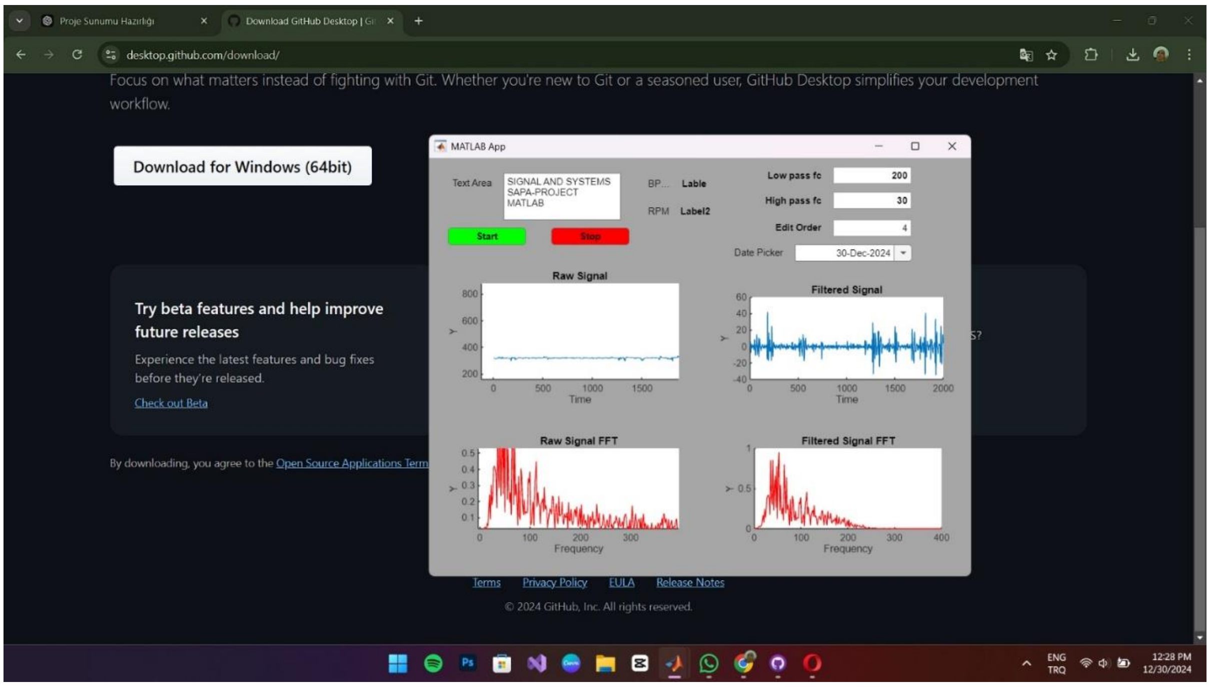


Figure 7

3.4.4 Waveform Analysis

In the cardiac circuit of this study, the design must involve receiving and analyzing the signal emitted from the heart in order to describe its functional state and diagnose various pathologies. Waveform analysis could be initiated by the process of obtaining raw signals from an electronic stethoscope. Generally, these raw signals exhibit low amplitude, replete with noise, and require filtering and amplification using specially designed circuits. Generally speaking, the heartbeat sounds will have a low-frequency range of 50 Hz-300 Hz; the amplitude depends on the strength of the contractions of the heart and the proximity of the stethoscope to the chest wall. A specially designed amplification circuit amplifies weak signals with minimal distortion; a low-pass filter attenuates unwanted high-frequency components, while a high-pass filter rejects ambient noise below the frequency band of interest. These filtered heart sound signals are presented as output on a graphical user interface developed in MATLAB that visually shows the raw and the spectrum of the raw and filtered signal while also giving the instantaneous heart rate obtained from periodicity in the waveform. It reflects a very good waveform for noise suppression with much smaller environmental and electronic noises for better diagnosis. This sensor is sensitive to each immediate change in the heart rate and the characteristics of the sound in real time; thus, it allows for dynamic monitoring. Waveform analysis reflects efficiency in cardiac circuitry that converts raw acoustic signals into usable diagnostic information, as clean and accurate waveforms are the foundation for reliable cardiac assessments.

3.4.5 MATLAB® GUI Signal Acquisitions


Focus on what matters instead of fighting with Git. Whether you're new to Git or a seasoned user, GitHub Desktop simplifies your development workflow.

Try beta features and help improve future releases

Experience the latest features and bug fixes before they're released.

By downloading, you agree to the Open Source Applications Term

© 2024 GitHub, Inc. All rights reserved.

Text Area: SIGNAL AND SYSTEMS SAPA-PROJECT MATLAB

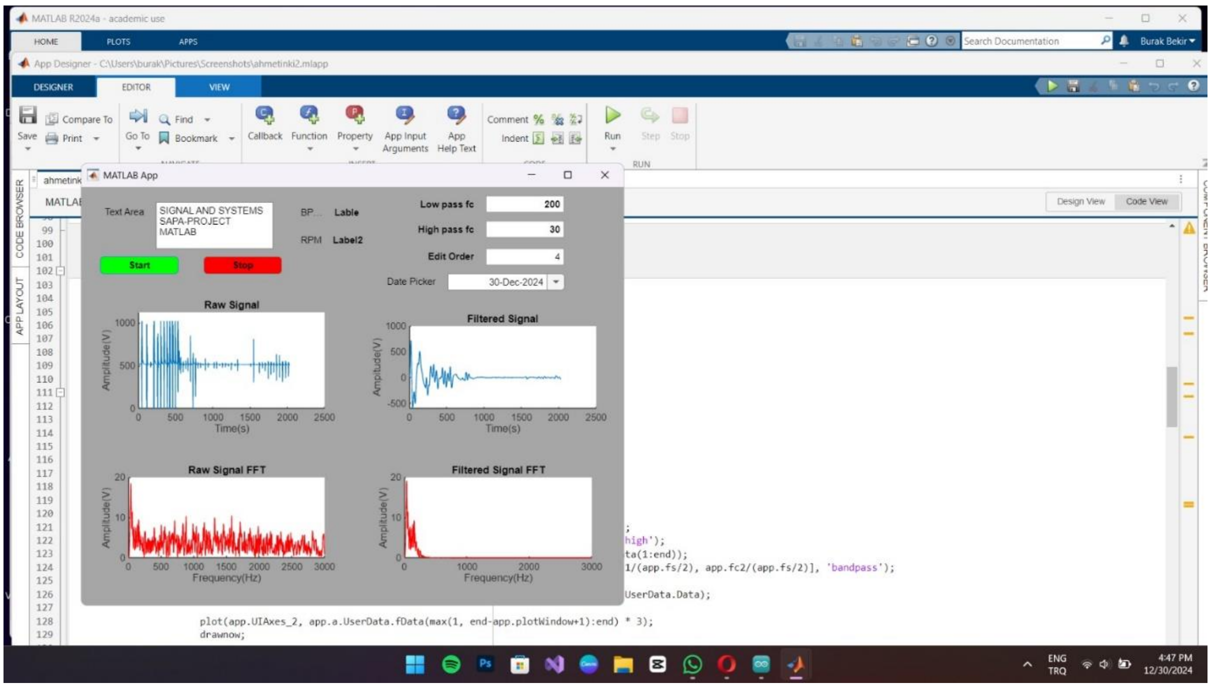
Low pass fc: 200

High pass fc: 30

Edit Order: 4

Date Picker: 30-Dec-2024

Figure 8


Text Area: SIGNAL AND SYSTEMS SAPA-PROJECT MATLAB

Low pass fc: 200

High pass fc: 30

Edit Order: 4

Date Picker: 30-Dec-2024

```
;
high');
ta(1:end));
1/(app.fs/2), app.fc2/(app.fs/2)], 'bandpass');

UserData.Data);

    plot(app.UIAxes_2, app.a.UserData.fData(max(1, end-app.plotWindow+1):end) * 3);
    drawnow;
```

