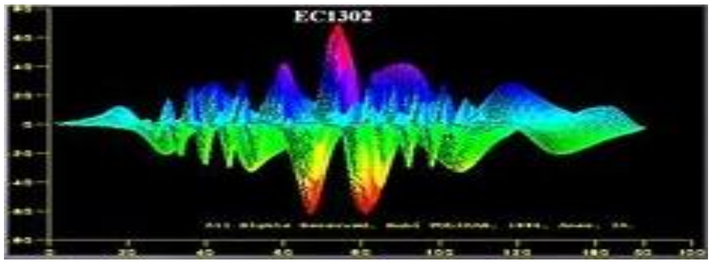
معهد الأهرامات العالى للهندسة والتكنولوجيا
P.H.I For Engineering and Technology



Digital Signal Processing

Project Name:

ECG Graph Monitoring

Team Members:

Abdelrahman Sherif
Sondos Reda
Ali Osama

Under Supervision:

Dr. Gamal Elsheikh
Dr. Heba Emara
Eng/Ahmed Fathy

Objective:

The objective of ECG data analysis using Arduino is to process and interpret the electrical signals recorded from the heart to extract meaningful insights about cardiac health. By leveraging Arduino's capabilities for data acquisition and processing, this project aims to analyze ECG signals in real-time or offline, detecting abnormalities such as arrhythmias, heart rate variability, and other cardiac anomalies. Through algorithmic analysis and visualization techniques, the goal is to provide users with actionable information about their heart's condition, aiding in early detection of potential health issues and facilitating informed decision-making regarding medical care and lifestyle choices.

Introduction:

In the realm of modern healthcare, the utilization of technology has revolutionized the way we monitor and assess vital physiological parameters. One such parameter, the Electrocardiogram (ECG), offers invaluable insights into cardiac health by recording the electrical activity of the heart. With the advent of affordable microcontroller platforms like Arduino, there emerges a promising avenue for enthusiasts and professionals alike to delve into the realm of ECG data analysis.

The ECG data analysis project using Arduino presents an innovative approach to real-time cardiac monitoring and health assessment. By integrating Arduino's versatile hardware capabilities with sophisticated signal processing algorithms, this project aims to provide users with a cost-effective and accessible solution for interpreting ECG signals.

Through this project, individuals can embark on a journey to understand the intricate patterns of the heart's electrical activity, detect anomalies, and potentially identify early indicators of cardiovascular disorders. Furthermore, the DIY nature of Arduino empowers enthusiasts to customize and expand upon the project, fostering a collaborative community focused on advancing cardiac health monitoring technology.

Hardware Components:

1. **Arduino**:

The Arduino Uno is a popular microcontroller board based on the ATmega328P chip. It's the most widely used Arduino board due to its simplicity and versatility.

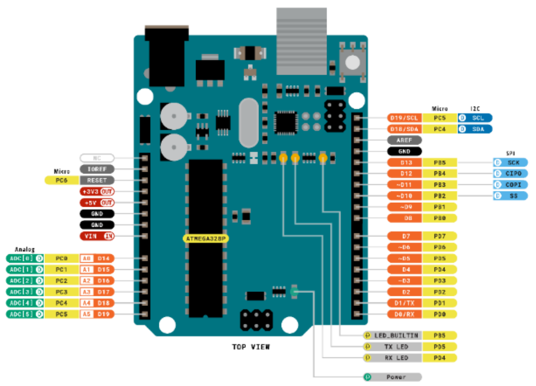


Figure 1 Arduino Uno

Here's a concise overview:

- **Microcontroller**: The heart of the Arduino Uno is the ATmega328P microcontroller, which runs at 16 MHz and has 32 KB of flash memory for storing code and 2 KB of RAM for data storage.

- **Digital and Analog I/O:** The Uno features 14 digital input/output pins, of which 6 can be used as PWM (Pulse Width Modulation) outputs, and 6 analog input pins. These pins allow interfacing with various sensors, actuators, and other electronic components.

- **USB Interface**: The Uno can be easily connected to a computer via USB, allowing for programming and serial communication. It uses a standard USB Type-B connector.

- **Power Options**: The board can be powered via USB connection or an external power supply (7-12V DC). It also has a built-in voltage regulator, allowing it to be powered directly from a 5V source.

- **Integrated Development Environment (IDE)**: Arduino Uno is programmed using the Arduino IDE, a user-friendly software environment that simplifies coding and uploading sketches (programs) to the board.

