Department of

Electrical & Electronics Engineering

Abdullah Gül University

Project Report

SAPA CAPSULE

Submitted on: 02.01.2025

Submitted by: BEYZA YILDIRIM (2211011069)

Group Number/Name: **NEB**

Group Partner: **ESRA YILDIRIM (2211011060)**

NİSANUR TÜRKMEN (2211011052)

Grade: / 100

Table of Contents

OBJECTIVE

The aim of this project is to develop an advanced electronic stethoscope system that can accurately acquire, filter, amplify and digitise body sounds in a computer environment. The system aims to provide a robust solution for analysing heart and lung signals by addressing the limitations of conventional stethoscopes, such as low volume levels and sensitivity to background noise, by designing separate filters to be able to analyse the heart and lung signals and by enhancing the signal. A dual-filter system is implemented to effectively separate physiological sounds, with heart sounds filtered in the 35-425 Hz range and lung sounds filtered in the 200-1150 Hz range, ensuring minimal interference between signals. The project improves signal quality through amplification and filtering circuits, improving clarity and power for better interpretation. The system also digitises these signals for real-time analysis and visualisation on a computer with the help of Arduino uno and matlab.

BACKGROUND

Since its invention by René Theophile Hyacinthe Laënnec in 1816, the stethoscope which is often known as a symbol of medicine has undergone tremendous modification. It started as a simple wooden tube used to avoid direct contact with patients while listening to body sounds. This invention transformed non-invasive diagnostics by amplifying sounds like heartbeats. Later, improvements such as the Bowles and Sprague stethoscope in 1925 added flexible tubing and diaphragm-bell chest pieces, forming the foundation of modern stethoscopes. However, the ability of conventional stethoscopes to reduce background noise and enhance weak sounds is limited. Electronic stethoscopes were created with characteristics including digital interfaces, noise reduction, and sound amplification to solve these problems. To provide a more precise diagnostic tool, this research makes use of these developments and concentrates on enhancing signal processing and filtering.[1]

The Sallen-Key filter is a simple and effective circuit used to filter signals. It uses an op-amp, two resistors, and two capacitors. This design is stable and works well to let certain signals pass while blocking others. It can be used as a low-pass, high-pass, or bandpass filter. In this project, Sallen-Key bandpass filters separate heart sounds (35–425 Hz) and lung sounds (200–1150 Hz). These filters reduce noise and make the signals clear and easy to analyze. The filter's range and strength can be changed by adjusting the resistor and capacitor values. Multiple filters can also be connected for better results. MATLAB is used to test and improve the filters, helping to check their performance and make changes as needed. By using Sallen-Key filters, this project provides clear and accurate signals, making the electronic stethoscope more reliable for medical diagnosis.[2]

The filtering and signal processing system is designed, tested, and optimized using MATLAB. The stethoscope's analog signals are first collected, then an analog-to-digital converter (ADC) is used to convert them to digital form. These digital signals are then loaded into MATLAB for analysis. First, the raw signals are viewed in the time domain to identify noise and unwanted components. MATLAB tools are used to plot and examine these signals. Afterward, the signals are filtered using the parameters of the Sallen-Key filters modeled in MATLAB. The filtered results are compared to the raw signals to confirm that the desired frequency ranges are preserved and noise is reduced. To study the signals further, the Fast Fourier Transform (FFT) is applied to convert them into the frequency domain. This step helps show the spectral components of the signals, making it easier to check how well the filters work. MATLAB generates graphs for both the raw and filtered signals in the frequency domain to display noise reduction and the isolation of key frequencies. Time-domain plots of the filtered signals are also created to show improvements in clarity and strength. By refining the filters and analyzing the results in both time and frequency domains, MATLAB ensures that the system meets its goals. These steps help verify that the electronic stethoscope can process and improve body sounds for real-time diagnostics.[3]

ANALYTICAL AND SIMULATION PROCEDURES

1. As the first step of the project, the Ltspice part was prepared. The Sallen-Key bandpass filter was chosen for its ease of design and performance.
2. After deciding the heart frequency range as 35 - 425 Hz, resistance and capacitor values were selected by various calculations. For the heart Sallen-Key filter, 47nF capacitor (2), 127k ohm resistor, 3.5k ohm resistor (2), 68nF capacitor (2), universal op amp (2) were used.

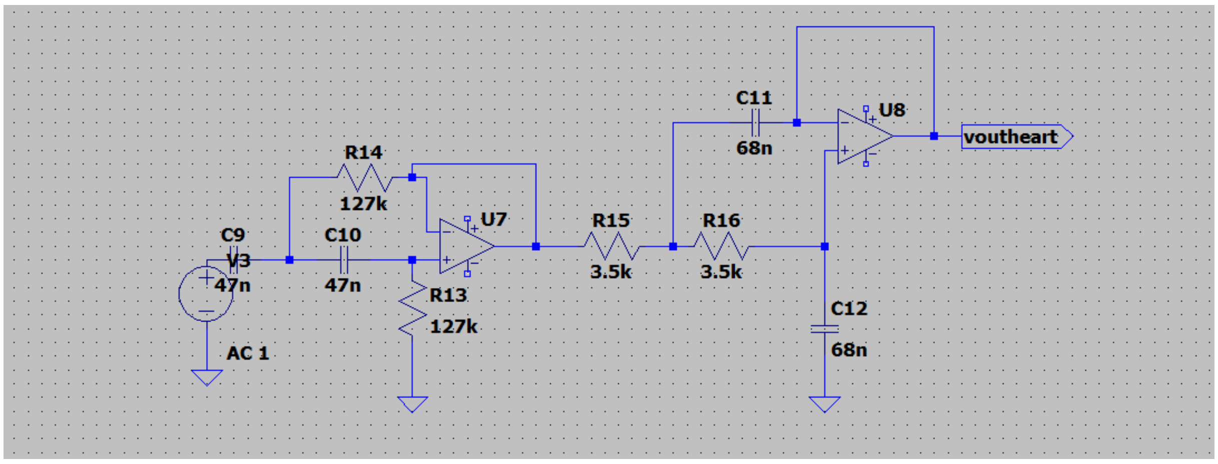
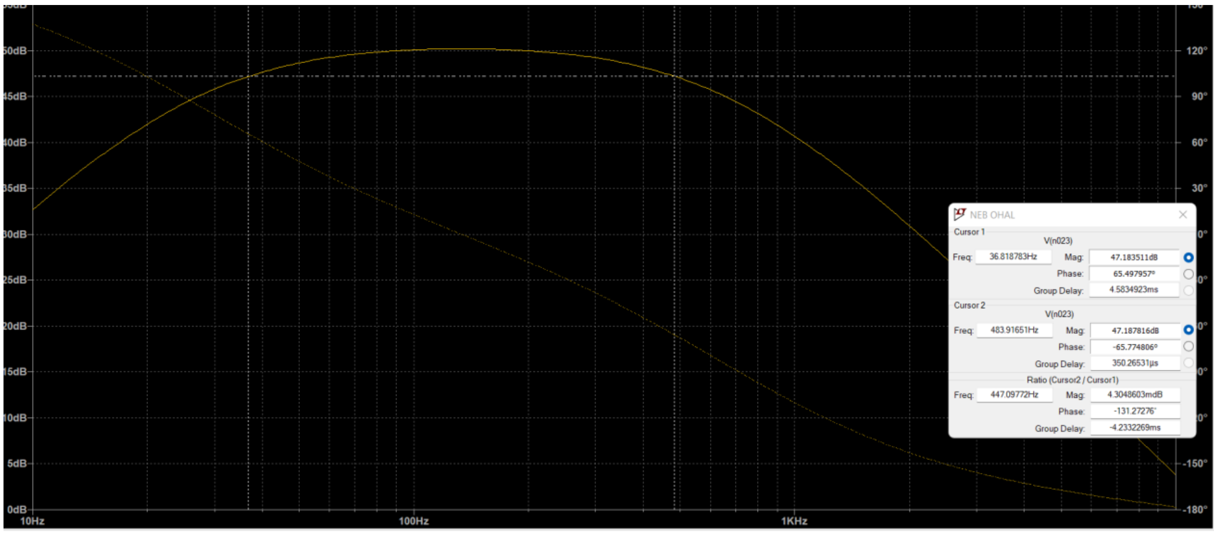


Figure 1: Sallen-Key Filter for heart signal

3. AC sweep analysis was used to check that the filter range was within the desired value. In this process, the -3 dB point was used to define the frequency at which the signal strength of a filter circuit drops to half its initial level.



Cursor 1
V(n023)
Freq: 36.818783Hz Mag: 47.183511dB
Phase: 65.497957°
Group Delay: 4.5834923ms

Cursor 2
V(n023)
Freq: 483.91651Hz Mag: 47.187816dB
Phase: -65.774806°
Group Delay: 350.26531µs

Ratio (Cursor2 / Cursor1)
Freq: 447.09772Hz Mag: 4.3048603mdB
Phase: -131.27276°
Group Delay: -4.2332269ms

Figure 2: Intervals of the heart filter

4. After deciding the frequency range of the lung filter from 200 to 1150, resistance and capacitor values were decided by various calculations.
5. For the lung Sallen-Key filter, 12n capacitor (2), 50k ohm resistor, 77k ohm resistor, 10k ohm resistor (2), 12nF capacitor (2), universal op amp (2) were used.