Figure 9

The MATLAB® GUI application is made to collect and analyze respiratory and cardiac signals. Both the raw and filtered signal domains show clear visual differences between healthy and ill individuals. The raw signals show anomalies and inconsistencies in sick individuals, whereas the filtered signals make pertinent features more clear. Furthermore, the signals' Fast

Fourier Transform (FFT) plots show significant variations in the frequency domain for both healthy and sick people. FFT thus offers important insights into the variances linked to various medical problems. The effectiveness of our MATLAB GUI application in differentiating between sick and healthy situations is highlighted by this thorough investigation, which also indicates potential signals as markers of health condition.

3.4.6 LTspice

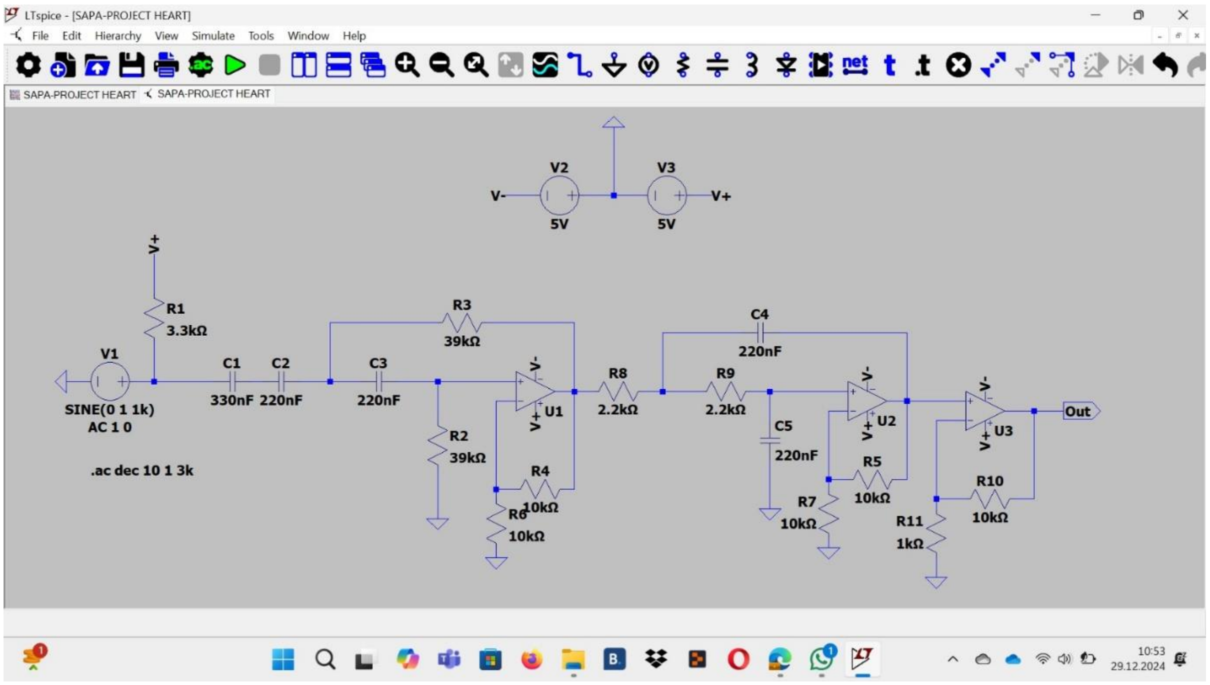
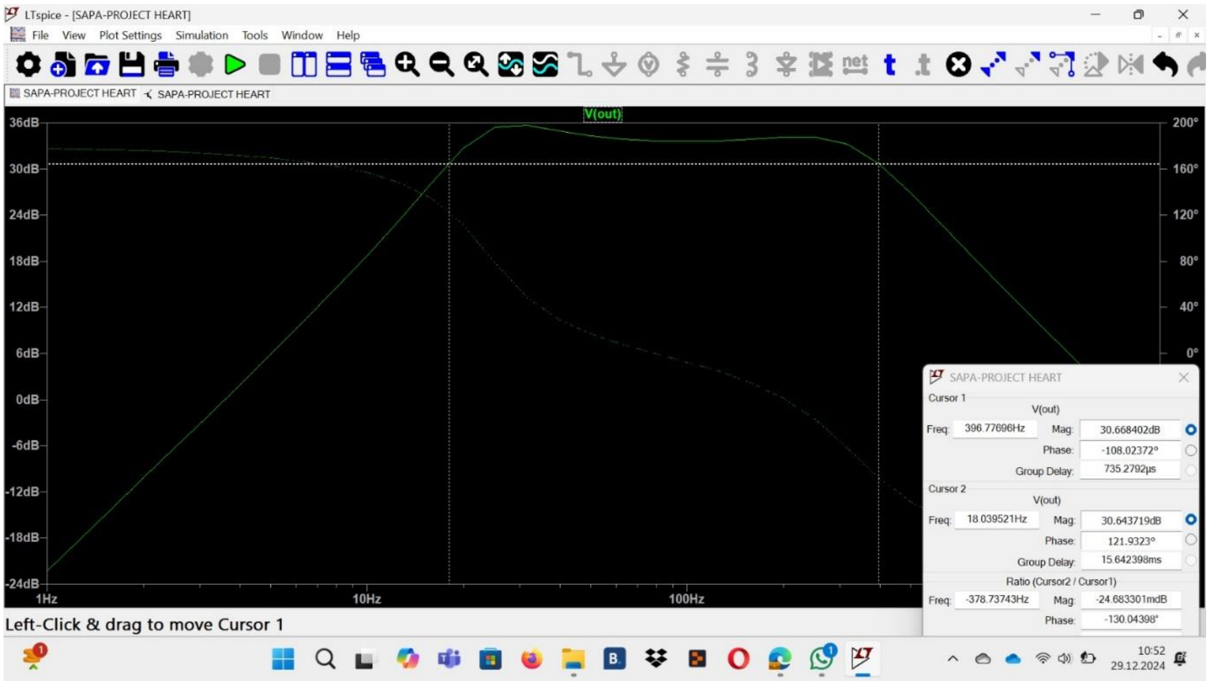


Figure 10



SAPA-PROJECT HEART

Cursor 1

V(out)

Freq: 396.77696Hz Mag: 30.668402dB

Phase: -108.02372°

Group Delay: 735.2792µs

Cursor 2

V(out)

Freq: 18.039521Hz Mag: 30.643719dB

Phase: 121.9323°

Group Delay: 15.642398ms

Ratio (Cursor2 / Cursor1)

Freq: -378.73743Hz Mag: -24.683301mdB

Phase: -130.04398°

Left-Click & drag to move Cursor 1

Figure 11

The half-power frequency (or -3 dB point) typically refers to the frequency at which the power of a signal drops to half its original value. This 3 dB drop is related to halving the power on a logarithmic scale. The reasoning behind this rule can be explained mathematically as follows:

1. **Power and Voltage Relationship**: In electronic circuits, the relationship between power (P) and voltage (V) is given by:

$$P \propto V^2$$

- That is, power is proportional to the square of the voltage.

