ANKARA UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENTS OF

BIOMEDICAL ENGINEERING,

COMPUTER ENGINEERING

AND

ELECTRICAL AND ELECTRONICS

ENGINEERING

MULTIDISCIPLINARY PROJECT WORK

2023-2024 SPRING SEMESTER

DESIGN AND TEST OF A PHONOCARDIOGRAM SYSTEM

Ankara University, Faculty of Engineering, Department of Biomedical, Computer and Electrical and Electronics Engineering	MULTIDISCIPLINARY PROJECT WORK EVALUATION REPORT	Semester: 2023-2024

TEAM			
Number	1	**Name**	TechHeartAttack

STUDENTS		
Number	**Name and Surname**	**Course Code**
20290024	Ahmad Hisham İzzat	BME324-A
20290893	Mohammad Abdelrazeq Issa Alsayed Ahmad	BME324-A
20290431	Sena Tabakoğlu	BME324-A
21290281	Zeynep Sıla Yurtalan	COM2044-B
21290704	Melike Ayaz	COM2044-B
21290720	Görkem Arslan	COM2044-B
22290338	Eren İşlek	COM2044-B
21290420	Talha Eken	COM2044-B
22290773	Rosha Abedini Karahroudi	EEE2222-A
21290756	Ayşe Selin Kaplan	EEE2222-A
20290849	Saltuk Buğra Acar	EEE2222-A

		Weights	Grade
Evaluation			
1	Literature Work	10	
2	Project Management	10	
3	Stethoscope design, tests, performance	10	
4	Electronic Circuit design, tests, performance	10	
5	Monitoring Software design, tests, performance	10	
6	Tests of the system	10	
7	An example of signal processing using the collected data	10	
8	Possible standards used or can be used for design	10	
9	Encountered difficulties related to multidisciplinary project work	10	
10	Format of the report	10	
	Total	**100**	

Evaluator(s):

ABSTRACT

A Phonocardiogram System is a valuable tool in cardiology used to record and analyze the sounds produced by the heart. This system consists of a microphone sensor placed on the chest to capture heart sounds, which are then converted into electrical signals. These signals are processed and displayed graphically, allowing healthcare professionals and researchers to visualize and interpret various cardiac events, such as heart murmurs, valve disorders, and abnormal rhythms. Understanding the principles behind Phonocardiogram Systems provides students with insights into cardiac physiology and diagnostics.

CONTENTS

1. INTRODUCTION

A phonocardiogram (PCG) is acquisition of acoustic waves caused by the mechanical activities of the heart and reflect cardiovascular pathological conditions. Segmentation of the PCG signal to identify the first (S1) and second (S2) heart sounds is a crucial step in automated analysis of the PCG signal and diagnosing heart disorders. Classification of pathological murmurs requires accurate localization of different parts of the PCG signal. PCG segmentation can also help to analyze the signal in detail, getting further information about each component, and assessing the presence or location of murmurs. Overall, correct segmentation of heart sounds and extraction of S1 and S2 can lead to an automated system for diagnosing cardiac defects.

The Phonocardiogram System is like a special tool that helps doctors listen to your heart in a detailed way. It works by using a microphone-like sensor that listens to your heart's sounds, like a stethoscope but even better, this sensor picks up the sounds, turns them into electrical signals, and then shows them on a graph or chart. This helps doctors understand if your heart is working as it should or if there might be any problems, like with your heart valves or if you have a murmur.

A sound waveform is a 2-dimensional plot of sound intensity (or loudness) versus time. It shows the intensity of a sound on the vertical (Y) axis and elapsed time on the horizontal (X) axis. A larger deflection magnitude indicates a louder sound, and a smaller deflection indicates a softer sound. Usually, the units on the Y-axis are arbitrary, although sometimes they may be measurements of loudness such as decibels. The units on the X-axis are usually seconds.

A phonocardiogram (PCG) is a special sound waveform that plots in high fidelity the intensity of heart sounds over time. The two fundamental heart sounds, called "S1" and "S2", are shown on a PCG as large magnitude deflections occurring one after the other, with S1 first. They correspond to the "lub" and "dub" people hear through a stethoscope. Depending upon where on the body the sounds were captured, the S1 deflection may be larger, S2 may be larger, or the sounds may be the same size (loudness). At other times the heart is silent, and so the PCG will be flat, without deflection, resting at the baseline.

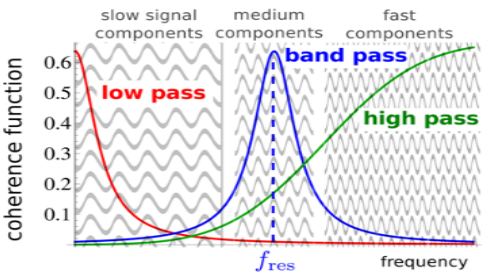


Figure (3): Filters behaviors.

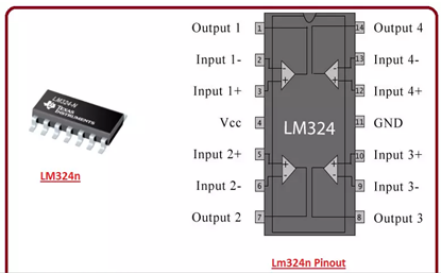


Figure (3): internal block diagram instrumentation amplifier.

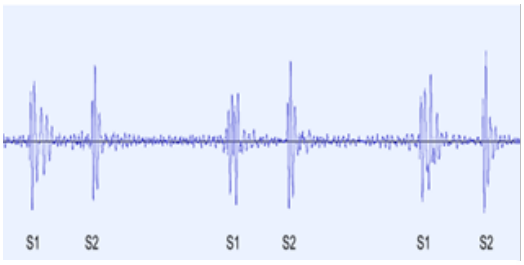


Figure (1): A phonocardiogram (PCG) showing the power (intensity) of sound on the Y-axis and time on the X-axis. S1 is the first heart sound; S2 is the second heart sound.

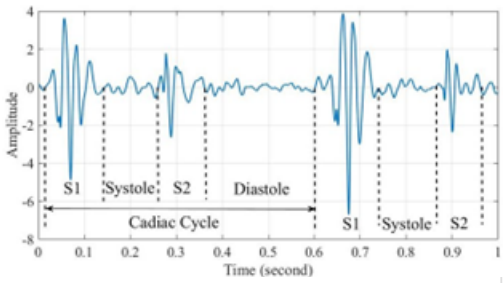


Figure (2): PCG Signal.

Phonocardiography (PCG) offers several benefits:

- Real-time traces of heartbeats and movements provide detailed insights into cardiac conditions.
- The passive method is safe and suitable for long-duration monitoring.
- It is a cost-effective diagnostic method

Drawbacks or disadvantages of Phonocardiography (PCG) include:

- Bulky and obstructive existing microphones used in phonocardiography.
- Risk of infection to patients undergoing surgery due to the requirement for a sterile field.
- Patients may experience frequent disturbance during testing.

1.i. PHONOCARDIOGRAM LITERATURE

Phonocardiograms (PCGs) are graphical representations of the sounds produced by the heart during each cardiac cycle. This paper provides a simplified overview of PCGs, aimed at facilitating understanding among students and enthusiasts in the field of cardiology. PCGs are recorded using specialized sensors placed on the chest, which capture the acoustic signals generated by the heart's mechanical activity. These signals are then converted into visual graphs or charts, allowing for detailed analysis of heart sounds. PCGs play a crucial role in diagnosing various cardiac conditions, including murmurs, valve disorders, and abnormal rhythms. By deciphering the information presented in PCGs, healthcare professionals can make informed decisions regarding patient care and treatment strategies. This paper highlights the significance of PCGs in cardiology practice and underscores the importance of understanding their interpretation for effective clinical management.