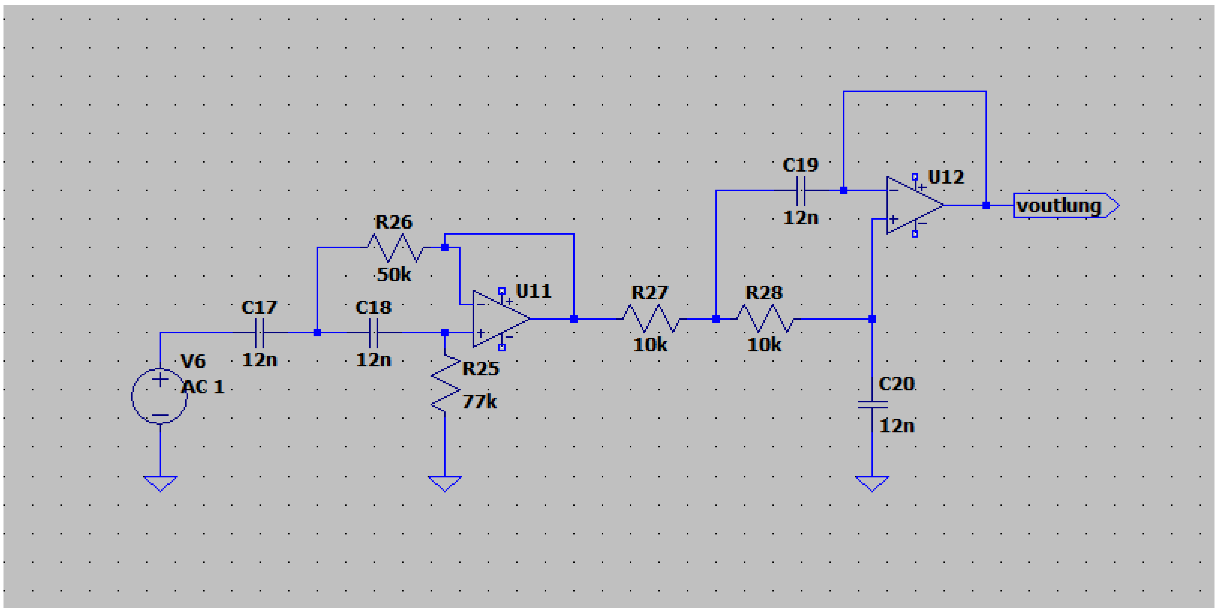
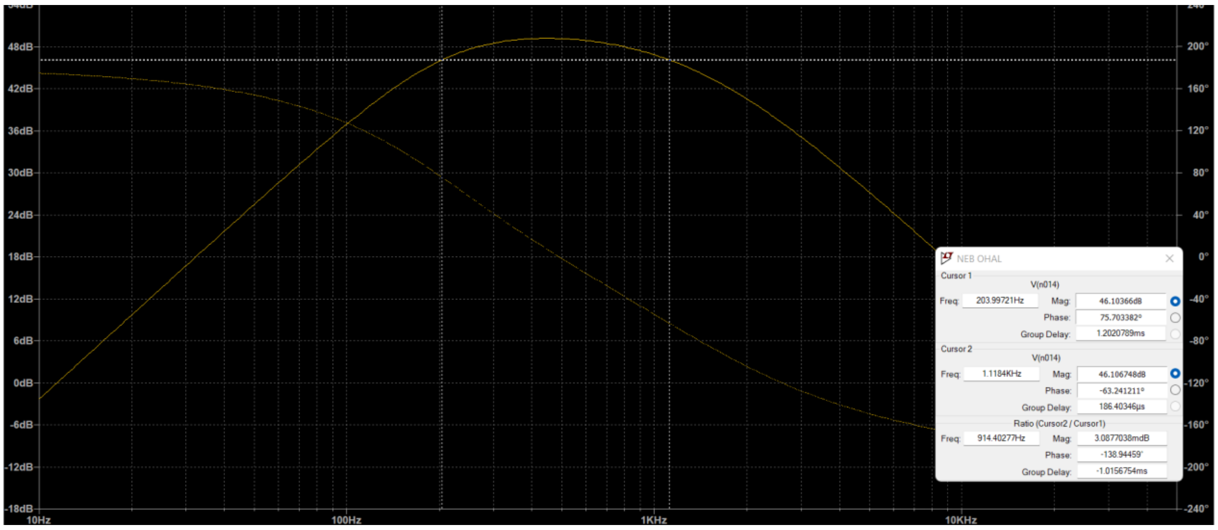


Figure 5: Sallen-Key Filter for lung signal

6. AC sweep analysis was used to check that the filter range was within the desired value. In this process, the -3 dB point was used to define the frequency at which the signal strength of a filter circuit drops to half its initial level.



Cursor 1
V(n014)
Freq: 203.99721Hz Mag: 46.10366dB
Phase: 75.703382°
Group Delay: 1.2020789ms
Cursor 2
V(n014)
Freq: 1.1184KHz Mag: 46.106748dB
Phase: -63.241211°
Group Delay: 186.40346µs
Ratio (Cursor2 / Cursor1)
Freq: 914.40277Hz Mag: 3.0877038mdB
Phase: -138.94459°
Group Delay: -1.0156754ms

Figure 4: Intervals of the lung filter

7. After the filter circuits were dried, the remaining 2 parts of the heart and lung circuits were installed. The first part is the microphone circuit and the second part is the speaker circuit.

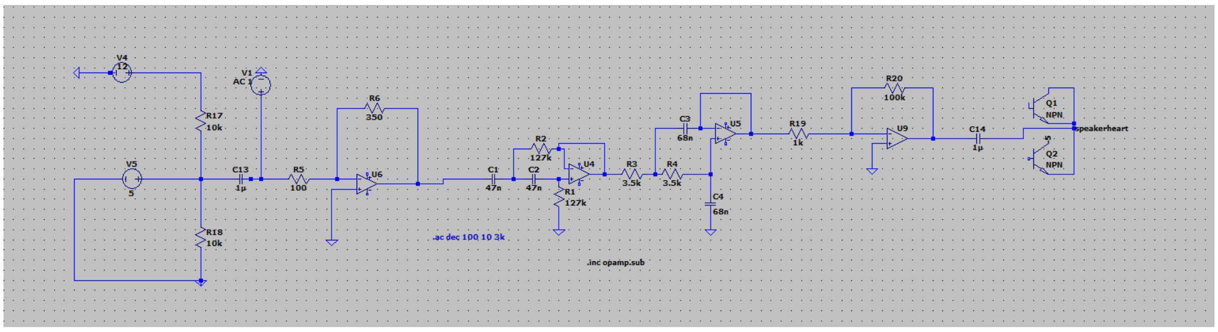


Figure 5: Heart circuit

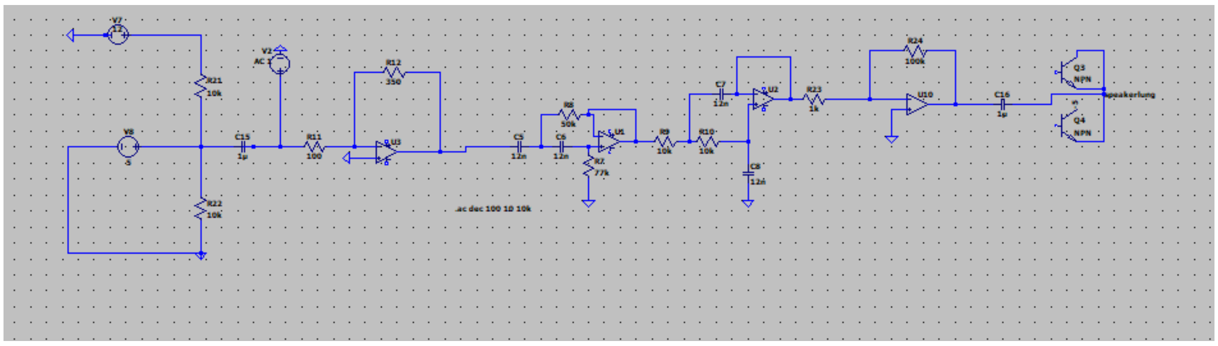


Figure 6: Lung circuit

Components	Numbers
NPN transistor	4
1 uF capacitor	4
47 nF capacitor	2
68 nF capacitor	2
12n capacitor	4
10k ohm	6
350 ohm	2
100 ohm	2
50k ohm	1
77k ohm	1
1k ohm	2
100k ohm	2
3.5k ohm	2
127k ohm	2
Universal op amp	8

Table 1: Elements used in Ltspice

8. In order to build the circuit in real life, the components closest to the LTspice design were selected. These components included TL072 op-amp, LM358P op-amp, transistors (BD139 and BD140) and speaker.