2. **Logarithmic Scale and Decibels**: Power levels are often expressed in decibels (dB). The dB value of power is calculated as:

$$L = 10 \cdot \log_{10}\left(\frac{P_0}{P}\right)$$

If the power is halved (P=P0/2):

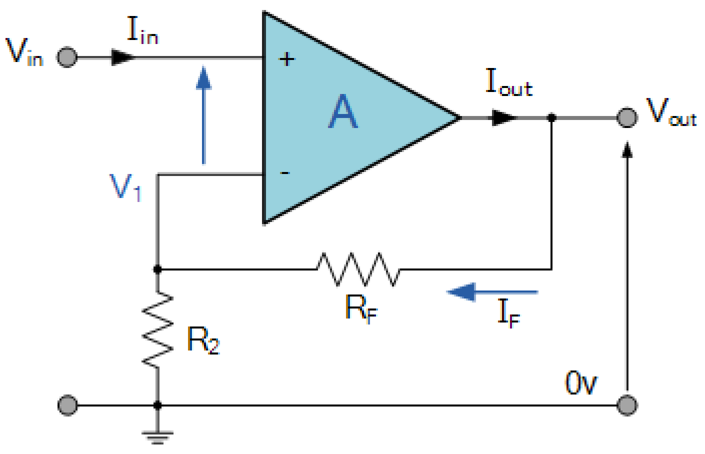
$$L = 10 \cdot \log_{10}\left(\frac{P_0}{P_0/2}\right)$$

$$L = 10 \cdot \log_{10}(2) \approx 10 \cdot (-0.3010) = -3.010\,\mathrm{dB}$$

This is approximately a -3 dB loss.

3. **Voltage Change**: In terms of voltage, a halving of the power corresponds to the voltage being divided by $\sqrt{2}$. In other words, when the voltage is reduced by a factor of $\sqrt{2}$, the power drops by -3 dB.

As a result, a -3 dB drop corresponds to the power being halved, and mathematically this defines the **half-power frequency**.



$$A_{(v)} = 1 + \frac{R_F}{R_2}$$

Figure 10

$$G_{U_1} = \frac{10}{10} + 1 = 2$$

$$G_{U_1} = 2$$

$$G_{u_2} = 1 + \frac{10}{10}$$

$$G_{U_2} = 2$$

$$G_{U_3} = \frac{10}{1} + 1$$

$$G_{U_3} = 11$$

$$G = G_{U_1} \cdot G_{U_2} \cdot G_{u_3}$$

$$G = 44$$

3.5 Lung Circuit

3.5.1 Active High-Pass Filter

An active high-pass filter is designed to pass high-frequency signals while low frequencies are attenuated. In the case of a digital stethoscope for lung sound analysis, this filter will be used to isolate the high-frequency components of the lung sounds, usually from 200 Hz to 2000 Hz. This active high-pass filter will remove the low-frequency noise so that the lung sounds can be distinctly and accurately picked up.

$$f_C = \frac{1}{2\pi RC}$$

R = 15 000 ohm

C = 100 nF

$f_C = \frac{1}{2 \cdot \pi \cdot 15.10^3 \cdot 100.10^{-9}}$ =106.103 hz

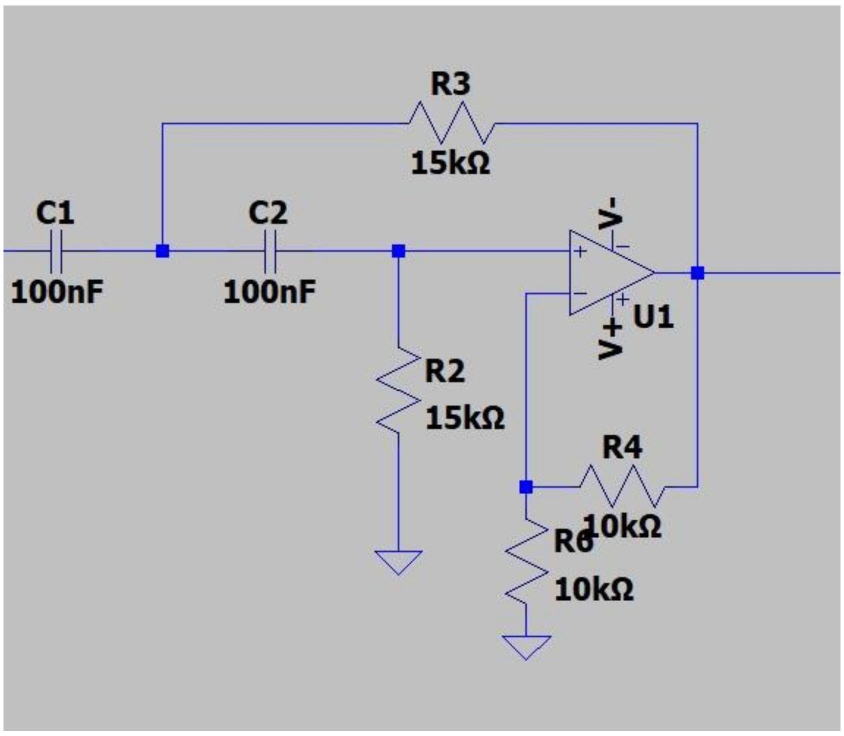


Figure 12

3.5.2 Active Low-Pass Filter

The active low-pass filter is crucial in the filtering out of high-frequency noise while preserving the low-frequency signals necessary for diagnostics in lung sounds. The filter design uses a single-pole RC configuration where the cut-off frequency is determined by the resistor and capacitor values. This cut-off frequency is carefully selected to ensure that only the desired frequency range is passed while attenuating noise. Also, there is an operational amplifier inside the active filter configuration in order to amplify the weak sound of lungs to be analysed.

Two switches are connected to the circuit. One was placed between the heart circuit and the lung circuit and the other between the microphone circuit and the filters. Thus, with a button, switching from heart to lung, from lung to heart was provided.