1.ii. PROJECT MANAGEMENT PLAN:

a) TEAM DETAILS:

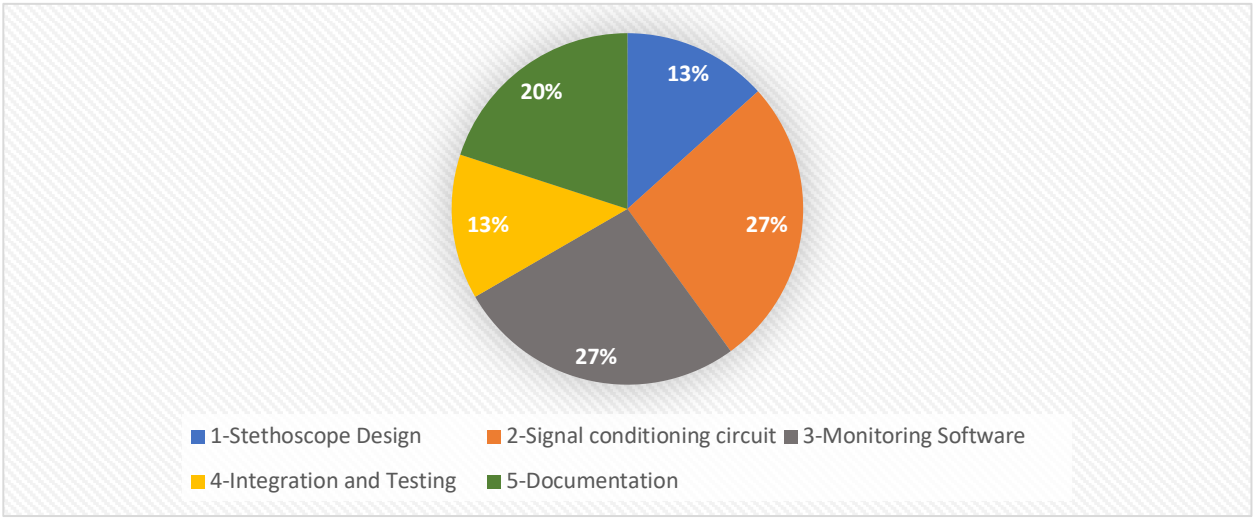
- **BIOMEDICAL ENGINEERING STUDENTS:**
  - Ahmet Hisham İzzat Ahmad (ahmed.aqell26@gmail.com)
  - Mohammad Abdelrazeq Issa Alsayed Ahmad (mohammad.m2002mm@gmail.com)
  - Sena Tabakoğlu (sena.tblgl25@gmail.com) | (Team Leader)
- **COMPUTER ENGINEERING STUDENTS:**
  - Zeynep Sıla Yurtalan (zsyurtalan@gmail.com)
  - Melike Ayaz (mlkayz827@gmail.com) | (Team Leader) | (Team Director)
  - Eren İşlek (erenislek63@gmail.com)
  - Görkem Arslan (grkma404@gmail.com)
  - Talha Eken (talhaeken157142@gmail.com)
- **ELECTRICAL AND ELECTRONICS STUDENTS:**
  - Rosha Abedini Karahroudi (Rocha.abedini@gmail.com)
  - Saltuk Buğra Acar (saltuk9c@gmail.com) | (Team Leader)
  - Ayşe Selin Kaplan (kaplanayseselin@gmail.com)

b) ROLES AND RESPONSIBILITIES:

- **BME Students:**
  - Took materials for the project and initiated the project design.
  - Assigned as the first designer of the project.
- **EEE Students:**
  - Constructed and soldered the signal conditioning circuits of the PCG.
  - Contributed to the architecture of connection between the signal conditioning circuits, the stethoscope and the microphone input.
  - Provided further explanation and understanding of signal conditioning circuits using circuit simulators.
  - Measured separate components of circuits via multimeters.
- **CompEng Students:**
  - Developing a monitoring software on java that creates GUI by taking data from audio files and developing monitoring software on java that creates GUI by receiving sound data from the prepared real time sound card

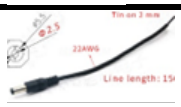
c) TIME CHART:

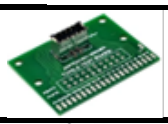
Task	Team name and Details	Percentage
Stethoscope design:	BME The microphone is attached to the stethoscope case by removing the top cover and threading the cables through the gap, then a glove is stretched over the underside where the microphone is positioned and secured with a rubber band.	13% (2 days)
Signal conditioning circuit:	EEE The individual electronic components are soldered to the board and each component's value is measured via utilizing a basic multimeter. The simulation software Proteus was used to illustrate and evaluate the progress of the construction of the signal conditioning circuit.	27% (4 days)
Monitoring software:	CompEng The requisite libraries have been installed. The audio data from the computer microphone has been read with Java, and the graphical output has been displayed on the screen at a rate that is easily comprehensible via a graphical user interface (GUI). Subsequently, the system was tested with a circuit.	27% (4 days)
Integration and testing:	All Departments The operational status of the pertinent hardware and software structures was evaluated. Any identified errors were duly rectified.	13% (2 days)
Documentation:	All Departmens In accordance with the instructions provided, the report draft was divided into sections. Subsequently, each member of the team proceeded to complete the sections relevant to their respective area of expertise. Subsequently, the common sections were drafted in collaboration with the online OneDrive file.	20% (3 days)



2. MATERIALS AND METHODS

- **MATERIALS (Phonocardiogram Project Components)**

Name of Materials	***Value***	***Required Number of Components for One Device***
1. Resistor	*1k* 	2
2. Resistor	*10k* 	1
3. Resistor	*68k*	1
4. Resistor	*150k* 	1
5. Resistor	*470k*	1
6. Resistor Pot	*50k* 	2
7. Capacitor	*100n* 	2
8. LM324N Chip		1
9. Header Pin		4
10. 2 Pin Power Connector 3.5		2
11. 2 Pin Power Connector 2.5		2
12. 9 Volt Battery Connector		2
13. Male to Male Mini jack stereo cable		1

14. 14 pin DIP chip Socket		1
15. BL50A Speaker		1
16. Peaker Housing		1
17. PCB Board		1
18. Glove latex without powder		1
19. Package rubber		1
20. 9 Volt Battery		2
21. 25 cm 2 wire cable		1

METHODS:

- Firstly, the project materials were obtained, and then the functions required of the Phonocardiogram System were identified. These included the accurate capture of heart sounds.
- Secondly, a plan was devised for the construction of the system, including the selection of sensors and electronic components.
- Subsequently, the initial design of the project was formulated.
- Subsequently, the system was assembled in accordance with the design, with particular attention paid to ensuring correct connectivity. Subsequently, the signal conditioning circuits of the PCG were constructed and soldered.
- The system was subjected to a series of tests to ascertain its functionality and to identify any potential issues. These tests included the recording of heart sounds.
- Subsequently, the system's performance was optimised based on the outcomes of the aforementioned tests.
- Finally, the system was evaluated to ascertain that it met the requisite specifications and was ready for operational use.

2.i. BLOCK DESIGN

The goal of a Phonocardiogram (PCG) is to record and analyze the sounds produced by the heart during its beating cycle. These sounds, often referred to as heart sounds, include the well-known "lub-dub" sounds associated with the closing of the heart valves. By analyzing these sounds, medical professionals can diagnose various heart conditions and monitor heart health.

Sound Sensor: This sensor, placed on the chest, detects the sounds produced by the heart. It converts these acoustic signals into electrical signals.

Amplifier (471 times): The initial electrical signal from the sound sensor is usually very weak. The first amplifier increases the signal's amplitude by 471 times, making it strong enough for further processing.