9. Before the microphone circuit was installed, the stethoscope was disconnected from the ear connection and the microphone was inserted into the stethoscope with the help of a thin tube. It was connected to the microphone circuit with the help of soldered jumpers. This circuit has the same characteristics as the circuit designed on LTspice.

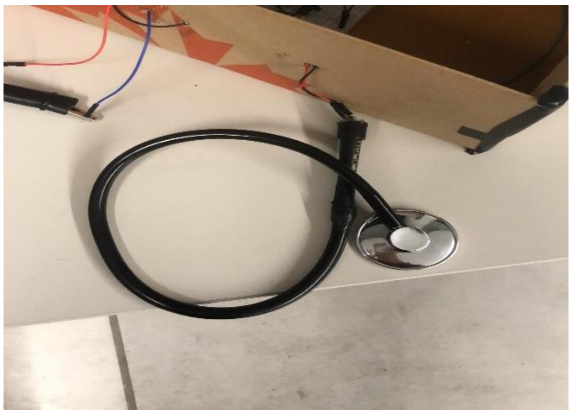


Figure 7: The stethoscope connected with microphone

10. A 1k resistor and 1uF capacitor were connected to the transistors and the speaker circuit was started to be installed. The speaker was again soldered to the jumpers and connected to the circuit.
11. The microphone and speaker circuits were mounted on the same breadboard. The positive ends were connected to separate inputs and the negative ends were grounded. Since the filters will be connected here later, the positive nutrition, negative nutrition and ground connections of this breadboard were completed.

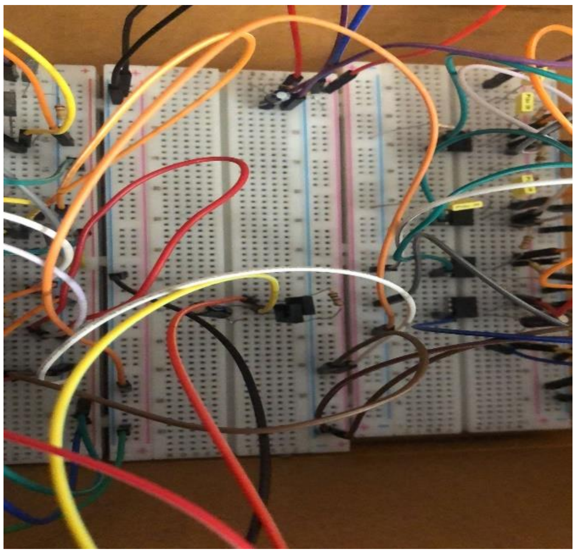


Figure 8: The speaker and mic common breadboard

12. Filters for the heart (35-425 Hz) and lung (200-1150 Hz) frequencies were built on breadboard and tested separately to ensure that they filtered the desired frequency ranges.

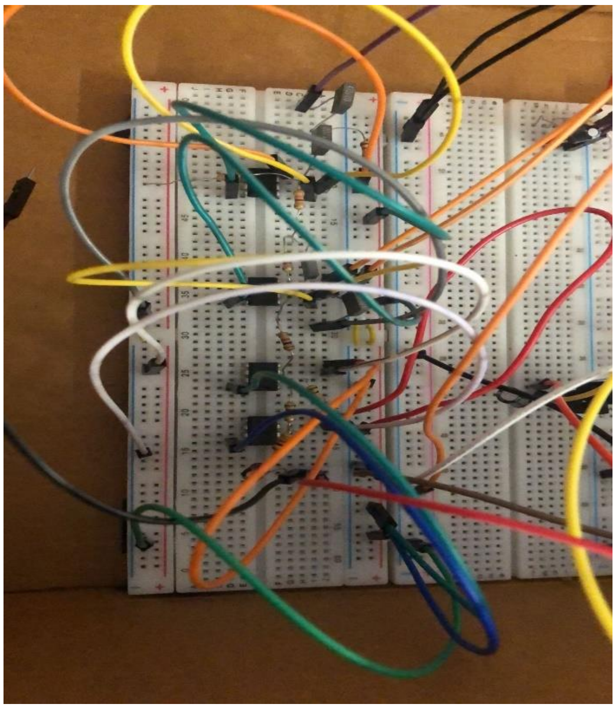


Figure 9: The circuit of the installed heart filter

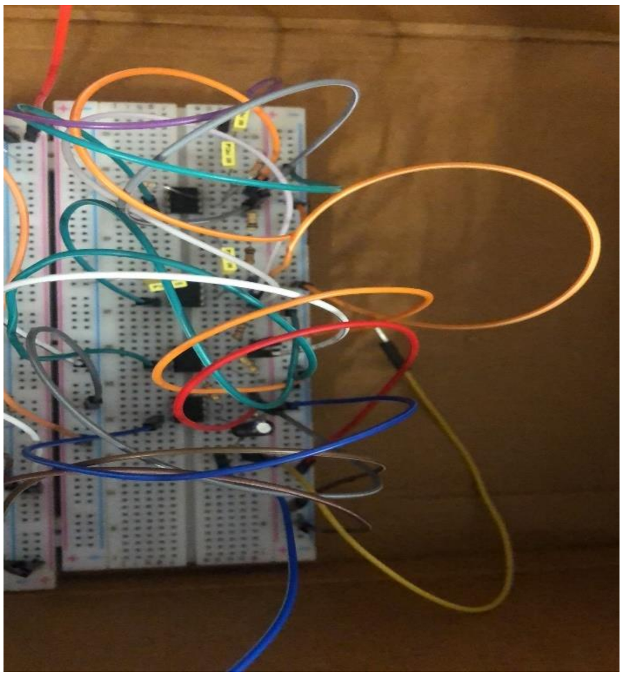


Figure 10: The circuit of the installed lung filter

13. The circuits were placed in a box, holes were drilled in the box for power and ground connections, and the microphone-speaker circuit was placed in the center of the box.
14. Heart and lung filters were placed on either side of the central circuit. Connections from the microphone output to the filter input and from the filter to the speaker input were made with jumper cables. The filters were connected to the + supply, - supply and ground of the central circuit.

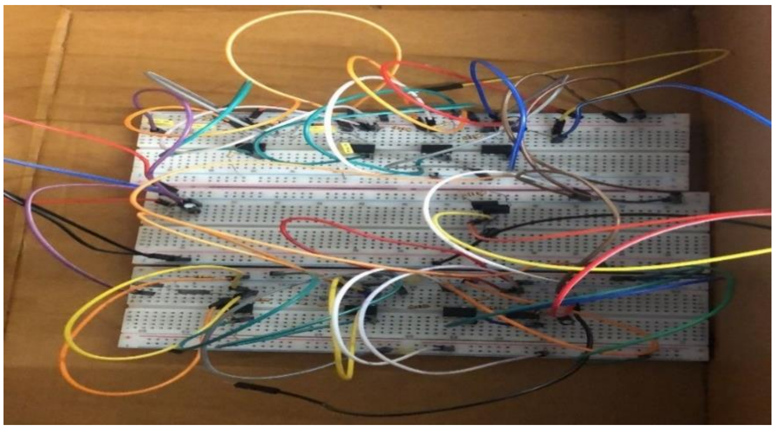


Figure 11: The circuits in the box

15. The heart filter was connected to the circuit and observed on an oscilloscope to verify that the signal and sound were working properly.

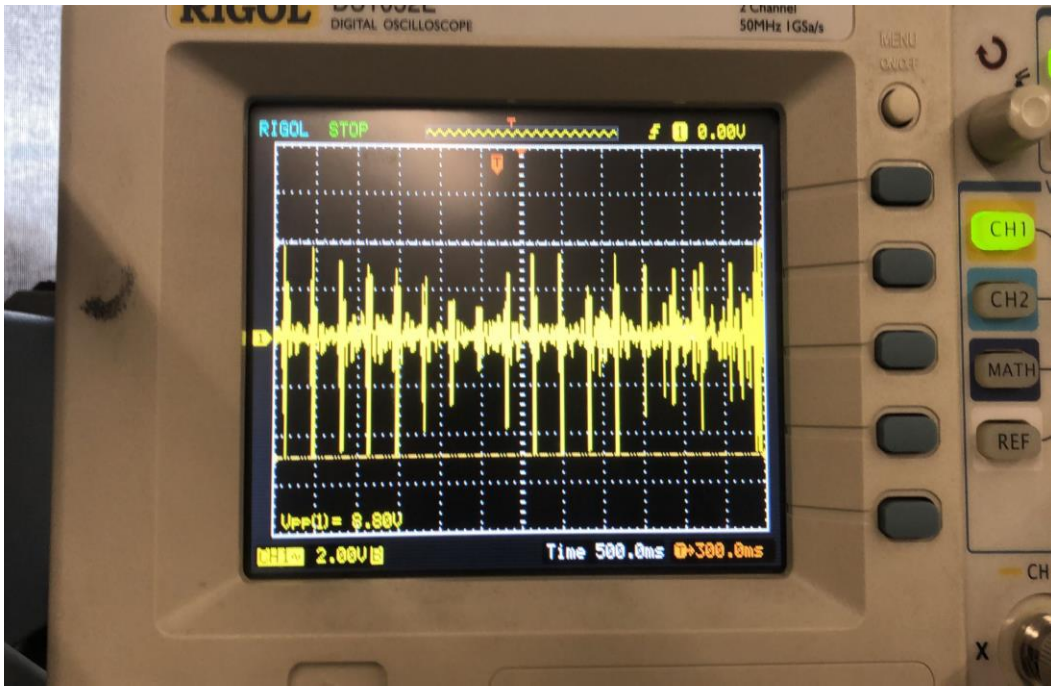


Figure 12: Phonocardiogram (PCG) signal on the oscilloscope

16. The lung filter was connected to the circuit and the signal and sound were checked on an oscilloscope for proper functioning.