$$f_C = \frac{1}{2\pi RC}$$

R = 80 000 ohm

C = 1 nF

$f_c = \frac{1}{2 \cdot \pi \cdot 80.10^3 \cdot 1.10^{-9}}$ =1989.43 hz

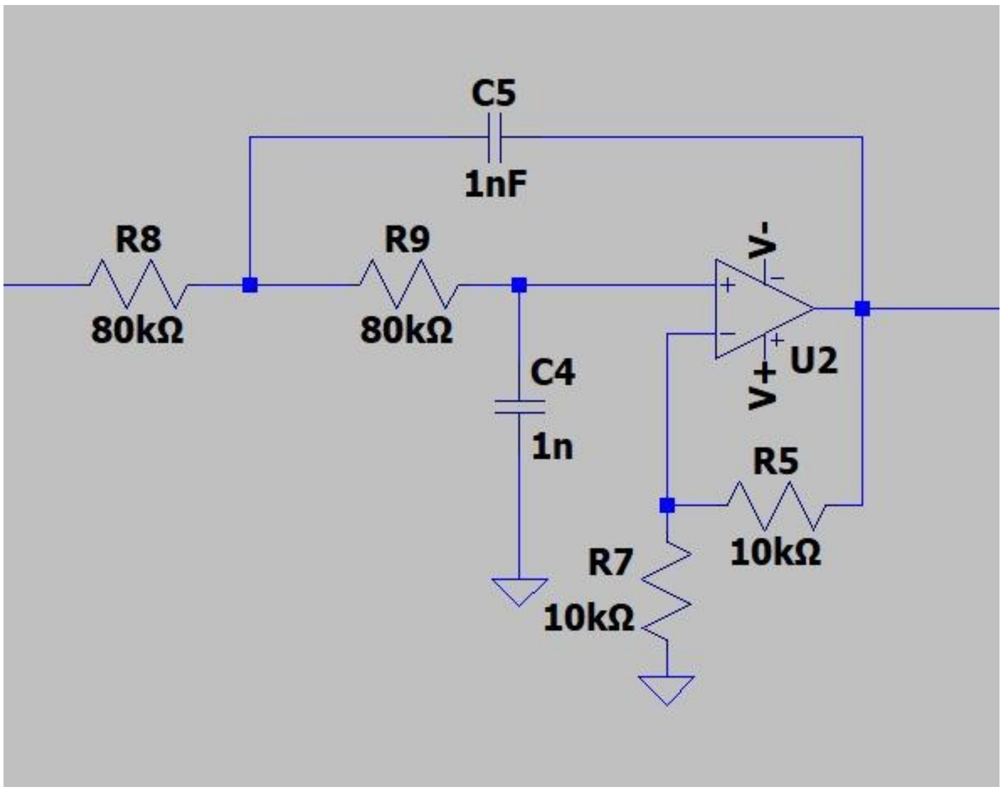


Figure 13

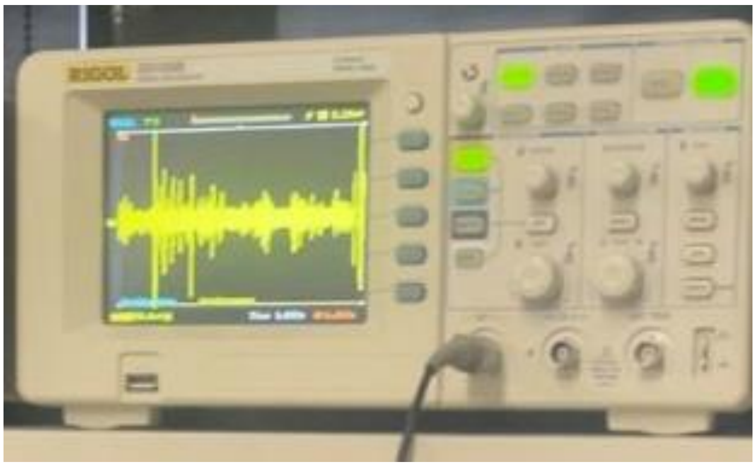
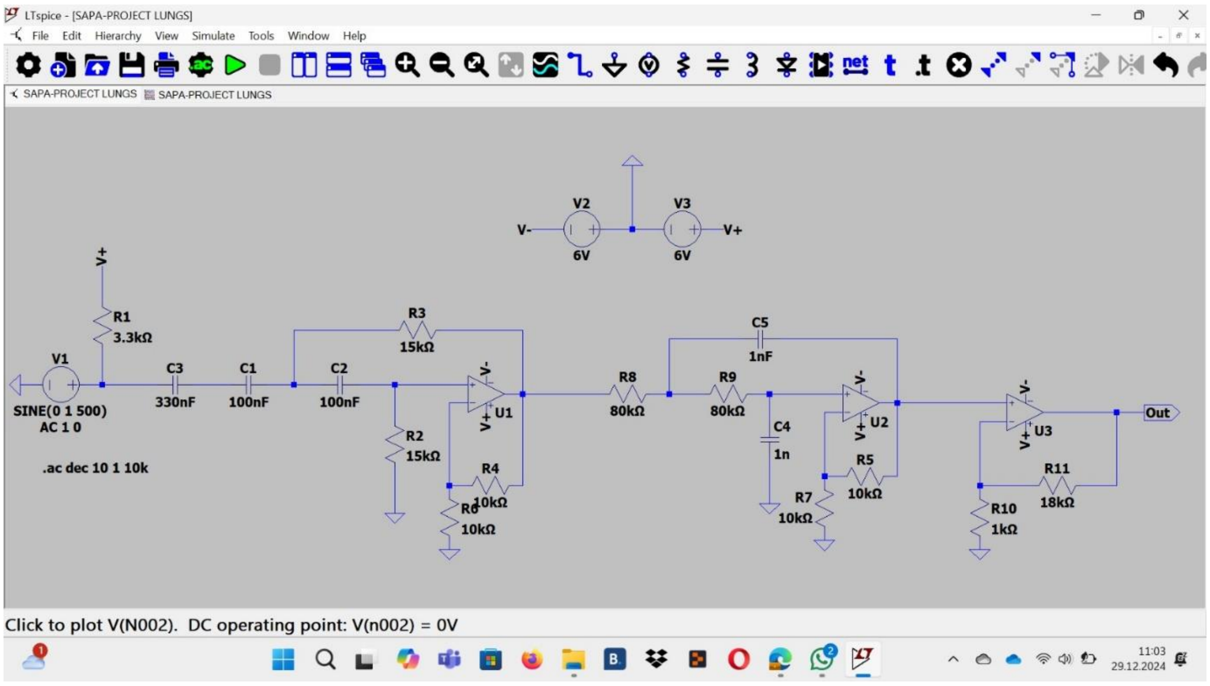


Figure 14

3.5.3 Waveform Analysis

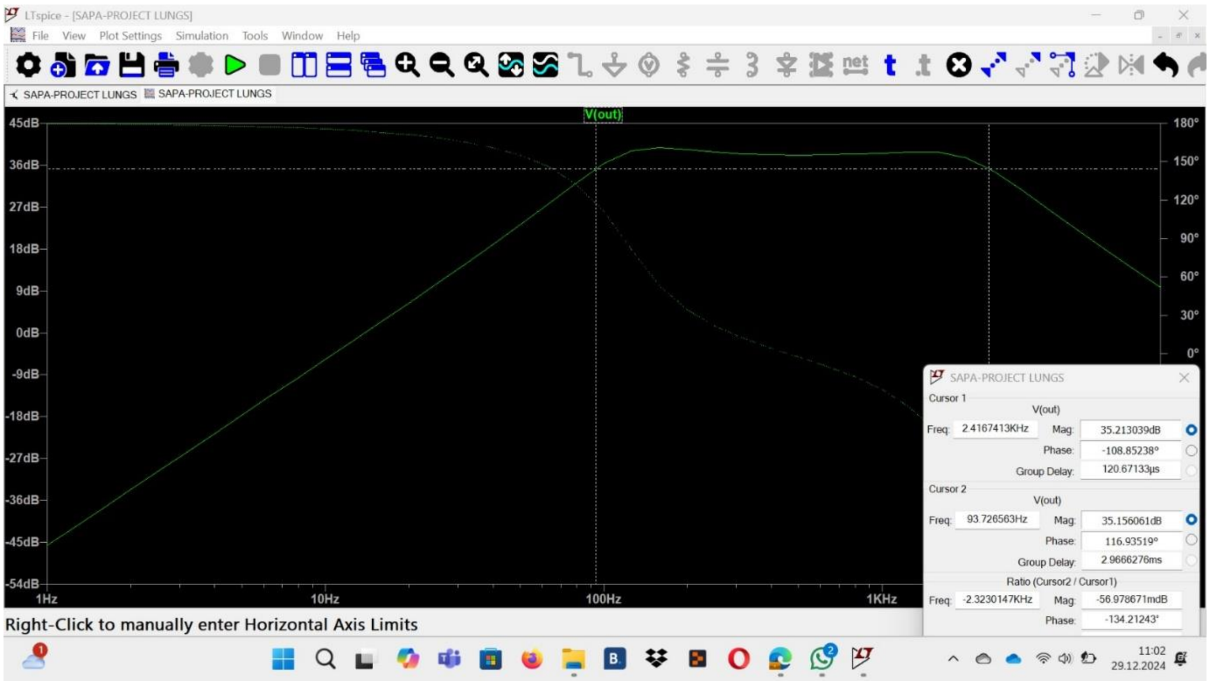
Waveform analysis is examining and interpreting the visual depiction of a wave or signal over time. Waveform analysis in relation to the capture of lung acoustic signals focuses on examining trends in the signals that indicate respiratory activity. Waveform analysis is crucial for lung signal collection since it might gives important information on a person's respiratory health and breathing habits. By going over the lung signal waveform's shape and properties, medical professionals Information on variables including inspiration and expiration periods, tidal volume, and respiratory rate and respiratory dynamics in general. With this information, lung function will be evaluated, detected, and respiratory conditions and keeping an eye on how well interventions or therapies are working.