Bandpass Filter: This filter allows only the frequencies within a certain range to pass through, typically the range associated with heart sounds (around 20 to 1000 Hz). It removes the noise and irrelevant frequencies outside this range, ensuring the signal is clean and focused on the heart sounds.

Amplifier (50 times): After filtering, the signal may still require further amplification to be suitable for analysis. The second amplifier boosts the signal by another 50 times.

Java Coding: The combination of Java Swing for the user interface, AWT for graphics rendering, and “javax.sound.sampled” for audio processing creates an application capable of visualizing both offline and real-time audio signals. This project demonstrates the effective use of Java's libraries to handle tasks such as audio visualization and user interaction.

2.ii. STETHOSCOPE

1. We placed carefully and secured a high-quality digital microphone within a custom-designed plastic enclosure. then we Ensured the microphone was positioned optimally to capture the desired audio signals.

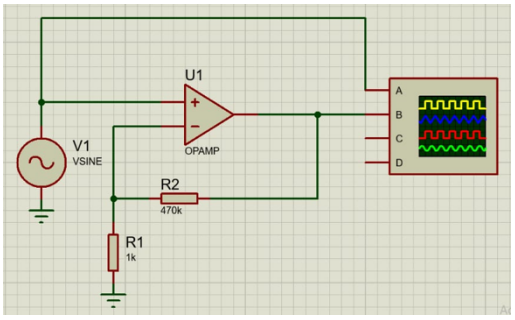
2. After that, we covered the plastic enclosure completely with a specialized acoustic insulation glove. This glove helps to minimize the impact of external noise and vibrations on the audio recordings.

3. We routed the necessary cables for transmitting the audio signal from the microphone to the data processing unit and secured the cables neatly and minimize any potential interference or noise pickup along the signal path.

2.iii. SIGNAL CONDITIONING CIRCUIT

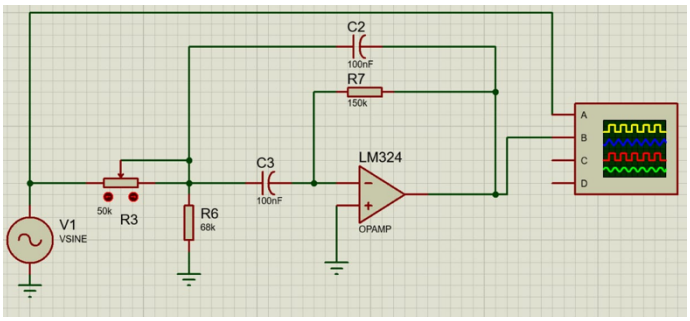
Signal conditioning is an electronic circuit that manipulates a signal in such a way as to prepare it for the next stage of processing.

1. Pre amplification: The raw electrical signals from the heart are typically very weak. They are often in the millivolt range. To amplify these weak signals while maintaining a high signal-to-noise ratio (SNR), the first stage of signal conditioning is a preamplifier. The pre-amplifier can use low noise operational amplifiers (op-amps) that are configured in a differential amplifier configuration to reject the common mode noise.



$$V_0 = \left(1 + \frac{R_f}{R_1}\right) V_1$$

2. Bandpass Filtering: Heart sounds typically occur in specific frequency bands (e.g. 20 Hz to 200 Hz) and filtering out unwanted frequencies can improve the clarity of the recorded signal. To selectively amplify signals within this frequency range while attenuating noise outside this range, bandpass filters are used. Active or passive filter configurations, such as multi-stage RC filters or active filter ICs, can be used to implement the bandpass filter.



$$q = \frac{f_r}{BW} \text{ where, } q > 0,5$$

Given $C = C1 = C2$,

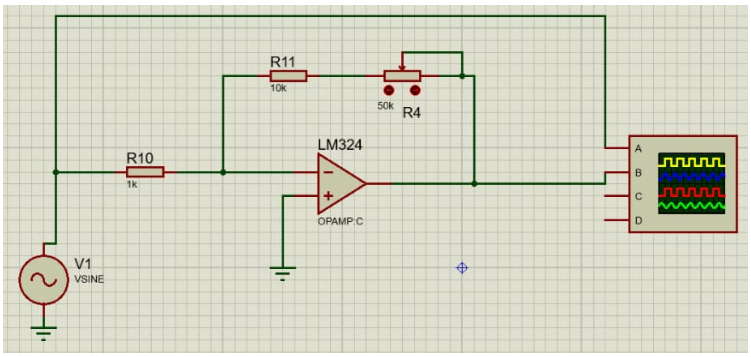
$$k = 2\pi f_0 C$$

$$R1A = Q\frac{H}{k}$$

$$R1B = \frac{Q}{(2Q^2 - H)k}$$

$$R2 = \frac{2Q}{k}$$

3. Inverse amplification: After filtering, the signal may be further amplified to ensure that it is within the optimum range for the analogue-to-digital converter (ADC). This gain stage can be adjusted to compensate for variations in signal strength between patients or recording conditions.



$$V_0 = -\frac{R_f}{R_1}V_1$$

Other signal conditioning circuits for PCG: Anti-aliasing filtering, Impedance matching, Isolation and Grounding, DC Offset removal, Temperature Compensation.

2.iv. MONITORING SOFTWARE

In this project, we utilized Java Swing to develop the graphical user interface, which includes the creation of buttons, tabs, and the main application window.

1. **JFrame:** This class was employed to create the primary application window. The JFrame served as the main container for the application. Additionally, it allowed for the integration of a tabbed pane that holds different panels for real-time and offline signal visualization.

```
public class SignalVisualizer extends JFrame {
    public SignalVisualizer() {
        setTitle("Phonocardiogram Signal Visualizer");
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        JTabbedPane tabbedPane = new JTabbedPane();

        realTimeSignalPanel = new RealTimeSignalPanel();
        offlineSignalPanel = new OfflineSignalPanel();

        tabbedPane.addTab("Real-Time Signal", realTimeSignalPanel);
        tabbedPane.addTab("Offline Signal", offlineSignalPanel);

        add(tabbedPane, BorderLayout.CENTER);
    }
}
```

2. **JPanel:** To organize various sections and controls within the window, we used the JPanel class. For instance, in the offline signal panel, JPanel was used to hold control buttons and display areas, allowing for a clear separation of different functional areas within the application.

```
public class OfflineSignalPanel extends JPanel implements ActionListener {
    private JPanel controlPanel;

    public OfflineSignalPanel() {
        setPreferredSize(new Dimension(800, 400));
        setLayout(new BorderLayout());

        controlPanel = new JPanel();
        controlPanel.add(signal1Button);
        controlPanel.add(signal2Button);
        controlPanel.add(signal3Button);
        controlPanel.add(resetButton);

        add(controlPanel, BorderLayout.NORTH);
        add(new SignalDisplayPanel(), BorderLayout.CENTER);
    }
}
```

3. **JButton and JRadioButton:** These components were implemented to enable user interaction. JButtons were used for general action triggers, such as resetting the application state, while JRadioButtons allowed users to select specific signal options. These interactive elements were essential for providing a responsive user experience.

```
private JRadioButton signal1Button, signal2Button, signal3Button;
private JButton resetButton;

signal1Button = new JRadioButton("Christians Heart");
signal2Button = new JRadioButton("Dong3 Heart");
signal3Button = new JRadioButton("Dong3 HeartORG");
resetButton = new JButton("Reset");
```

To generate graphics from sound and handle user-triggered events, we utilized **Java AWT (Abstract Window Toolkit) :**

1. **BufferedImage and Graphics2D:** These classes were used for drawing the waveform. BufferedImage provides a drawing surface, while Graphics2D facilitates drawing on this surface. This combination allowed us to render the waveform accurately.

```
private BufferedImage image;
private Graphics2D g2d;

image = new BufferedImage(800, 400, BufferedImage.TYPE_INT_RGB);
g2d = image.createGraphics();
```