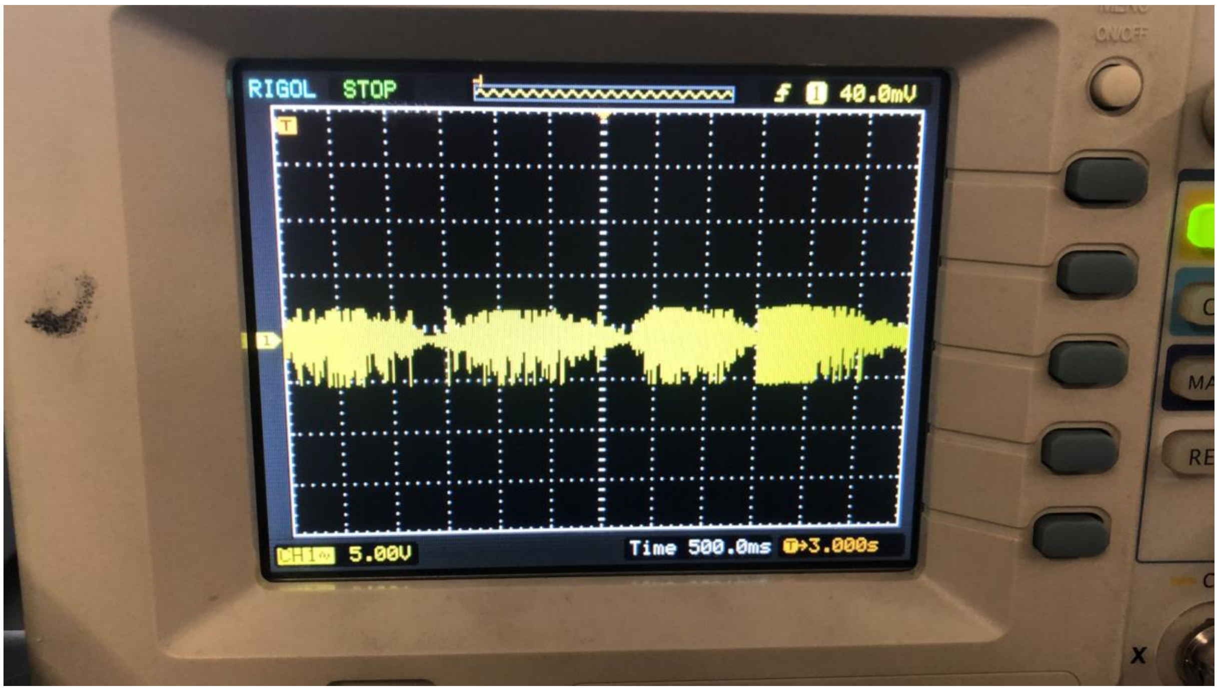


Figure 13: Respiratory signal on the oscilloscope

17. All connections were reviewed, circuits were fine-tuned for optimal performance and the completed system was tested with a stethoscope.

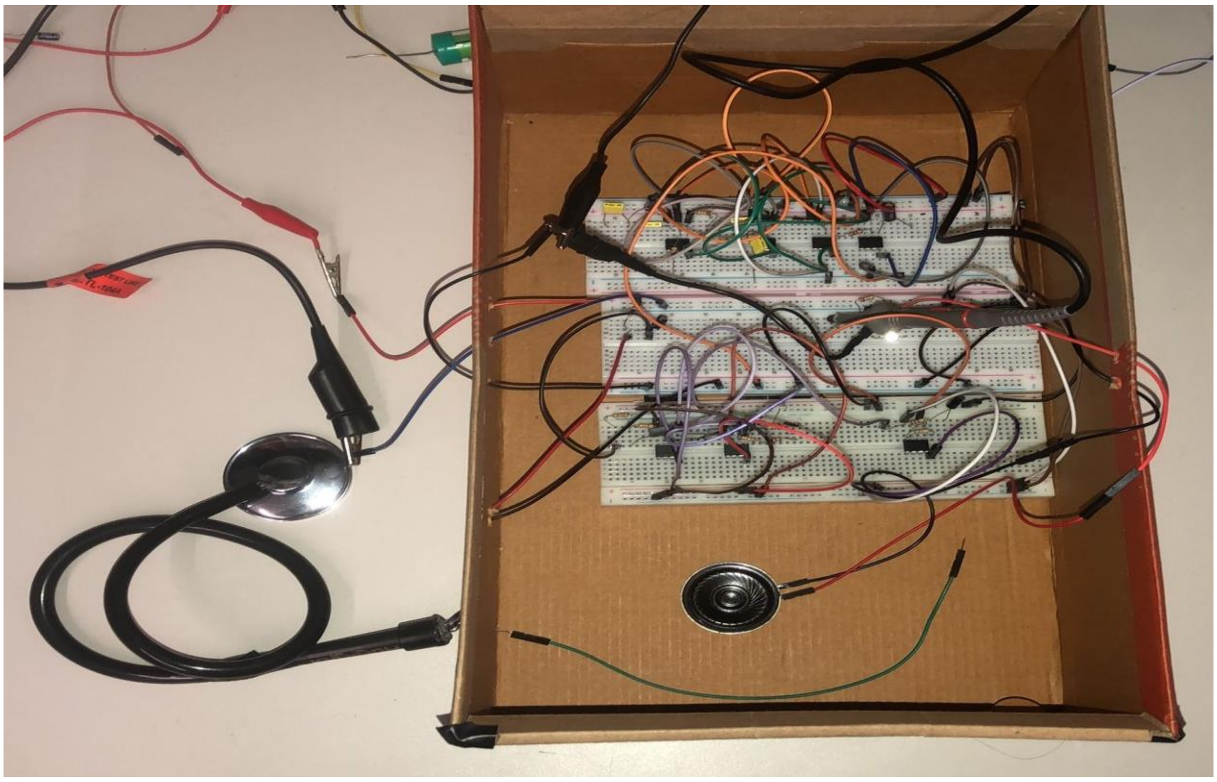


Figure 14: Final version of the circuit in the box

When designing filter circuits, the desired range was set with the help of calculators used to calculate the filter range. [4] [5]

Code Part

```
classdef testcode < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                        matlab.ui.Figure
        Label2                          matlab.ui.control.Label
        RPMLabel                        matlab.ui.control.Label
        BPMLabel                        matlab.ui.control.Label
        LableLabel                      matlab.ui.control.Label
        OrderofFilterEditField          matlab.ui.control.NumericEditField
        OrderofFilterEditFieldLabel     matlab.ui.control.Label
        LowPassCutoffEditField          matlab.ui.control.NumericEditField
        LowPassCutoffEditFieldLabel     matlab.ui.control.Label
        HighPassCutoffEditField         matlab.ui.control.NumericEditField
        HighPassCutoffEditFieldLabel    matlab.ui.control.Label
        StopButton                      matlab.ui.control.Button
        StartButton                     matlab.ui.control.Button
        UIAxes_4                        matlab.ui.control.UIAxes
        UIAxes_3                        matlab.ui.control.UIAxes
        UIAxes_2                        matlab.ui.control.UIAxes
        UIAxes                          matlab.ui.control.UIAxes
    end
```