3.5.4 LTspice



Click to plot V(N002). DC operating point: V(n002) = 0V

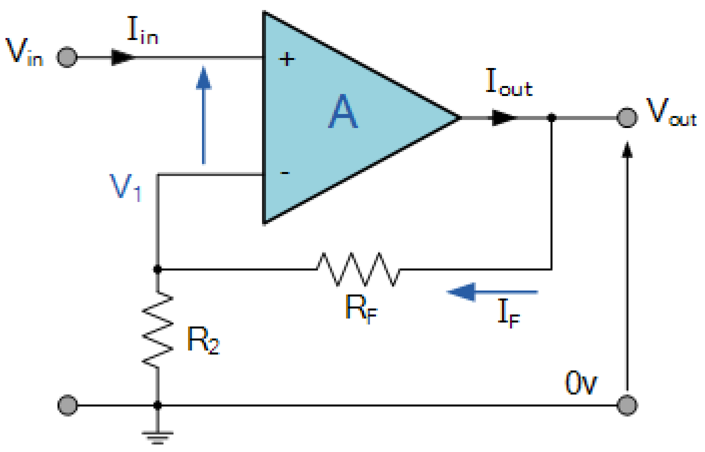
Figure 15



```
SAPA-PROJECT LUNGS
Cursor 1
                    V(out)
Freq: 2.4167413KHz    Mag:         35.213039dB
                      Phase:       -108.85238°
                      Group Delay: 120.67133µs
Cursor 2
                    V(out)
Freq: 93.726563Hz     Mag:         35.156061dB
                      Phase:       116.93519°
                      Group Delay: 2.9666276ms
            Ratio (Cursor2 / Cursor1)
Freq: -2.3230147KHz   Mag:         -56.978671mdB
                      Phase:       -134.21243°
```

Right-Click to manually enter Horizontal Axis Limits

Figure 16



$$A_{(v)} = 1 + \frac{R_F}{R_2}$$

Figure 17

$$G_{U_1} = \frac{10}{10} + 1 = 2$$

$$G_{U_1} = 2$$

$$G_{u_2} = 1 + \frac{10}{10}$$

$$G_{U_2} = 2$$

$$G_{U_3} = \frac{18}{1} + 1$$

$$G_{U_3} = 19$$

$$G = G_{U_1} \cdot G_{U_2} \cdot G_{u_3}$$

$$G = 76$$

3.5.5 MATLAB® GUI Signal Acquisitions

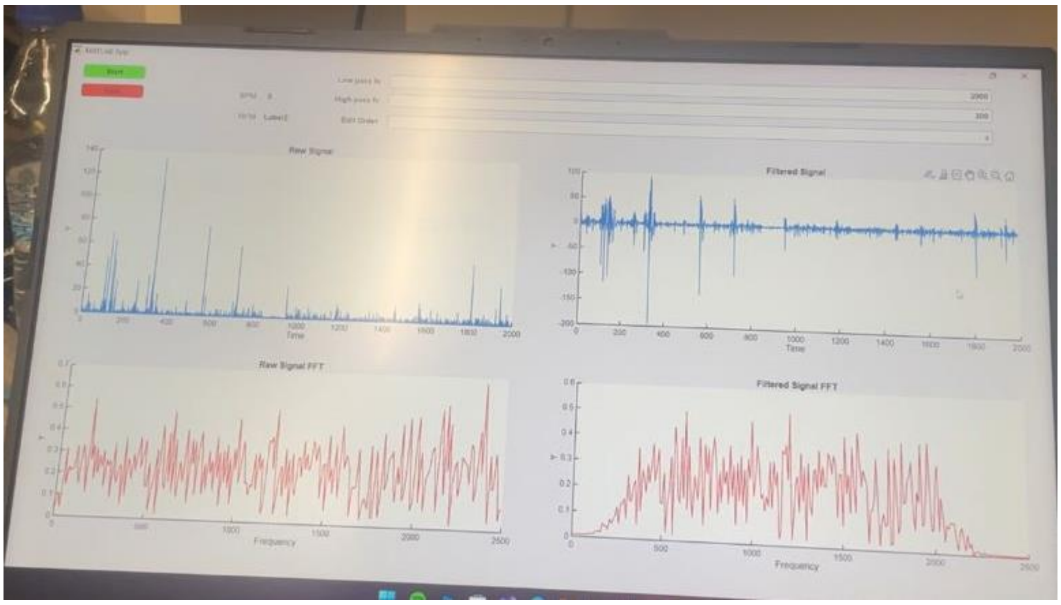


Figure 18

This section's GUI MATLAB® application is developed for collecting and analyzing lung signals based on predefined cutoff frequencies by the user. Furthermore, this application will also facilitate different graphical outputs in terms of comparison of sick versus healthy for both raw and filtered signals. Since the noise reduction increases clarity in the filtered signals, it highlights the health-related abnormalities more conspicuously. Besides this, FFT analysis shows other frequency characteristics that could be important for a deep understanding of changes in breathing patterns related to specific health conditions. This is where the results show the capability of the developed GUI application in MATLAB for effective study and identification of respiratory problems. The main purpose is to make the heart signal clear and sharp in the frequency range of 200 Hz to 2500 Hz to further assist in the analysis of respiration health for efficient detection of the indication of health status.

3.6. MATLAB® GRAPHICAL USER INTERFACE DESIGN

Digital Stethoscope Interface Description

This digital stethoscope interface is developed to process and analyze heart and lung sounds or any other biological signal in real time. This interface is created using MATLAB App Designer to provide a user-friendly atmosphere with the following important features:

Control Buttons:

Start: Triggers the acquisition and processing of the signal.

Stop: Stops the acquisition process.