2. **Color:** The Color class was used to apply colors to the waveform and the background. This enabled clear visualization of the waveform against a contrasting background.

```
g2d.setColor(Color.BLACK);
g2d.fillRect(0, 0, image.getWidth(), image.getHeight());
g2d.setColor(Color.BLUE);
```

3. **addActionListener:** This method was employed to assign event handlers to buttons. It allowed us to define actions that should be taken when specific buttons are pressed.

```
// Reset butonu
resetButton = new JButton("Reset");
resetButton.addActionListener(e -> resetSignal());
```

```
// Signal seçeneği butonları
signal1Button = new JRadioButton("Christians Heart");
signal2Button = new JRadioButton("Dong3 Heart");
signal3Button = new JRadioButton("Dong3 HeartORG");

signal1Button.addActionListener(this);
signal2Button.addActionListener(this);
signal3Button.addActionListener(this);
```

```
// Reset signal butonuna basıldığında gerçekleşecek fonksiyon:
private void resetSignal() {
    g2d.setColor(Color.BLACK);
    g2d.fillRect(0, 0, image.getWidth(), image.getHeight());
    repaint();
}
```

```
// Signal butonlarına basıldığında gerçekleşecek fonksiyon:
@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == signal1Button) {
        selectedFile = "resources/Christians_Heart.wav";
    } else if (e.getSource() == signal2Button) {
        selectedFile = "resources/Dong3_Heart.wav";
    } else if (e.getSource() == signal3Button) {
        selectedFile = "resources/Dong3_HeartORG.wav";
    }
    loadAndDrawSignal();
    repaint();
}
```

2.v. SIGNAL PROCESSING EXAMPLE METHODS

The javax.sound.sampled library is used for both reading audio files and capturing real-time audio data. It also manages errors using exceptions such as UnsupportedAudioFileException and LineUnavailableException.

- **AudioInputStream**: Utilized for reading audio files.
- **AudioSystem**: Provides access to the audio system.
- **AudioFormat**: Defines the format of the audio data.
- **UnsupportedAudioFileException**: Catches errors when an unsupported audio file format is encountered.

```
import javax.sound.sampled.*;

private void loadAndDrawSignal() {
    File file = new File(selectedFile);
    try {
        AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(file);
        AudioFormat format = audioInputStream.getFormat();
        byte[] bytes = audioInputStream.readAllBytes();
        int[] samples = new int[bytes.length / 2];
        for (int i = 0; i < bytes.length; i += 2) {
            samples[i / 2] = (bytes[i] << 8) | (bytes[i + 1] & 0xFF);
        }
        drawWaveform(samples);
    } catch (UnsupportedAudioFileException | IOException e) {
        e.printStackTrace();
    }
}
```

- **TargetDataLine**: Used for capturing real-time audio data.
- **DataLine.Info**: Provides information for the TargetDataLine.
- **LineUnavailableException**: Thrown when a suitable audio line cannot be found.

```
import javax.sound.sampled.*;

public class RealTimeSignalPanel extends JPanel implements Runnable {
    private static final int BUFFER_SIZE = 4096;
    private TargetDataLine line;
    private Thread thread;

    public RealTimeSignalPanel() {
        setPreferredSize(new Dimension(800, 400));
        startCapture();
    }

    private void startCapture() {
        try {
            AudioFormat format = new AudioFormat(44100, 16, 1, true, true);
            DataLine.Info info = new DataLine.Info(TargetDataLine.class, format);
            line = (TargetDataLine) AudioSystem.getLine(info);
            line.open(format);
            line.start();
            thread = new Thread(this);
            thread.start();
        } catch (LineUnavailableException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void run() {
        byte[] buffer = new byte[BUFFER_SIZE];
        while (thread != null) {
            int bytesRead = line.read(buffer, 0, buffer.length);
            if (bytesRead > 0) {
                int[] samples = new int[bytesRead / 2];
                for (int i = 0; i < bytesRead; i += 2) {
                    samples[i / 2] = (buffer[i] << 8) | (buffer[i + 1] & 0xFF);
                }
                drawWaveform(samples);
                repaint();
            }
        }
    }
}
```

The calculateBPM function takes sound level values and calculates the mean and standard deviation. It then determines a threshold value using the formula ($mean + 1.5 * standard\ deviation$). Peaks are identified as sound levels exceeding this threshold and occurring at least 0.5 seconds apart. The number of peaks is counted, and the Beats Per Minute (BPM) is determined based on the number of peaks over the duration of the sound.

```
    private int calculateBPM(int[] samples, int sampleRate) {
        int minPeakDistance = sampleRate / 2; // Minimum distance between peaks
(0.5 seconds)
        int peakCount = 0;
        int[] peakIndices = new int[samples.length];

        double mean = 0;
        for (int sample : samples) {
            mean += sample;
        }
        mean /= samples.length;

        double stdDev = 0;
        for (int sample : samples) {
            stdDev += Math.pow(sample - mean, 2);
        }
        stdDev = Math.sqrt(stdDev / samples.length);

        double threshold = mean + 1.5 * stdDev;

        for (int i = 1; i < samples.length - 1; i++) {
            if (samples[i] > threshold && samples[i] > samples[i - 1] && samples
[i] > samples[i + 1]) {
                if (peakCount == 0 || (i - peakIndices[peakCount - 1] > minPeakD
istance)) {
                    peakIndices[peakCount++] = i;
                }
            }
        }

        peakCountLabel.setText("Peaks: " + peakCount);

        if (peakCount < 2) {
            return 0;
        }

        double totalIntervals = 0;
        for (int i = 1; i < peakCount; i++) {
            totalIntervals += (peakIndices[i] - peakIndices[i - 1]) / (double) s
ampleRate;
        }

        double averageInterval = totalIntervals / (peakCount - 1);
        return (int) (60 / averageInterval);
    }
```

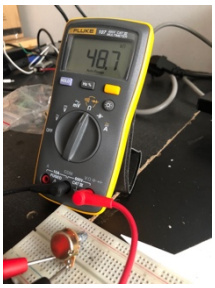
2.vi. POSSIBLE STANDARDS USED OR CAN BE USED FOR DESIGN

- **Signal Quality:** The design should prioritize high signal-to-noise ratio to accurately capture heart sounds amidst background noise. This involves careful selection of sensors, amplifiers, and filters to minimize interference.
- **Frequency Response:** The system should have a wide frequency response to capture both low-frequency components (such as S1 and S2 heart sounds) and higher frequency components (such as murmurs and clicks).
- **Dynamic Range:** The PCG system should have a wide dynamic range to accommodate the varying intensity of heart sounds across different patients and conditions.
- **Sampling Rate:** Adequate sampling rate is crucial for capturing fast-changing heart sounds accurately. The system should typically sample at rates higher than the Nyquist frequency of the highest frequency component of interest in heart sounds.
- **Anti-Aliasing Filters:** Proper anti-aliasing filters should be employed to prevent aliasing artifacts during analog-to-digital conversion.
- **Portability and Comfort:** For wearable or portable PCG devices, comfort and ease of use are essential design considerations to ensure patient compliance with long-term monitoring.
- **Data Storage and Transmission:** The system should incorporate efficient data storage and transmission capabilities, especially for applications involving remote monitoring or telemedicine.
- **Power Consumption:** Low power consumption is crucial for battery-operated devices to enable long-term monitoring without frequent recharging.
- **Calibration and Maintenance:** The system should be calibrated regularly to maintain accuracy, and it should be designed for easy maintenance and servicing.
- **Regulatory Compliance:** Compliance with relevant regulatory standards, such as FDA regulations in the United States or CE marking in Europe, is essential for ensuring safety and effectiveness.