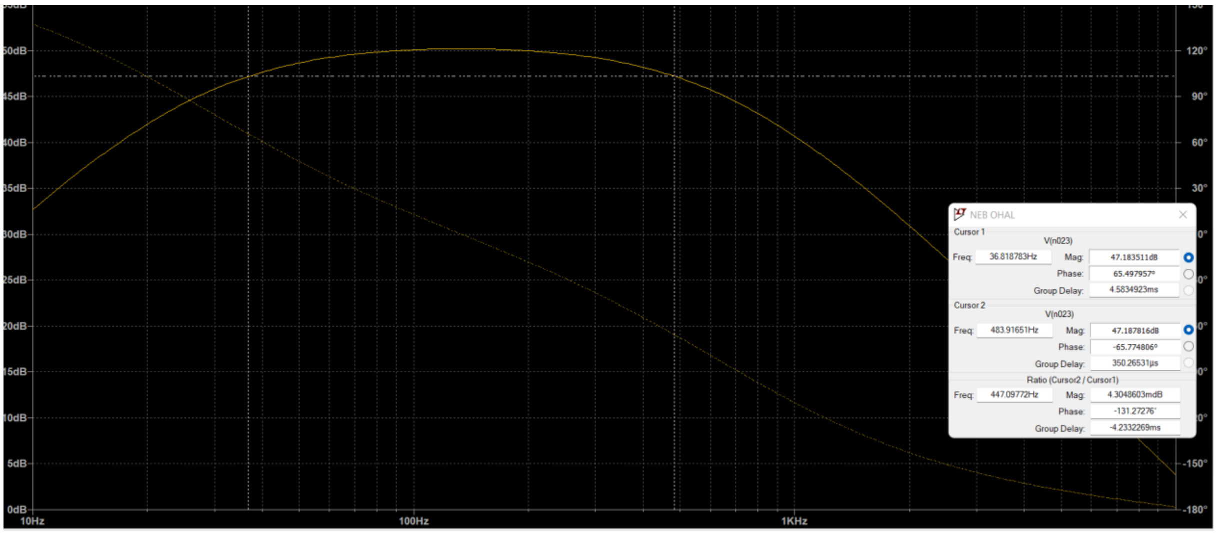
```
     properties (Access = private)
        a
        f
        f1
        Y
        Y1
        P1
        P2
        P3
        P4
        filteredPlot
        rawPlot
        serialport
        Fs = 8000;
        fs = 5000;
        fc1 = 30;
        fc2 = 200;
        T = 1/5000;
        n = 1;
        m = 1;
        m1 = 1;
        plotWindow = 2000;
        IsRunning = false;
        Timer
        Data
        ffData
        fdata
        coefficient_1
        coefficient_2
        fc3=30
        rpm
        bpm
        coefficient_3
        coefficient_4
        orderOfFilter=4;
        end

    methods (Access = private)


   function calculateRPM(app)
    % Get the last window of the signal
    signalWindow = app.a.UserData.Data(max(1, end-app.plotWindow+1):end);

    % Find the peaks
    threshold = 510; % Threshold value for peak detection
    [peaks, locs] = findpeaks(signalWindow, 'MinPeakHeight', threshold);

    % Calculate the time interval
    if length(locs) > 1
        timeIntervals = diff(locs) * app.T; % Time differences between peaks (in
seconds)
        avgTimePerCycle = mean(timeIntervals); % Average time difference
app.rpm = 10 / avgTimePerCycle; % Calculate RPM

    else
        app.rpm = 0; % If there aren't enough peaks, RPM is zero
    end
         % Update the label for RPM
    app.Label2.Text = sprintf('%.2f RPM', app.rpm);

end



        function timerCallback = func(app)
            if ~app.IsRunning
            stop(app.Timer);
        end
    end
 end

    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            app.IsRunning=true;
            app.Timer=timer;
            app.Timer.Period=0.03;
            app.Timer.ExecutionMode='fixedRate';
            app.Timer.TimerFcn=@(~,~)app.timerCallback;
            start(app.Timer);


        end

        % Button pushed function: StartButton
        function StartButtonPushed(app, event)
            app.IsRunning = true;
           app.a =serialport ("COM8", 115200);
            configureTerminator(app.a, "CR/LF");
            flush(app.a);
            app.a.UserData = struct("Data", [], "Order", 1, "fData",
[],"ffData",[]);
            app.IsRunning = true;

            while app.a.UserData.Order < 100000
                    app.Data = readline(app.a);
                    app.a.UserData.Data(end+1) = str2double(app.Data);
                    app.a.UserData.Order = app.a.UserData.Order + 1;
                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))
                      [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));
                    [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);
```

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

                    configureTerminator(app.a, "CR/LF");

                end
                 if mod(app.a.UserData.Order,500)==0
                     configureCallback(app.a, "off");

                     app.Y = fft(app.a.UserData.ffData(end-499+1:end));
                     app.P2 = abs(app.Y / 500); % Use 400 for normalization
                     app.f = app.fs * (0:(length(app.Y)/2))/length(app.Y); %
Adjust frequency axis
                     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]);

                     app.Y1 = fft(app.a.UserData.fData(end-499+1:end));
                     app.P3 = abs(app.Y1 / 500); % Use 400 for normalization
                     app.f1 = app.fs * (0:(length(app.Y1)/2))/length(app.Y1); %
Adjust frequency axis
                     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
if mod(app.a.UserData.Order, 1500) == 0
    % Measure elapsed time
    timeAtPointBPM = toc;

    % Detect peaks for BPM
    thresholdBPM = 510; % Threshold value for BPM detection
    [peaksBPM, ~] = findpeaks(app.a.UserData.Data(end-1499+1:end),
'MinPeakHeight', thresholdBPM);
    numPeaksBPM = numel(peaksBPM); % Number of peaks for BPM

    % Calculate BPM
    app.bpm = (10 * numPeaksBPM) / timeAtPointBPM; % BPM hesaplama
    disp(int16(app.bpm)); % Print to console
    app.LableLabel.Text = num2str(app.bpm); % Display in GUI
    tic;

    % Detect peaks for RPM
    thresholdRPM = 510; % Threshold value for RPM detection
    [peaksRPM, ~] = findpeaks(app.a.UserData.Data(end-1499+1:end),
'MinPeakHeight', thresholdRPM);
    numPeaksRPM = numel(peaksRPM); % Number of peaks for RPM

    if numPeaksRPM > 1
        % Calculate the time interval
        timeIntervals = diff(find(peaksRPM)) ; % Time differences between peaks
        avgTimePerCycle = mean(timeIntervals); % Average time difference
        app.rpm = 0.01 / avgTimePerCycle; % Calculate RPM
    else
        app.rpm = 0; If not enough peaks, RPM is zero
    end

    % Display RPM
    disp(int16(app.rpm)); % Print to console
    app.Label2.Text = num2str(app.rpm); % Display in GUI
end

                end
        end

        % Button pushed function: StopButton
        function StopButtonPushed(app, event)
            app.IsRunning=false;
            delete(app.a);
        end

        % Value changed function: LowPassCutoffEditField
        function LowPassCutoffEditFieldValueChanged(app, event)
            b=app.LowPassCutoffEditField.Value;
            app.fc2=b;
        end

        % Value changed function: HighPassCutoffEditField
        function HighPassCutoffEditFieldValueChanged(app, event)
             c= app.HighPassCutoffEditField.Value;
             app.fc1=c;
        end

        % Value changed function: OrderofFilterEditField
        function OrderofFilterEditFieldValueChanged(app, event)
            j = app.OrderofFilterEditField.Value;
            app.orderOfFilter=j;
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            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';

            % Create UIAxes
            app.UIAxes = uiaxes(app.UIFigure);
            title(app.UIAxes, 'Raw-Signal (Oscilloscope)')
            xlabel(app.UIAxes, 'Time')
            ylabel(app.UIAxes, 'Y')
            zlabel(app.UIAxes, 'Z')
            app.UIAxes.YLim = [-1000 1000];
            app.UIAxes.Position = [24 228 300 174];

            % Create UIAxes_2
            app.UIAxes_2 = uiaxes(app.UIFigure);
            title(app.UIAxes_2, 'Filtered-Signal')
            xlabel(app.UIAxes_2, 'Time')
            ylabel(app.UIAxes_2, 'Y')
            zlabel(app.UIAxes_2, 'Z')
            app.UIAxes_2.YLim = [-1000 1000];
            app.UIAxes_2.Position = [364 237 301 156];

            % Create UIAxes_3
            app.UIAxes_3 = uiaxes(app.UIFigure);
            title(app.UIAxes_3, 'Raw-Signal FFT')
            xlabel(app.UIAxes_3, 'Frequency')
            ylabel(app.UIAxes_3, 'Y')
            zlabel(app.UIAxes_3, 'Z')
            app.UIAxes_3.Position = [24 44 300 155];

            % Create UIAxes_4
            app.UIAxes_4 = uiaxes(app.UIFigure);
            title(app.UIAxes_4, 'Filtered-Signal FFT')
            xlabel(app.UIAxes_4, 'Frequency')
            ylabel(app.UIAxes_4, 'Y')
            zlabel(app.UIAxes_4, 'Z')
            app.UIAxes_4.Position = [377 44 288 155];

            % Create StartButton
            app.StartButton = uibutton(app.UIFigure, 'push');
            app.StartButton.ButtonPushedFcn = createCallbackFcn(app,
@StartButtonPushed, true);
            app.StartButton.BackgroundColor = [0 0.6 0];
            app.StartButton.FontWeight = 'bold';
            app.StartButton.Position = [24 503 283 22];
            app.StartButton.Text = 'Start';

            % Create StopButton
            app.StopButton = uibutton(app.UIFigure, 'push');
            app.StopButton.ButtonPushedFcn = createCallbackFcn(app,
@StopButtonPushed, true);
            app.StopButton.BackgroundColor = [0.5412 0.0118 0.1059];
            app.StopButton.FontWeight = 'bold';
            app.StopButton.Position = [24 471 283 22];
            app.StopButton.Text = 'Stop';