Input Parameters:
Low-Pass Cutoff Frequency: The frequency above which the low-pass filter should cut off the signal. High-Pass Cutoff Frequency: Allows users to specify the lower frequency limit for the high-pass filter. Filter Order: Users can specify the order of the digital filter in order to adjust the precision of filtering.

Visualization Panels:
Raw Signal: This is a presentation of the unfiltered biological signal in the time domain.
Filtered Signal: This plots the signal that has been processed by filters to bring into view that part of the frequency range the signal has targeted.
Raw Signal FFT: It shows, in its frequency spectrum, for noise analysis, studying signal characteristics.
Filtered Signal FFT: Visualizes the frequency spectrum of the filtered signal, considering relevant biological signal frequencies.

More Features:
The Date Picker will let the user log data with selected dates. Adjustable Parameters: The feature lets this interface be flexible; therefore, it is capable of dynamically changing filter settings to optimize processing of the signal. A good platform is provided, filtering, amplifying and analyzing real time signals may be done on various biomedical research and diagnostic applications.

MATLAB emerged as the tool of choice for project members because to its strong capabilities in signal processing and GUI app building. Project participants effectively filtered signals using MATLAB, revealing their FFT properties and enhancing visibility by the establishment of low and high cutoff points. The ability to create user-friendly interfaces was made possible by MATLAB's App Designer's crucial flexibility. MATLAB App Designer's wealth of callback and property choices revolutionized the development process, simplifying dynamic interactions with the signals being analyzed and guaranteeing a flawless user experience.

Moreover, team members added an Arduino into the circuit system to allow the analog readout of voltage readings. Data could be easily sent to MATLAB because Arduino is a very reliable platform in terms of retrieving raw signal values. This incorporation not only expanded the range of circuit data collecting capabilities while simultaneously increasing the circuit's overall signal processing efficiency. The MATLAB and Arduino partnership in this project put into focus the team's holistic approach to integrating strengths of the two programs for a more complex and insightful analysis of respiratory signals. Certain key GUI components are described and summarized below, with an active link to GitHub provided at the end for accessing the entire GUI for more clarity. The demonstration of FFT functions well is at the end.

4. MATLAB and ARDUINO CODE PART

a.)ARDUINO CODE PART

```
const int analogPin = A0;        // Analog input pin

const int samplingFreq = 6000;   // Sampling frequency in Hz

const int delayTimeMicroseconds = 1000000 / samplingFreq;  // Delay time in microseconds
based on the sampling frequency



void setup() {

    Serial.begin(9600);  // Initialize serial communication at 9600 baud rate

}



void loop() {

    int sensorValue = analogRead(analogPin);  // Read the analog signal from pin A0

    Serial.println(sensorValue);             // Print the read value to the serial monitor

    delayMicroseconds(delayTimeMicroseconds); // Wait for the next sample based on the
sampling frequency

}
```