3. RESULTS

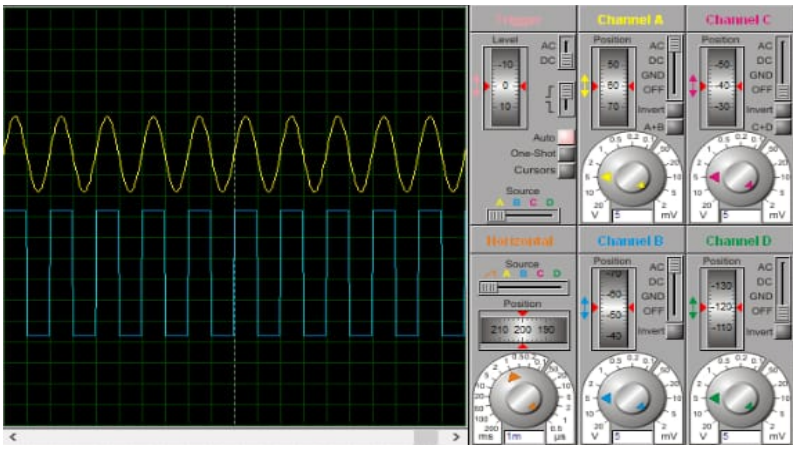
3.i. SEPERATE TESTS OF COMPONENTS

The circuitry was subjected to tests and measurements using a simple multimeter, as demonstrated in the example below.



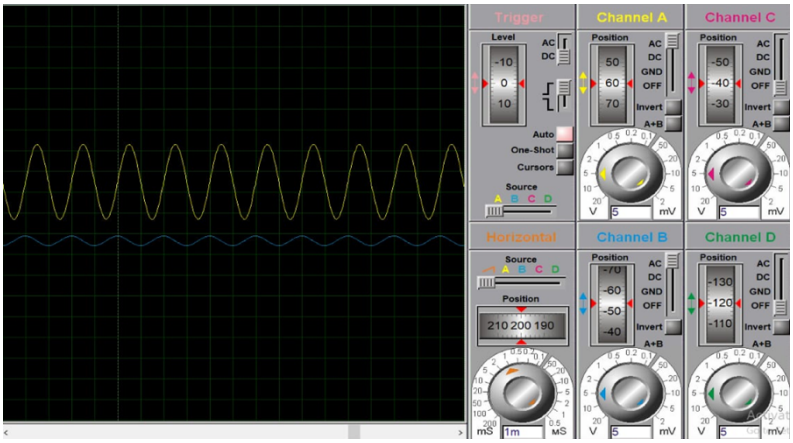
The amplitude of AC source used in constructing the Phonocardiogram is 9V and the frequency is between 50hz to 500hz. The VCC of OP amp is 4.5V and -4.5V.

Simulation derived Oscilloscope results for Pre-Amp:



$$V_0 = \left(1 + \frac{463.2k\Omega}{0.98k\Omega}\right)(9V)$$
$$V_0 = 4262.9V$$

Simulation derived Oscilloscope results for bandpass filtering:



Transfer function according to calculations:

$$G(s) = \frac{-2023.1050738059}{s^2 + 1353.3305057748s + 2372191.4273322}$$

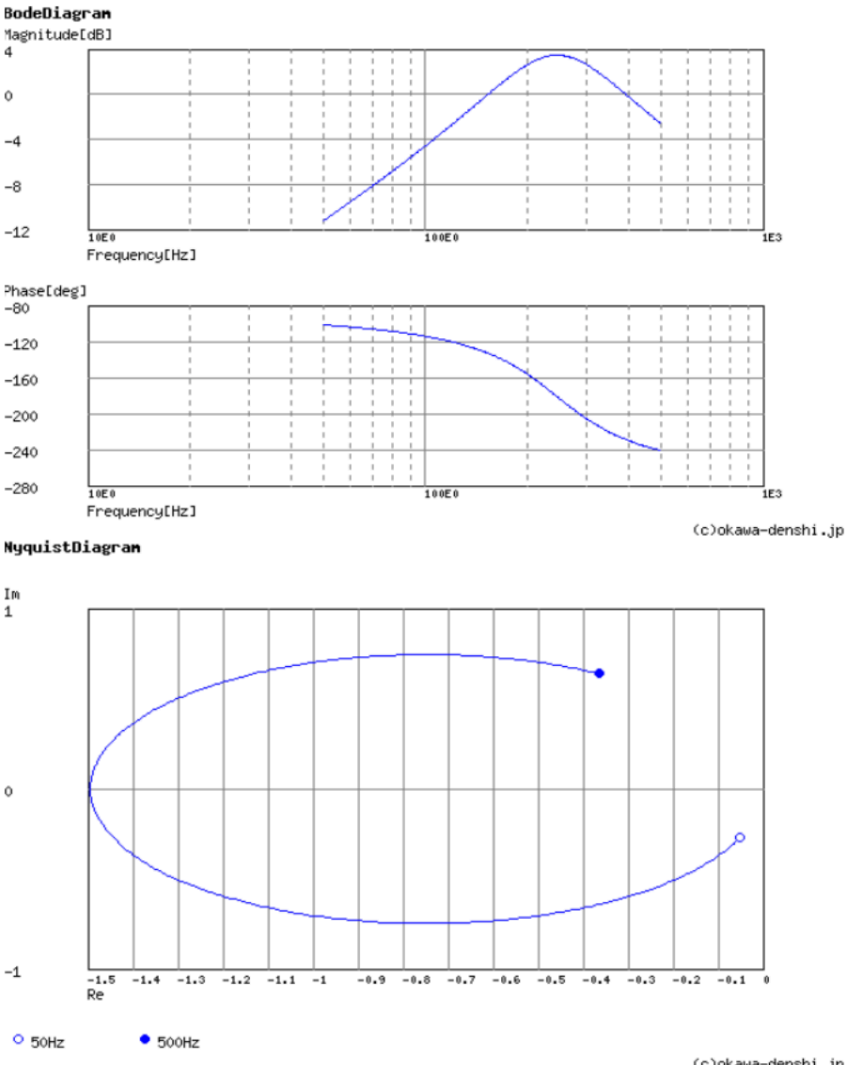
Center frequency:
$f_0 = 245.1291716848$[Hz]

Gain at f0:
G_{pk} = -1.4949083503055[times] (3.492291355862)[dB]