            % Create HighPassCutoffEditFieldLabel
            app.HighPassCutoffEditFieldLabel = uilabel(app.UIFigure);
            app.HighPassCutoffEditFieldLabel.HorizontalAlignment = 'right';
            app.HighPassCutoffEditFieldLabel.FontWeight = 'bold';
            app.HighPassCutoffEditFieldLabel.Position = [364 471 102 22];
            app.HighPassCutoffEditFieldLabel.Text = 'High-Pass Cutoff';

            % Create HighPassCutoffEditField
            app.HighPassCutoffEditField = uieditfield(app.UIFigure, 'numeric');
            app.HighPassCutoffEditField.ValueChangedFcn = createCallbackFcn(app,
@HighPassCutoffEditFieldValueChanged, true);
            app.HighPassCutoffEditField.FontWeight = 'bold';
            app.HighPassCutoffEditField.Position = [479 471 150 22];
            app.HighPassCutoffEditField.Value = 30;

            % Create LowPassCutoffEditFieldLabel
            app.LowPassCutoffEditFieldLabel = uilabel(app.UIFigure);
            app.LowPassCutoffEditFieldLabel.HorizontalAlignment = 'right';
            app.LowPassCutoffEditFieldLabel.FontWeight = 'bold';
            app.LowPassCutoffEditFieldLabel.Position = [364 503 100 22];
            app.LowPassCutoffEditFieldLabel.Text = 'Low-Pass Cutoff';

            % Create LowPassCutoffEditField
            app.LowPassCutoffEditField = uieditfield(app.UIFigure, 'numeric');
            app.LowPassCutoffEditField.ValueChangedFcn = createCallbackFcn(app,
@LowPassCutoffEditFieldValueChanged, true);
            app.LowPassCutoffEditField.FontWeight = 'bold';
            app.LowPassCutoffEditField.Position = [479 503 150 22];
            app.LowPassCutoffEditField.Value = 200;

            % Create OrderofFilterEditFieldLabel
            app.OrderofFilterEditFieldLabel = uilabel(app.UIFigure);
            app.OrderofFilterEditFieldLabel.HorizontalAlignment = 'right';
            app.OrderofFilterEditFieldLabel.FontName = 'AvantGarde';
            app.OrderofFilterEditFieldLabel.FontWeight = 'bold';
            app.OrderofFilterEditFieldLabel.Position = [364 436 85 22];
            app.OrderofFilterEditFieldLabel.Text = 'Order of Filter';

            % Create OrderofFilterEditField
            app.OrderofFilterEditField = uieditfield(app.UIFigure, 'numeric');
            app.OrderofFilterEditField.ValueChangedFcn = createCallbackFcn(app,
@OrderofFilterEditFieldValueChanged, true);
            app.OrderofFilterEditField.Position = [479 436 150 22];
            app.OrderofFilterEditField.Value = 4;

            % Create LableLabel
            app.LableLabel = uilabel(app.UIFigure);
            app.LableLabel.FontName = 'AvantGarde';
            app.LableLabel.FontWeight = 'bold';
            app.LableLabel.Position = [155 415 37 22];
            app.LableLabel.Text = 'Lable';

            % Create BPMLabel
            app.BPMLabel = uilabel(app.UIFigure);
            app.BPMLabel.Position = [121 415 31 22];
            app.BPMLabel.Text = 'BPM';

            % Create RPMLabel
            app.RPMLabel = uilabel(app.UIFigure);
            app.RPMLabel.BackgroundColor = [0.651 0.651 0.651];
            app.RPMLabel.Position = [121 436 32 22];
            app.RPMLabel.Text = 'RPM';

            % Create Label2
            app.Label2 = uilabel(app.UIFigure);
            app.Label2.FontName = 'AvantGarde';
            app.Label2.FontWeight = 'bold';
            app.Label2.Position = [152 436 43 22];
            app.Label2.Text = 'Label2';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
```

```
        methods (Access = public)

            % Construct app
            function app = ardadeneme2

                % Create UIFigure and components
                createComponents(app)

                % Register the app with App Designer
                registerApp(app, app.UIFigure)

                % Execute the startup function
                runStartupFcn(app, @startupFcn)

                if nargout == 0
                    clear app
                end
            end

            % Code that executes before app deletion
            function delete(app)

                % Delete UIFigure when app is deleted
                delete(app.UIFigure)
            end
        end
    end
```

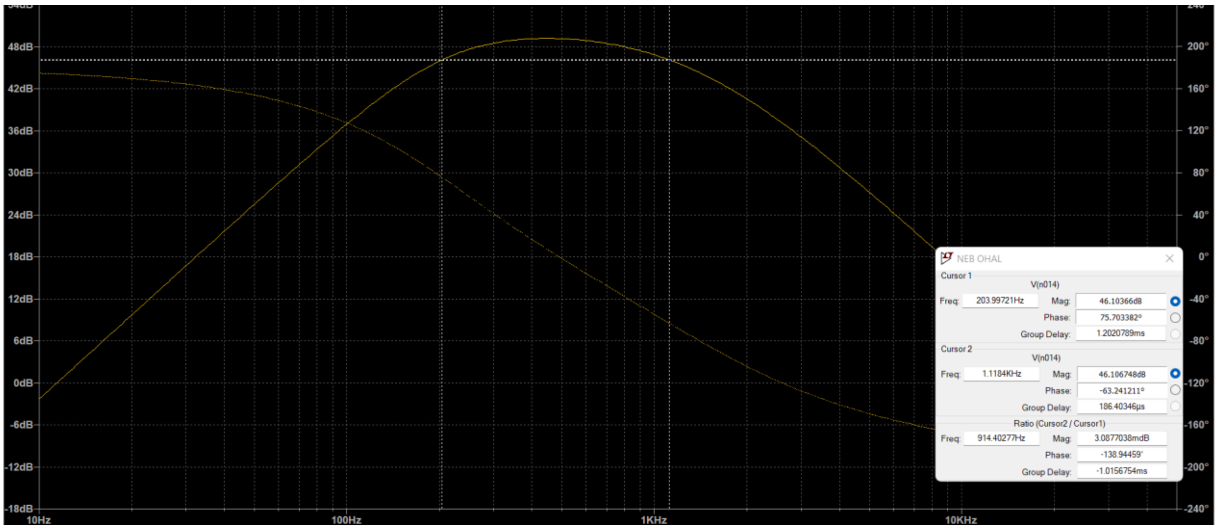
RESULTS

LTSPICE and CIRCUIT PART