b.)MATLAB CODE PART

```
classdef project < matlab.apps.AppBase


   % Public properties for UI components
   properties (Access = public)
        UIFigure                 matlab.ui.Figure   % Main application window
        TextArea                 matlab.ui.control.TextArea   % Text area for additional information
        TextAreaLabel            matlab.ui.control.Label  % Label for text area
        DatePicker              matlab.ui.control.DatePicker   % Date picker UI
        DatePickerLabel          matlab.ui.control.Label   % Label for date picker
        Label2                  matlab.ui.control.Label  % Placeholder label
        RPMLabel                 matlab.ui.control.Label  % Label for RPM display
        BPMLabel                 matlab.ui.control.Label  % Label for BPM display
        LableLabel              matlab.ui.control.Label  % Placeholder label
        EditOrderEditField       matlab.ui.control.NumericEditField  % Filter order edit field
        EditOrderEditFieldLabel   matlab.ui.control.Label   % Label for filter order
        LowpassfcEditField       matlab.ui.control.NumericEditField   % Low-pass cutoff
frequency field
        LowpassfcEditFieldLabel   matlab.ui.control.Label   % Label for low-pass frequency
        HighpassfcEditField      matlab.ui.control.NumericEditField   % High-pass cutoff
frequency field
        HighpassfcEditFieldLabel  matlab.ui.control.Label   % Label for high-pass frequency
        StopButton              matlab.ui.control.Button  % Button to stop data acquisition
        StartButton             matlab.ui.control.Button  % Button to start data acquisition
        UIAxes_4                matlab.ui.control.UIAxes  % Axes for FFT of filtered signal
        UIAxes_3                matlab.ui.control.UIAxes  % Axes for FFT of raw signal
        UIAxes_2                matlab.ui.control.UIAxes  % Axes for filtered signal
        UIAxes                  matlab.ui.control.UIAxes  % Axes for raw signal
    end

    % Private properties for internal logic
    properties (Access = private)
        a  % Serial port object
        f  % Frequency array for FFT (raw signal)
        f1 % Frequency array for FFT (filtered signal)
        Y  % FFT of raw signal
        Y1 % FFT of filtered signal
        P1 % Processed FFT data for raw signal
        P2 % Magnitude data for raw signal
        P3 % Processed FFT data for filtered signal
        P4 % Magnitude data for filtered signal
        filteredPlot % Filtered signal plot data
        rawPlot      % Raw signal plot data
        serialport   % Serial port object for data acquisition

        fs = 6000;  % Sampling frequency
        fc1 = 30;   % High-pass filter cutoff frequency
        fc2 = 200;  % Low-pass filter cutoff frequency
        T = 1/5000; % Sampling period
        n = 1;      % Counter for managing data processing
        m = 1;      % Counter for FFT updates
        m1 = 1;     % Counter for additional data processes
        plotWindow = 6000; % Data points displayed in the plot
        IsRunning = false; % Boolean flag for running state
        Timer % Timer object for periodic tasks
        Data % Raw data buffer
        ffData % Filtered data buffer (high-pass)
        fdata % Filtered data buffer (bandpass)
        coefficient_1 % High-pass filter coefficients
        coefficient_2 % High-pass filter coefficients
        fc3=30  % High-pass filter cutoff frequency (duplicate, could be optimized)
        bpm % Beats per minute (calculated)
        coefficient_3 % Band-pass filter coefficients
        coefficient_4 % Band-pass filter coefficients
        orderOfFilter=4; % Default filter order
    end
```

```
    % Private methods for internal functionality
    methods (Access = private)

        % Timer callback function to stop the timer if the application is not running
        function timerCallback = func(app)
            if ~app.IsRunning
                stop(app.Timer);
            end
        end

        % Function to calculate BPM (beats per minute)
        function bpmCalculation(app)
            fs = app.fs;  % Sampling frequency

            % Detect peaks in the signal
            [pks, locs] = findpeaks(signal, 'MinPeakHeight', max(signal) * 0.6,
'MinPeakDistance', 50);

            % Calculate BPM if there are multiple peaks
            if length(locs) > 1
                periods = diff(locs) / fs;  % Time intervals between peaks
                bpm = 60 / mean(periods);  % Convert to beats per minute
            else
                bpm = 0; % Default to 0 if not enough peaks are found
            end

            % Update BPM display in the application
            app.BPMLabel.Text = sprintf('BPM: %.2f', bpm);
        end
```

```
end

% More private methods for app startup and data handling
methods (Access = private)

    % Startup function to initialize the timer
    function startupFcn(app)
        app.IsRunning = true;
        app.Timer = timer;
        app.Timer.Period = 0.1;
        app.Timer.ExecutionMode = 'fixedRate';
        app.Timer.TimerFcn = @(~, ~) app.timerCallback;
        start(app.Timer);
    end

    % Function to handle Start button press
    function StartButtonPushed(app, ~)
        app.IsRunning = true;
        app.a = serialport("COM5", 9600); % Initialize serial port
        configureTerminator(app.a, "CR/LF");
        flush(app.a); % Clear serial buffer
        app.a.UserData = struct("Data", [], "Order", 1, "fData", [], "ffData", []);

        % Continuous data acquisition and processing loop
        while app.a.UserData.Order < 100000
            app.Data = readline(app.a); % Read data from serial port
            app.a.UserData.Data(end+1) = str2double(app.Data); % Append new data
            app.a.UserData.Order = app.a.UserData.Order + 1;
            % Update raw signal plot periodically
            if mod(app.a.UserData.Order, 10) == 0
                configureCallback(app.a, "off");
                plot(app.UIAxes, app.a.UserData.Data(max(1, end-app.plotWindow+1):end));
                % Adjust plot limits based on data range
                minValue = min(app.a.UserData.Data);
                maxValue = max(app.a.UserData.Data);
                ylim(app.UIAxes, [minValue-0.1*abs(minValue),
maxValue+0.1*abs(maxValue)]);

                % Apply high-pass filtering
                [app.coefficient_1, app.coefficient_2] = butter(4, app.fc3/(app.fs/2), 'high');
                app.a.UserData.ffData = filter(app.coefficient_1, app.coefficient_2,
app.a.UserData.Data(1:end));

                % Apply band-pass filtering
                [app.coefficient_3, app.coefficient_4] = butter(app.orderOfFilter,
[app.fc1/(app.fs/2), app.fc2/(app.fs/2)], 'bandpass');
                app.a.UserData.fData = filter(app.coefficient_3, app.coefficient_4,
app.a.UserData.Data);

                % Update filtered signal plot
                plot(app.UIAxes_2, app.a.UserData.fData(max(1, end-app.plotWindow+1):end)
* 3);
                drawnow;
                configureTerminator(app.a, "CR/LF");
            end

            % Perform FFT and update frequency plots every 500 samples
            if mod(app.a.UserData.Order, 500) == 0
                configureCallback(app.a, "off");
```

```
% FFT for raw signal
dataLengthFF = length(app.a.UserData.ffData);
if dataLengthFF >= 6000
    app.Y = fft(app.a.UserData.ffData(end-5999:end));
else
    app.Y = fft(app.a.UserData.ffData);
end
app.P2 = abs(app.Y / max(1, dataLengthFF));
app.f = app.fs * (0:(length(app.Y)/2)) / length(app.Y);
app.P1 = app.P2(1:length(app.Y)/2+1);
app.P1(2:end-1) = 2 * app.P1(2:end-1);
plot(app.UIAxes_3, app.f, app.P1, 'Color', 'r');
set(gca, 'ylim', [0, 10]);

% FFT for filtered signal
dataLengthF = length(app.a.UserData.fData);
if dataLengthF >= 6000
    app.Y1 = fft(app.a.UserData.fData(end-5999:end));
else
    app.Y1 = fft(app.a.UserData.fData);
end
app.P3 = abs(app.Y1 / max(1, dataLengthF));
app.f1 = app.fs * (0:(length(app.Y1)/2)) / length(app.Y1);
app.P4 = app.P3(1:length(app.Y1)/2+1);
app.P4(2:end-1) = 2 * app.P4(2:end-1);
plot(app.UIAxes_4, app.f1, app.P4, 'Color', 'r');
set(gca, 'ylim', [0, 15]);
```

```
            grid on;
            app.m = app.m + 500;
            configureTerminator(app.a, "CR/LF");
        end

        % BPM calculation every 1500 samples
        if mod(app.a.UserData.Order, 1500) == 0
            timeAtPointBPM = toc;
            tic;
            app.bpmCalculation();
        end
    end
end

% Function to handle Stop button press
function StopButtonPushed(app, ~)
    app.IsRunning = false;
    delete(app.a);
end

% Update low-pass cutoff frequency when user modifies it
function LowpassfcEditFieldValueChanged(app, ~)
    app.fc2 = app.LowpassfcEditField.Value;
end

% Update high-pass cutoff frequency when user modifies it
function HighpassfcEditFieldValueChanged(app, ~)
    app.fc1 = app.HighpassfcEditField.Value;
end
```

```
    % Update filter order when user modifies it
    function EditOrderEditFieldValueChanged(app, ~)
        app.orderOfFilter = app.EditOrderEditField.Value;
    end
end

% Private method to create the UI components
methods (Access = private)
    function createComponents(app)
        app.UIFigure = uifigure('Visible', 'off');
        app.UIFigure.Color = [0.651 0.651 0.651];
        app.UIFigure.Position = [100 100 692 533];
        app.UIFigure.Name = 'MATLAB App';

        % Axes for raw signal
        app.UIAxes = uiaxes(app.UIFigure);
        title(app.UIAxes, 'Raw Signal');
        xlabel(app.UIAxes, 'Time(s)');
        ylabel(app.UIAxes, 'Amplitude(V)');
        app.UIAxes.Position = [25 219 300 174];

        % Axes for filtered signal
        app.UIAxes_2 = uiaxes(app.UIFigure);
        title(app.UIAxes_2, 'Filtered Signal');
        xlabel(app.UIAxes_2, 'Time(s)');
        ylabel(app.UIAxes_2, 'Amplitude(V)');
        app.UIAxes_2.Position = [371 219 301 156];
```

```
            % Axes for FFT of raw signal
            app.UIAxes_3 = uiaxes(app.UIFigure);
            title(app.UIAxes_3, 'Raw Signal FFT');
            xlabel(app.UIAxes_3, 'Frequency(Hz)');
            ylabel(app.UIAxes_3, 'Amplitude(V)');
            app.UIAxes_3.Position = [25 29 300 155];

            % Axes for FFT of filtered signal
            app.UIAxes_4 = uiaxes(app.UIFigure);
            title(app.UIAxes_4, 'Filtered Signal FFT');
            xlabel(app.UIAxes_4, 'Frequency(Hz)');
            ylabel(app.UIAxes_4, 'Amplitude(V)');
            app.UIAxes_4.Position = [378 29 288 155];

            % Start button
            app.StartButton = uibutton(app.UIFigure, 'push');
            app.StartButton.ButtonPushedFcn = createCallbackFcn(app, @StartButtonPushed,
true);
            app.StartButton.BackgroundColor = [0 1 0];
            app.StartButton.FontWeight = 'bold';
            app.StartButton.Position = [25 425 100 22];
            app.StartButton.Text = 'Start';

            % Stop button
            app.StopButton = uibutton(app.UIFigure, 'push');
            app.StopButton.ButtonPushedFcn = createCallbackFcn(app, @StopButtonPushed,
true);
            app.StopButton.BackgroundColor = [1 0 0];
            app.StopButton.FontWeight = 'bold';
            app.StopButton.Position = [157 425 100 22];
            app.StopButton.Text = 'Stop';


            % High-pass filter cutoff frequency field and label
            app.HighpassfcEditFieldLabel = uilabel(app.UIFigure);
            app.HighpassfcEditFieldLabel.HorizontalAlignment = 'right';
            app.HighpassfcEditFieldLabel.FontWeight = 'bold';
            app.HighpassfcEditFieldLabel.Position = [425 471 77 22];
            app.HighpassfcEditFieldLabel.Text = 'High pass fc';


            app.HighpassfcEditField = uieditfield(app.UIFigure, 'numeric');
            app.HighpassfcEditField.ValueChangedFcn = createCallbackFcn(app,
@HighpassfcEditFieldValueChanged, true);
            app.HighpassfcEditField.FontWeight = 'bold';
            app.HighpassfcEditField.Position = [517 471 100 22];
            app.HighpassfcEditField.Value = 30;


            % Low-pass filter cutoff frequency field and label
            app.LowpassfcEditFieldLabel = uilabel(app.UIFigure);
            app.LowpassfcEditFieldLabel.HorizontalAlignment = 'right';
            app.LowpassfcEditFieldLabel.FontWeight = 'bold';
            app.LowpassfcEditFieldLabel.Position = [428 503 74 22];
            app.LowpassfcEditFieldLabel.Text = 'Low pass fc';


            app.LowpassfcEditField = uieditfield(app.UIFigure, 'numeric');
            app.LowpassfcEditField.ValueChangedFcn = createCallbackFcn(app,
@LowpassfcEditFieldValueChanged, true);
            app.LowpassfcEditField.FontWeight = 'bold';
            app.LowpassfcEditField.Position = [517 503 100 22];
            app.LowpassfcEditField.Value = 200;
            % Filter order field and label
            app.EditOrderEditFieldLabel = uilabel(app.UIFigure);
            app.EditOrderEditFieldLabel.HorizontalAlignment = 'right';
            app.EditOrderEditFieldLabel.FontWeight = 'bold';
            app.EditOrderEditFieldLabel.Position = [438 436 64 22];
            app.EditOrderEditFieldLabel.Text = 'Edit Order';

            app.EditOrderEditField = uieditfield(app.UIFigure, 'numeric');
            app.EditOrderEditField.ValueChangedFcn = createCallbackFcn(app,
@EditOrderEditFieldValueChanged, true);
            app.EditOrderEditField.Position = [517 436 100 22];
            app.EditOrderEditField.Value = 4;

            % BPM and RPM labels
            app.BPMLabel = uilabel(app.UIFigure);
            app.BPMLabel.Position = [281 492 31 22];
            app.BPMLabel.Text = 'BPM';

            app.RPMLabel = uilabel(app.UIFigure);
            app.RPMLabel.Position = [281 457 32 22];
            app.RPMLabel.Text = 'RPM';

            % Text area for displaying messages or status
            app.TextAreaLabel = uilabel(app.UIFigure);
            app.TextAreaLabel.HorizontalAlignment = 'right';
            app.TextAreaLabel.Position = [26 493 55 22];
            app.TextAreaLabel.Text = 'Text Area';

            app.TextArea = uitextarea(app.UIFigure);
            app.TextArea.Position = [96 457 150 60];
            app.TextArea.Value = {'SIGNAL AND SYSTEMS'; 'SAPA-PROJECT';
'MATLAB'};

            app.UIFigure.Visible = 'on';
        end
    end

    % Public methods for app lifecycle
    methods (Access = public)
        % Constructor
        function app = project
            createComponents(app);
            registerApp(app, app.UIFigure);
            runStartupFcn(app, @startupFcn);

            if nargout == 0
                clear app;
            end
        end

        % Destructor
        function delete(app)
            delete(app.UIFigure);
        end
    end
```