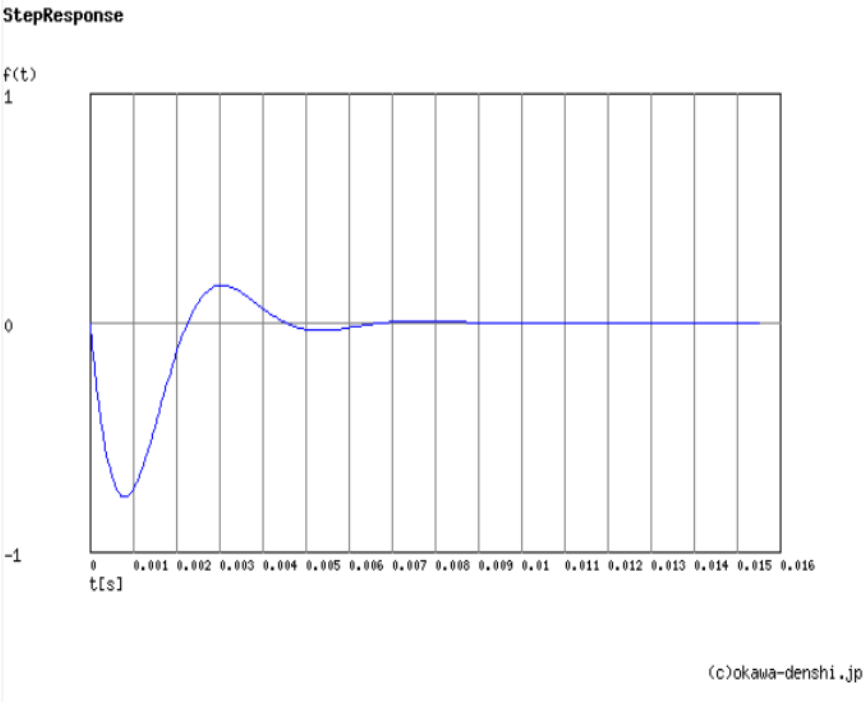
Quality factor:
Q = 1.1380752915263

Damping ratio:
ζ = 0.43933824389549

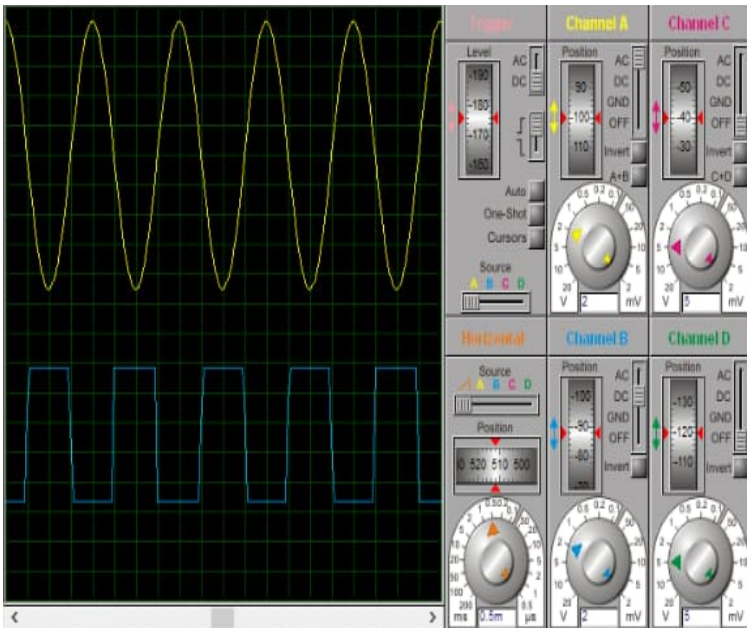
Frequency Analysis between 50 and 500 Hz:



Transient Analysis:



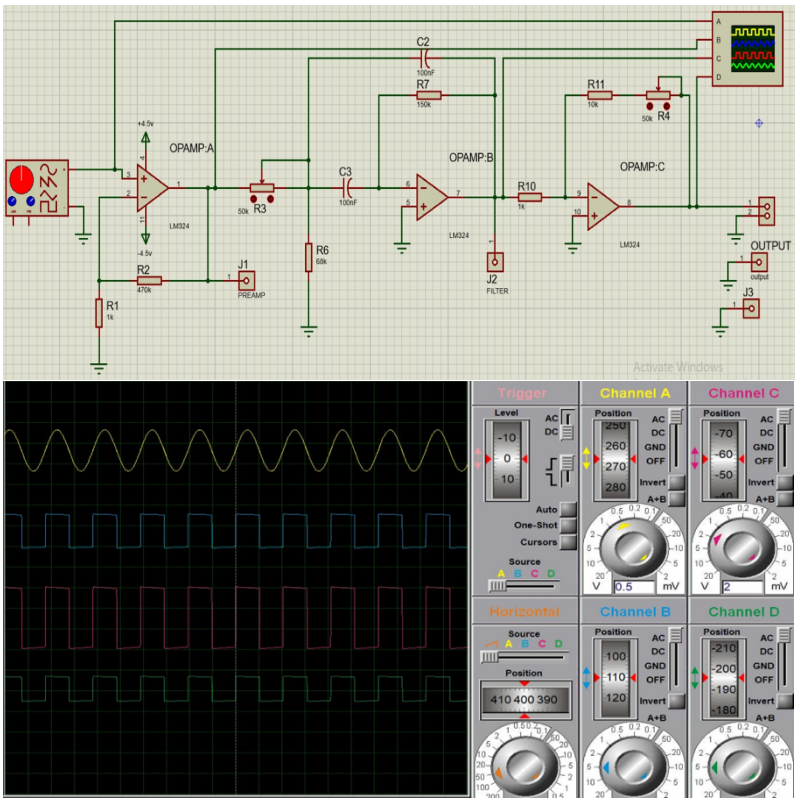
Simulation derived Oscilloscope results for inverse amplification:



$$V_0 = -\frac{48.7k\Omega + 9.93k\Omega}{0.99k\Omega}(9V)$$

$$V_0 = -533V$$

Results for signal conditioning circuit:



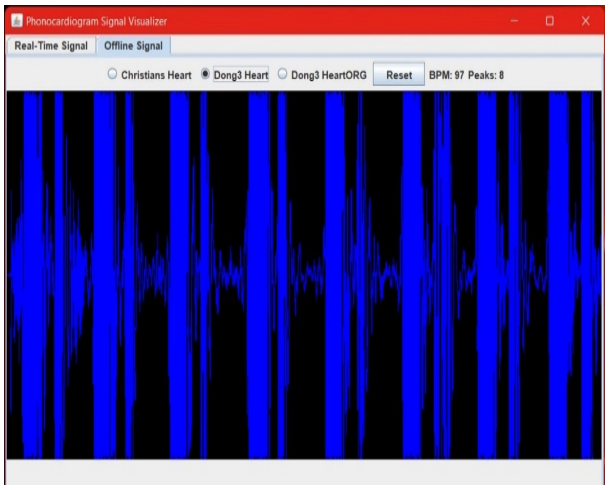
3.ii. SYSTEM TESTS

System tests were conducted to assess the performance and accuracy of our phonocardiogram system. These tests aimed to ensure the seamless operation of both hardware and software components. During the design of the stethoscope, the materials used were flawless and checked for errors, and the functionality of the cables was also tested.

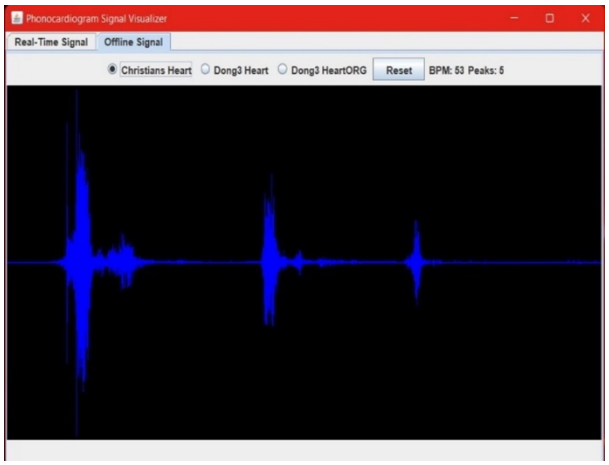
System Tests Include:

- **Signal Acquisition Test:** Verification of accurate capture and digital conversion of heart sound signals from the stethoscope.
- **Visualization Test:** Ensuring correct and prompt real-time display of heart sounds in the software interface.
- **Filtering and Processing Test:** Confirmation of the effectiveness of the applied filters and processing algorithms in enhancing signal clarity.