```
Cursor 1
                 V(n023)
Freq: 36.818783Hz    Mag:          47.183511dB
                     Phase:        65.497957°
                     Group Delay:  4.5834923ms
Cursor 2
                 V(n023)
Freq: 483.91651Hz    Mag:          47.187816dB
                     Phase:        -65.774806°
                     Group Delay:  350.26531µs
          Ratio (Cursor2 / Cursor1)
Freq: 447.09772Hz    Mag:          4.3048603mdB
                     Phase:        -131.27276°
                     Group Delay:  -4.2332269ms
```

Figure 15: Intervals of the heart filter



Cursor 1
V(n014)
Freq: 203.99721Hz Mag: 46.10366dB
Phase: 75.703382°
Group Delay: 1.2020789ms

Cursor 2
V(n014)
Freq: 1.1184KHz Mag: 46.106748dB
Phase: -63.241211°
Group Delay: 186.40346µs

Ratio (Cursor2 / Cursor1)
Freq: 914.40277Hz Mag: 3.0877038mdB
Phase: -138.94459°
Group Delay: -1.0156754ms

Figure 16: Intervals of the lung filter

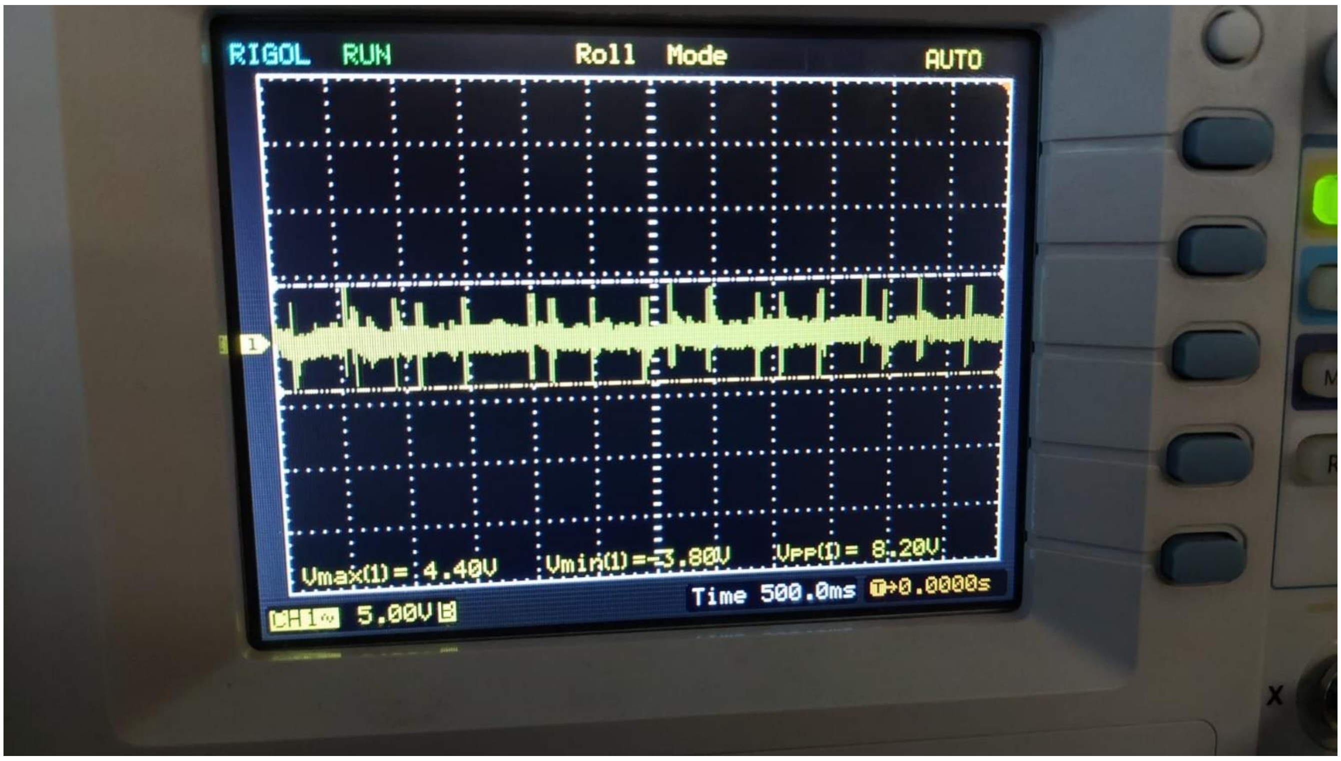


Figure 17: The result of the work of the heart filter

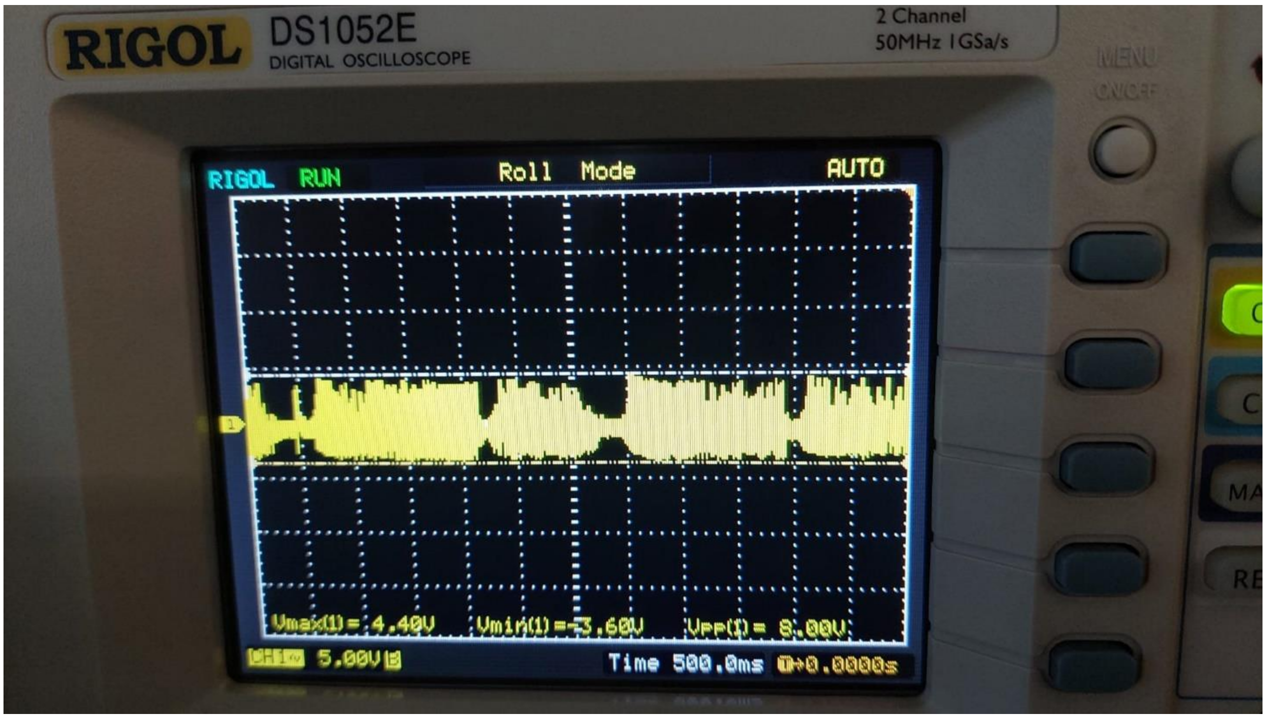
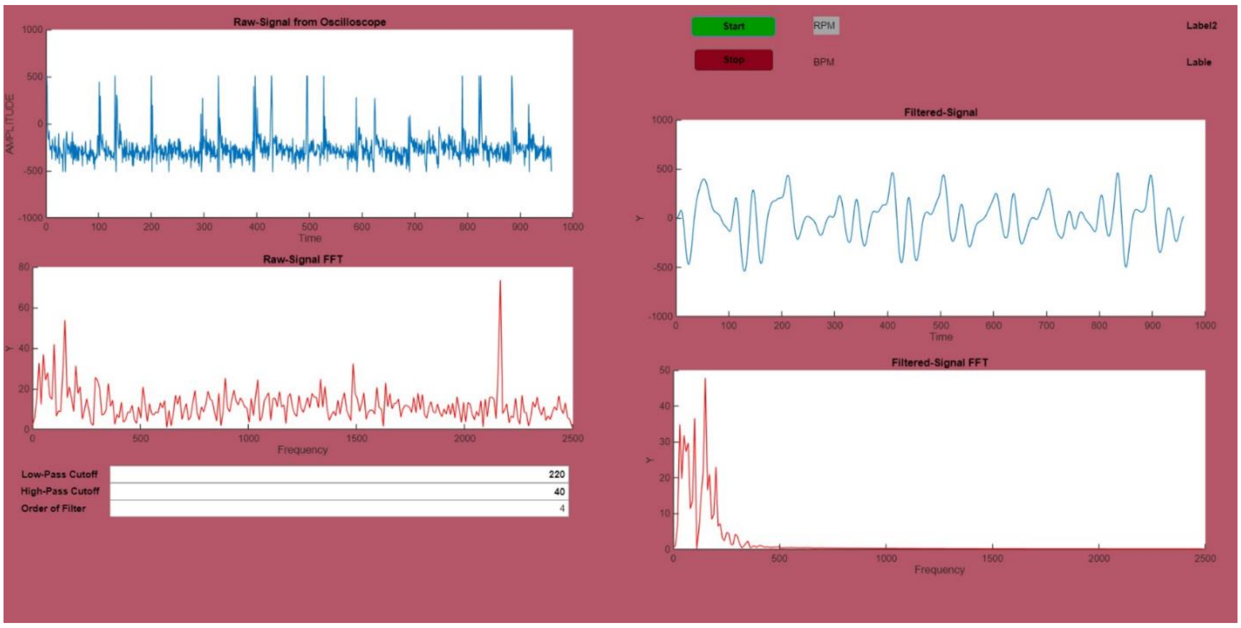


Figure 18: The result of the work of the lung filter

Using the designed stethoscope circuit, heart and lung signals were successfully captured and analyzed in both time and frequency domain. The signals were first observed on an oscilloscope to verify the performance of the circuit. A bandpass filter targeting the frequency range 35-425 Hz was applied for heart sounds and a filter targeting the range 200-1150 Hz for lung sounds. These filters effectively attenuated unwanted frequencies, resulting in a clear and focused signal. Circuit components such as microphone (audio input), amplifier (signal boost) and loudspeaker (audio output) worked as expected. The filter responses and the overall circuit design were first tested and verified using LTspice simulations. After the physical circuit was constructed, real-world tests showed consistency with the simulation results.

MATLAB PART

The project's signal processing stage involved utilizing MATLAB's Graphical User Interface (GUI) functionality. GUIs incorporate interactive elements such as buttons, boxes, text fields, and windows, allowing users to interact with software and execute wanted actions. GUIs are extensively used in applications ranging from operating systems to multimedia tools and scientific software. The data transfer process from Arduino to MATLAB was implemented using Arduino code, ensuring efficient and seamless communication between the two platforms. Firstly a code was written for uploading the communication with Arduino Matlab with the help of Arduino IDE and then the Matlab code part came after. The raw signal from oscilloscope graph helped about the ensuring that at least that part works properly because there were a chance to check if the graphs are the same with oscilloscope. The filtered signal graph is the part that the cleans the noise while transitioning. FFT (Fast Fourier Transform) analysis is a mathematical tool used to transform the signal from the time domain (where it is represented as amplitude versus time) into the frequency domain (where it is represented as amplitude versus frequency).

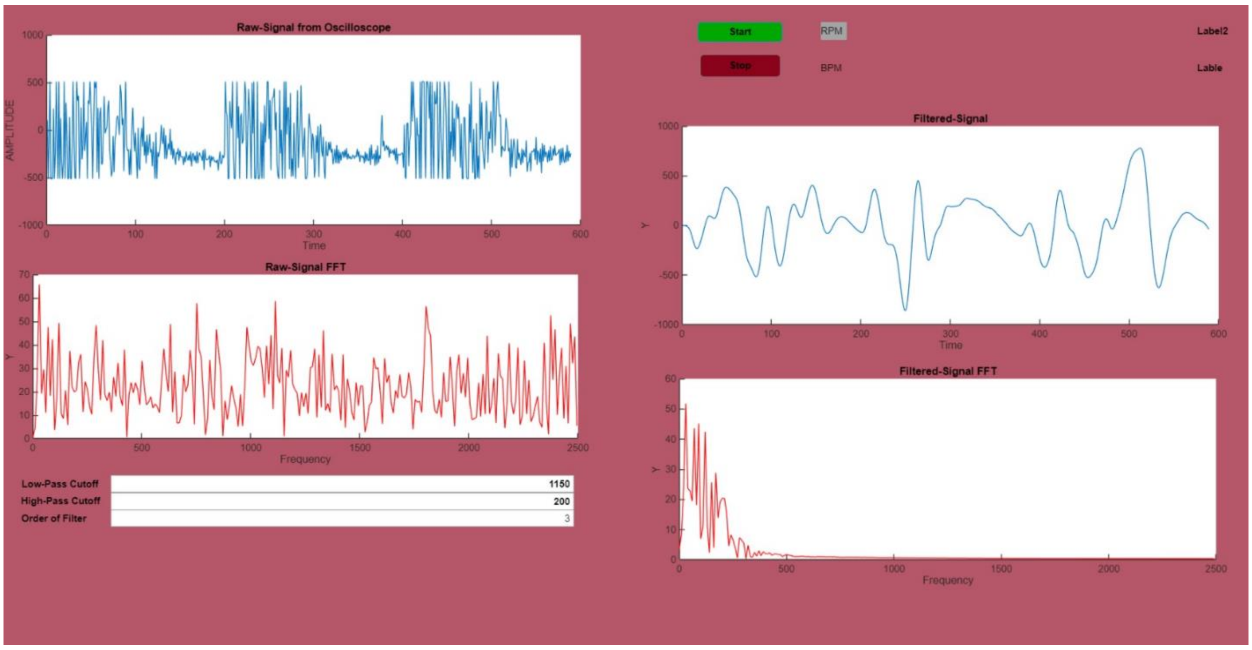


Low-Pass Cutoff 220
High-Pass Cutoff 40
Order of Filter 4

Figure 19: the gui that designed, and the graphs of heart signal as raw signal and it's fft and filtered signal finally it's fft

The image shows the results of signal acquisition and processing performed on a bioelectrical signal, originating from a heart sound detection system. The "Raw Signal from Oscilloscope" represents the unprocessed data directly captured from the microphone, showing significant noise and amplitude variations that obscure the desired heart signal. The "Raw Signal FFT" provides the frequency spectrum of the raw signal, revealing the presence of energy distributed across a broad range of frequencies, including components outside the range typically associated with heart sounds. This indicates the need for filtering to isolate the desired frequency band. After applying a bandpass filter with a low-pass cutoff of 220 Hz and a high-pass cutoff of 40 Hz, the "Filtered Signal" is obtained. This signal demonstrates a periodic pattern, which closely resembles the expected "lub-dub" rhythm of heart sounds. The "Filtered Signal FFT" confirms the effectiveness of the filtering process, as it shows a concentration of energy within the desired frequency range (40–220 Hz) while significantly attenuating noise and unwanted high-frequency components. The comparison between the

FFT of the raw and filtered signals underscores the success of the filter in eliminating extraneous frequencies and enhancing the clarity of the heart sound signal.



Low-Pass Cutoff 1150
High-Pass Cutoff 200
Order of Filter 3

Figure 20: the gui and the graphs of raw signal filtered signal and their fft's.

This image presents the results of signal acquisition and processing performed on a biological signal, an acoustic signal from the lungs. The "Raw Signal from Oscilloscope" graph displays the unprocessed data captured directly from the microphone, showing significant noise and amplitude variations over time. The "Raw Signal FFT" provides the frequency spectrum of the raw signal, revealing energy distributed across a wide frequency range, including both low- and high-frequency components, which indicates the presence of unwanted noise.