This is a class file written to describe a developed MATLAB application that processes, filters, and visualizes electronic stethoscope data. The code identifies components that make up the graphical user interface of the application and their respective functions. It contains many components, such as: axes for plotting raw signals and filtered signals, start-stop buttons, fields for changing the cut-off frequency and filter order. Besides, real-time processing of data is provided using a timer after the start of an application. These signals then pass through high pass and band pass filters during data processing to eliminate unwanted noise and to obtain useful signals. The filtered signals are also analyzed with Fast Fourier Transform (FFT) and the frequency spectra calculated. It's also possible to monitor the signals in real time and change required parameters on the user interface. The 'Start' button starts the data acquisition and processing, plots signals in graphs, and performs some analysis. The 'Stop' button stops this process and cleans up the resources in use. Upon opening the application, all the interface components are created and the initialization function is called. The application should process and analyze electronic stethoscope data through a user-friendly interface.

5. RESULT AND DISCUSSION

The Electronic Stethoscope Project was to design a system capable of capturing, amplifying, filtering, and analyzing sounds such as heartbeats and lung activity. Special amplification and filtering circuits were developed that could suppress noise and emphasize important frequencies of the sound. Active low-pass and high-pass filters are employed in the system, thus it can block unwanted noise and the signals are clearer and more reliable.

Hardware design was cautiously planned and built on breadboards using high-quality components, such as TL072 operational amplifiers, to enhance the clarity and stability of the signal. It includes switches for easy switching between heart and lung circuits by team members.

On the software side, MATLAB supported the system in processing and analyzing the signals in real time. A custom graphical interface allowed team members to see raw and filtered signals; they can adjust filters and see all vital measurements such as heart rate and respiratory rate dynamically.

Analysis of signals via FFT decomposed them into constituent frequencies, which can provide much better insights into frequency anomalies that may point out a health-related problem.

Analogue signals were acquired quite efficiently by Arduino and ported onto MATLAB. In MATLAB, a very strong platform for data processing, visualization, and analysis of these signals could be effectively carried out in real time. In particular, dynamic filter settings and data visualization tools allowed users to inspect the signals in detail and effortlessly adjust the filter parameters. Besides, a graphical interface developed on MATLAB enabled the system to provide an interactive, user-friendly experience, while allowing the visualization of both raw and filtered signals.

In essence, this is where the electronic stethoscope project brings the combination of theory and application of concepts into problem-solving. Being successful through iterative design and testing entailed ensuring that adaptation and persistence overcame obstacles to be sure outcomes were achieved in engineering.

6. CONCLUSION

The Electronic Stethoscope Project successfully bridged the gap from traditional stethoscope functionality to the latest in digital technology. Carefully designed hardware circuits, among them custom filters and amplifiers, captured and enhanced heart and lung sounds by reducing noise and isolating important frequencies. With the described project, MATLAB played an invaluable central part while allowing the processing of real signals with minimal interaction in a user-friendly display raw and filtered signals, enable manipulation in filter settings, and observation in some key metrics, like heartbeat and respiration rate. Integration with Arduino ensures proper data acquisition and smooth transmission into this software for analysis. The project addressed the need for flexibility, innovation, and teamwork in finding engineering solutions through a blend of theoretical knowledge and hands-on experimentation; thus, delivering a practical and reliable diagnostic tool with great potential for modern healthcare.

7. REFERENCES

1) https://www.electronics-tutorials.ws/opamp/opamp_3.html
2) https://www.electronics-tutorials.ws/filter/filter_7.html
3) Smith, S. W. (1997). *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing.
4) Oppenheim, A. V., & Schafer, R. W. (2010). *Discrete-Time Signal Processing*. Pearson.
5) Sedra, A. S., & Smith, K. C. (2015). *Microelectronic Circuits* (7th Edition). Oxford University Press.
6) Madisetti, V. K., & Williams, D. B. (1999). *Digital Signal Processing Handbook*. CRC Press.
7) Fourier transform - Wikipedia