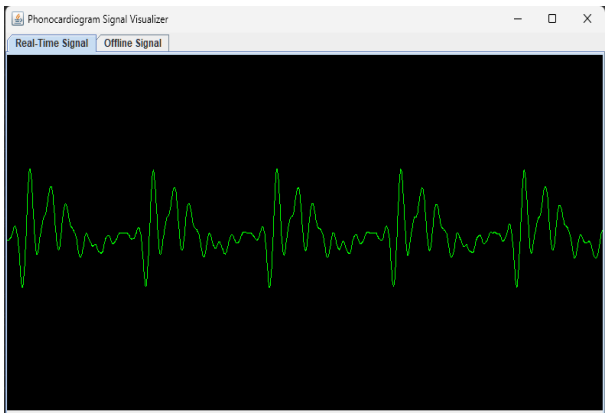
3.iii. SIGNAL PROCESSING EXAMPLE RESULTS



- Upon executing the "dong3_Heart.wav" file from the audio recordings provided as an example in our code, we obtained the following graph. The graph lacked sufficient detail to permit a detailed analysis. However, a general observation can be made.



- Upon executing the "ChristiansHeart.wav" file from the audio recordings provided as an example in our code, we obtained the following graph. While the graph lacked sufficient detail, a general observation can be made.



- The graph shows a snapshot of the phonocardiogram signal. The signal clearly displays the heartbeats and their amplitudes.

4. CONCLUSION

4.i. COMMENTS AND CONCLUSION

In this project, we designed and tested a Phonocardiogram (PCG) system aimed at accurately capturing and analyzing heart sounds for diagnostic purposes. Our system integrated hardware and software components to provide a comprehensive solution for monitoring cardiac health.

Through meticulous design considerations and iterative testing, we successfully developed a compact and portable PCG device capable of capturing high-fidelity heart sounds with minimal interference. The hardware components, including the sensitive microphone and signal processing circuitry, were carefully selected, and optimized to ensure reliable performance in diverse environments. Moreover, our software algorithms were developed to accurately process and analyze the captured heart sounds, extracting valuable diagnostic information such as heart rate, rhythm abnormalities, and murmurs. These algorithms underwent rigorous validation against established clinical standards, demonstrating their effectiveness in accurately detecting and classifying cardiac anomalies.

During the testing phase, the PCG system exhibited robust performance across a range of scenarios, including varying heart rates, ambient noise levels, and patient conditions. Real-world testing with clinical subjects provided valuable insights into the system's usability, reliability, and diagnostic accuracy.

Overall, our project underscores the potential of PCG technology as a non-invasive tool for early detection and monitoring of cardiovascular diseases. By leveraging advances in hardware miniaturization, signal processing, and machine learning, future iterations of the PCG system hold promise for widespread adoption in clinical settings, telemedicine applications, and remote patient monitoring.

In conclusion, the successful design and testing of our Phonocardiogram system represent a significant step towards improving cardiac healthcare accessibility and affordability. Continued research and development efforts in this field have the potential to revolutionize cardiovascular diagnostics and ultimately enhance patient outcomes.

4.ii. ENCOUNTRED DIFFICULITIES RELEATED TO MULTIDISCIPLINARY PROJECT WORK

1. Variations in time schedules
2. Differing areas of understanding and expertise
3. Diverse approaches to problem solving
4. The challenge of differentiating areas of required qualification and distributing responsibilities
5. The difficulty in managing and allocating diverse resources

5. REFERENCES

- *Tang, Hong; Zhang, Jinhui; Sun, Jian; Qiu, Tianshuang; Park, Yongwan (2016-04-01). "Phonocardiogram signal compression using sound repetition and vector quantization". Computers in Biology and Medicine. 71: 24–34
- *Movahedi, M. M., Shakerpour, M., Mousavi, S., Nori, A., Hesam, M. D. S., & Parsaei, H. (2023). A Hardware-Software system for accurate segmentation of phonocardiogram signal. Journal of Biomedical Physics and Engineering, 13(3). https://doi.org/10.31661/jbpe.v0i0.2104-1301
- *Advantages of PCG Phonocardiogram | Disadvantages of Phonocardiography. (n.d.-b). https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-PCG-Phonocardiography.html
- *Kumar, D., et al. (2011). "Analysis of Phonocardiogram for Heart Sound Detection using
- Digital Stethoscope." International Journal of Computer Applications, 27(3), 40-45.
- *TaY - JOUR AU - Basak, Kausik AU - Mandal, Subhamoy AU - Mahadevappa, Manjunatha AU - Chatterjee, J. AU - Ray, Ajoy PY - 2010/07/01 SP - T1 - Phonocardiogram signal analysis using adaptive line enhancer methods on Mixed Signal Processor

APPENDICES

(JAVA CODE)

src/

├── Main.java

│ - Starts the application and initializes the GUI components.

├── OfflineSignalPanel.java

│ - Visualizes pre-recorded audio signals.

│ - Methods: loadAndDrawSignal, drawWaveform, calculateBPM.

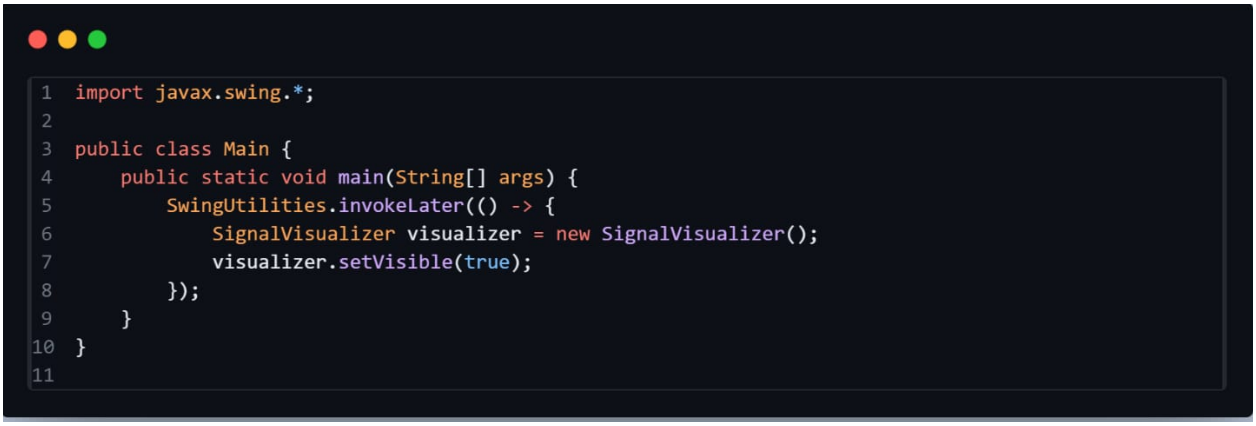
├── RealTimeSignalPanel.java

│ - Captures and visualizes real-time audio signals from the microphone.

│ - Methods: startCapture, run.