The signal was processed using a bandpass filter with a low-pass cutoff frequency of 1150 Hz and a high-pass cutoff frequency of 200Hz. The resulting "Filtered Signal" graph demonstrates a more regular and rhythmic pattern, characteristic of lung sounds. The "Filtered Signal FFT" confirms the effectiveness of the filtering process by showing that energy is concentrated in the desired frequency range (200–1150 Hz) while unwanted high-frequency components are significantly attenuated. The comparison between the raw and filtered signals, both in the time and frequency domains, highlights the effectiveness of the filtering process in isolating and preserving the meaningful components of the lung signal while minimizing noise.

DISCUSSIONS

The development of the electronic stethoscope involved several critical stages, including the design of a microphone circuit, filter circuits tailored for heart and lung sound frequencies, and a speaker circuit for audio output. The filter for heart sounds was designed to pass frequencies in the range of 35-425 Hz, while the lung sound filter operated within 200-1150 Hz. These ranges were chosen based on nominal component values, as the exact values used in the LTspice simulations were unavailable in practice. Consequently, the implemented filter ranges were broader than ideal. This is shown when translating simulation-based designs into real-world circuits, as LTspice operates in an idealized environment where parasitics and tolerances are not factors, unlike physical circuits. Additionally, the choice of broader filter ranges aimed to account for potential variations in component tolerances and ensure that key frequencies were adequately captured.

During the development and testing of the system, an issue with the code prevented the successful calculation and display of BPM (beats per minute) and RPM (respiratory rate per minute) values. Despite obtaining and processing the raw and filtered signals effectively, the malfunction in the coding implementation restricted the extraction of these critical parameters. This limitation highlights the need for further debugging and refinement of the code to ensure that the system can provide accurate and real-time BPM and RPM measurements. Even though sometimes some BPM or RPM results were detected, they were completely wrong.

The FFT analysis of the signals, as shown in the graphs, reveals certain limitations in the quality of the frequency-domain representation. In the "Raw Signal FFT," the energy distribution is inconsistent and lacks a clear concentration within the expected frequency range, indicating the presence of significant noise and interference. Also, the "Filtered Signal FFT" shows some attenuation of unwanted frequencies, but the overall spectral resolution remains suboptimal. This could be attributed to inadequate sampling rates, imperfect filter design, or noise contamination during signal acquisition. The low quality of the FFT results limits the ability to isolate and analyze the desired frequency components accurately, reducing the effectiveness of the signal processing workflow

CONCLUSIONS

As a result, during the project, it was first decided which filter selection would give a more successful result in the construction of an electronic stethoscope and the Sallen - Key filter, the other band pass filter, was preferred. The values of the circuit elements required for the frequency range values selected for the heart and lungs were determined from the Sallen Key band pass application. After the heart and lung circuits were installed, measurements were made and signals were received, the signal and image processing part of the project was started. Since the volt values are within the Arduino operating range, no volt reduction was performed in this step. In the MATLAB application, the analog signals from the oscilloscope were converted into digital signals. Then, for both circuits, the original signal, the FFT of the original signal, the filtered signal and the FFT of the filtered signal were successfully obtained via MATLAB and GUI. All steps were successfully recorded and saved for use in the report. Within the scope of the electronic stethoscope project and what it teaches, this technology has the potential for practical use in many areas such as health monitoring devices, environmental monitoring systems and educational tools, and the real-life applications of the project are quite wide. Beyond improving our technical knowledge, this project allowed us to experience the satisfaction of using what we learned in the real world.

REFERENCES

[1] Mba, J. N. P. (2024, May 21). *All you need to know about electronic stethoscopes*. Digital Health Central. https://digitalhealthcentral.com/all-you-need-to-know-about-electronic-stethoscopes/

[2] Admin-Ef. (2024, August 18). *Sallen-Key Filter Circuit: A Building Block for Filters*. Electronics Calculations. https://electronics-formulas.com/sallen-key-filter-circuit-a-building-block-for-filters/

[3] Technology, E. (2020, May 2). *Analog to Digital Converter (ADC) – Block Diagram, Factors & Applications*. ELECTRICAL TECHNOLOGY. https://www.electricaltechnology.org/2019/02/analog-to-digital-converter-adc.html

[4] Changpuak, A. C. F. A. (n.d.). *Online Sallen Key Bandpass Filter Designer*. www.changpuak.ch. https://www.changpuak.ch/electronics/Sallen_Key_Bandpass_light.php

[5] *Filter Design and Analysis*. (n.d.). http://sim.okawa-denshi.jp/en/Fkeisan.htm