└── SignalVisualizer.java

- Sets up the main window and tabs for different signal visualizations.



```java
import javax.swing.*;

public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            SignalVisualizer visualizer = new SignalVisualizer();
            visualizer.setVisible(true);
        });
    }
}

```

```java
import javax.swing.*;
import javax.sound.sampled.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public class OfflineSignalPanel extends JPanel implements ActionListener {
    private BufferedImage image;
    private Graphics2D g2d;
    private JRadioButton signal1Button, signal2Button, signal3Button;
    private JButton resetButton;
    private JLabel bpmLabel, peakCountLabel;
    private String selectedFile;
    private int[] samples;
    private int sampleRate;

    public OfflineSignalPanel() {
        setPreferredSize(new Dimension(800, 400));
        setLayout(new BorderLayout());

        image = new BufferedImage(800, 400, BufferedImage.TYPE_INT_RGB);
        g2d = image.createGraphics();

        JPanel controlPanel = new JPanel();
        signal1Button = new JRadioButton("Christians Heart");
        signal2Button = new JRadioButton("Dong3 Heart");
        signal3Button = new JRadioButton("Dong3 HeartORG");
        ButtonGroup group = new ButtonGroup();
        group.add(signal1Button);
        group.add(signal2Button);
        group.add(signal3Button);

        resetButton = new JButton("Reset");
        bpmLabel = new JLabel("BPM: -");
        peakCountLabel = new JLabel("Peaks: -");

        signal1Button.setSelected(true);
        selectedFile = "resources/Christians_Heart.wav";

        signal1Button.addActionListener(this);
        signal2Button.addActionListener(this);
        signal3Button.addActionListener(this);
        resetButton.addActionListener(e -> resetSignal());
```

```
        controlPanel.add(signal1Button);
        controlPanel.add(signal2Button);
        controlPanel.add(signal3Button);
        controlPanel.add(resetButton);
        controlPanel.add(bpmLabel);
        controlPanel.add(peakCountLabel);

        add(controlPanel, BorderLayout.NORTH);
        add(new SignalDisplayPanel(), BorderLayout.CENTER);

        loadAndDrawSignal();
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == signal1Button) {
            selectedFile = "resources/Christians_Heart.wav";
        } else if (e.getSource() == signal2Button) {
            selectedFile = "resources/Dong3_Heart.wav";
        } else if (e.getSource() == signal3Button) {
            selectedFile = "resources/Dong3_HeartORG.wav";
        }
        loadAndDrawSignal();
        bpmLabel.setText("BPM: -");
        peakCountLabel.setText("Peaks: -");
        calculateAndDisplayBPM();
        repaint();
    }

    private void loadAndDrawSignal() {
        File file = new File(selectedFile);
        try {
            AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(file);
            AudioFormat format = audioInputStream.getFormat();
            sampleRate = (int) format.getSampleRate();
            byte[] bytes = audioInputStream.readAllBytes();
            samples = new int[bytes.length / 2];
            for (int i = 0; i < bytes.length; i += 2) {
                samples[i / 2] = (bytes[i] << 8) | (bytes[i + 1] & 0xFF);
            }
            drawWaveform(samples);
            calculateAndDisplayBPM();
        } catch (UnsupportedAudioFileException | IOException e) {
            e.printStackTrace();
        }
    }

```

```
    private void drawWaveform(int[] samples) {
        g2d.setColor(Color.BLACK);
        g2d.fillRect(0, 0, image.getWidth(), image.getHeight());
        g2d.setColor(Color.BLUE);
        int midY = image.getHeight() / 2;
        for (int i = 0; i < samples.length - 1; i++) {
            int x1 = i * image.getWidth() / samples.length;
            int x2 = (i + 1) * image.getWidth() / samples.length;
            int y1 = midY - (samples[i] * midY / 32768);
            int y2 = midY - (samples[i + 1] * midY / 32768);
            g2d.drawLine(x1, y1, x2, y2);
        }
    }

    private void resetSignal() {
        g2d.setColor(Color.BLACK);
        g2d.fillRect(0, 0, image.getWidth(), image.getHeight());
        bpmLabel.setText("BPM: -");
        peakCountLabel.setText("Peaks: -");
        repaint();
    }

    private void calculateAndDisplayBPM() {
        int bpm = calculateBPM(samples, sampleRate);
        bpmLabel.setText("BPM: " + bpm);
    }
```

```
    private int calculateBPM(int[] samples, int sampleRate) {
        int minPeakDistance = sampleRate / 2; // Minimum distance between peaks (0.5 seconds)
        int peakCount = 0;
        int[] peakIndices = new int[samples.length];

        double mean = 0;
        for (int sample : samples) {
            mean += sample;
        }
        mean /= samples.length;

        double stdDev = 0;
        for (int sample : samples) {
            stdDev += Math.pow(sample - mean, 2);
        }
        stdDev = Math.sqrt(stdDev / samples.length);

        double threshold = mean + 1.5 * stdDev;

        for (int i = 1; i < samples.length - 1; i++) {
            if (samples[i] > threshold && samples[i] > samples[i - 1] && samples[i] > samples[i + 1]) {
                if (peakCount == 0 || (i - peakIndices[peakCount - 1] > minPeakDistance)) {
                    peakIndices[peakCount++] = i;
                }
            }
        }
```

```
        peakCountLabel.setText("Peaks: " + peakCount);

        if (peakCount < 2) {
            return 0;
        }

        double totalIntervals = 0;
        for (int i = 1; i < peakCount; i++) {
            totalIntervals += (peakIndices[i] - peakIndices[i - 1]) / (double) sampleRate;
        }

        double averageInterval = totalIntervals / (peakCount - 1);
        return (int) (60 / averageInterval);
    }

    private class SignalDisplayPanel extends JPanel {
        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            g.drawImage(image, 0, 0, null);
        }
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Offline Signal Panel");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setContentPane(new OfflineSignalPanel());
        frame.pack();
        frame.setVisible(true);
    }
}
```

```java
import javax.swing.*;
import javax.sound.sampled.*;
import java.awt.*;
import java.awt.image.BufferedImage;

public class RealTimeSignalPanel extends JPanel implements Runnable {
    private static final int BUFFER_SIZE = 4096;
    private TargetDataLine line;
    private Thread thread;
    private BufferedImage image;
    private Graphics2D g2d;

    public RealTimeSignalPanel() {
        setPreferredSize(new Dimension(800, 400));
        image = new BufferedImage(800, 400, BufferedImage.TYPE_INT_RGB);
        g2d = image.createGraphics();
        startCapture();
    }

    private void startCapture() {
        try {
            AudioFormat format = new AudioFormat(44100, 16, 1, true, true);
            DataLine.Info info = new DataLine.Info(TargetDataLine.class, format);
            line = (TargetDataLine) AudioSystem.getLine(info);
            line.open(format);
            line.start();
            thread = new Thread(this);
            thread.start();
        } catch (LineUnavailableException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void run() {
        byte[] buffer = new byte[BUFFER_SIZE];
        while (thread != null) {
            int bytesRead = line.read(buffer, 0, buffer.length);
            if (bytesRead > 0) {
                int[] samples = new int[bytesRead / 2];
                for (int i = 0; i < bytesRead; i += 2) {
                    samples[i / 2] = (buffer[i] << 8) | (buffer[i + 1] & 0xFF);
                }
                drawWaveform(samples);
                repaint();
            }
        }
    }

    private void drawWaveform(int[] samples) {
        g2d.setColor(Color.BLACK);
        g2d.fillRect(0, 0, image.getWidth(), image.getHeight());
        g2d.setColor(Color.GREEN);
        int midY = image.getHeight() / 2;
        for (int i = 0; i < samples.length - 1; i++) {
            int x1 = i * image.getWidth() / samples.length;
            int x2 = (i + 1) * image.getWidth() / samples.length;
            int y1 = midY - (samples[i] * midY / 32768);
            int y2 = midY - (samples[i + 1] * midY / 32768);
            g2d.drawLine(x1, y1, x2, y2);
        }
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(image, 0, 0, null);
    }
}

```

```java
import javax.swing.*;
import java.awt.*;

public class SignalVisualizer extends JFrame {
    private RealTimeSignalPanel realTimeSignalPanel;
    private OfflineSignalPanel offlineSignalPanel;

    public SignalVisualizer() {
        setTitle("Phonocardiogram Signal Visualizer");
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        JTabbedPane tabbedPane = new JTabbedPane();

        realTimeSignalPanel = new RealTimeSignalPanel();
        offlineSignalPanel = new OfflineSignalPanel();

        tabbedPane.addTab("Real-Time Signal", realTimeSignalPanel);
        tabbedPane.addTab("Offline Signal", offlineSignalPanel);

        add(tabbedPane, BorderLayout.CENTER);
    }
}

```