- **Open-Source Platform**: Arduino Uno is part of the Arduino ecosystem, which is open-source. This means the hardware designs, software, and documentation are freely available, encouraging collaboration and innovation among the community.

Overall, the Arduino Uno is a versatile and beginner-friendly platform for electronics projects, ranging from simple blinking LED experiments to complex automation and robotics projects. Its ease of use, extensive community support, and vast array of compatible sensors and shields make it a favorite among hobbyists, educators, and professionals alike.

2. ECG Sensor:

An ECG sensor in electronics is a device used to measure the electrical activity of the heart. It typically consists of electrodes that are placed on the body to detect the electrical impulses generated by the heart's muscle during each heartbeat. These impulses are then amplified, filtered, and processed to produce an electrocardiogram (ECG) signal, which represents the heart's activity over time. ECG sensors are commonly used in medical devices, such as ECG machines and wearable monitors, to diagnose various heart conditions and monitor heart health. The pinout of an ECG sensor can vary depending on the specific model and manufacturer, but typically, it includes the following pins:

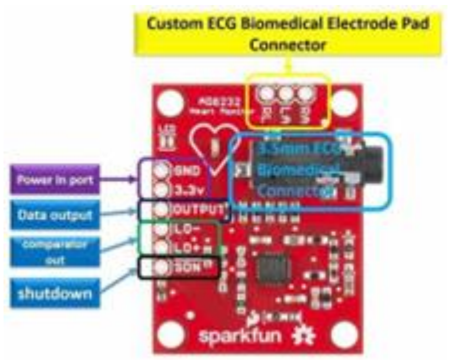


Figure 1 ECG Sensor

1. Power Supply (VCC):

This pin provides the operating voltage required by the ECG sensor. It is typically connected to a power source such as a 3.3V or 5V pin on the microcontroller or power supply.

2. Ground (GND):

This pin is connected to the ground of the circuit, completing the electrical circuit and providing a reference voltage for the sensor.

3. Data Output:

This pin outputs the electrical signals detected by the ECG sensor. These signals represent the electrical activity of the heart and are typically analog or digital signals that can be processed by a microcontroller or other electronic circuitry.

4. Reference Electrode:

Some ECG sensors may include a separate pin for connecting a reference electrode, which serves as a stable reference point for the measurement of heart electrical activity. This electrode is typically placed at a location on the body away from the electrodes detecting the heart signals.

5. Other Pins:

Depending on the specific features and functionality of the ECG sensor, there may be additional pins for features such as calibration, temperature compensation, or communication with other devices. It's essential to refer to the datasheet or documentation provided by the manufacturer for the specific pinout and connection details of the ECG sensor you are using. Additionally, proper electrode placement and signal conditioning are crucial for accurate ECG measurements.

Circuit Diagram:

The AD8232 Heart Rate Monitor breaks out nine connections from the IC.

We’ll connect five of the nine pins on the board to Arduino. The five pins you need are labeled GND, 3.3v, OUTPUT, LO-, and LO+.

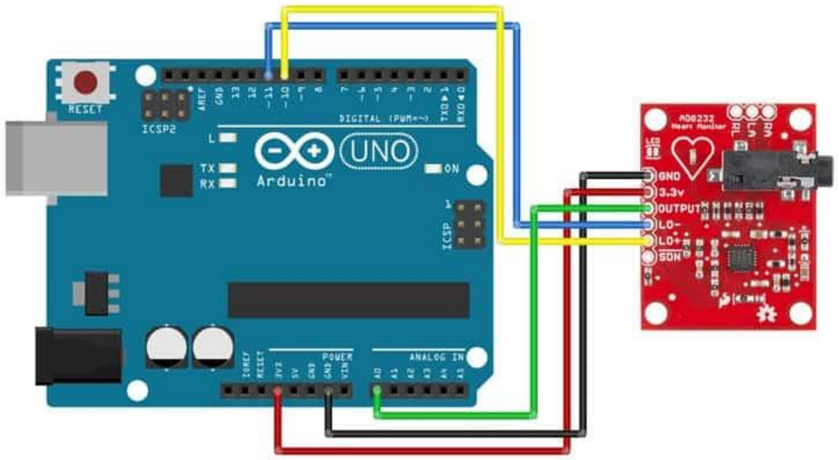
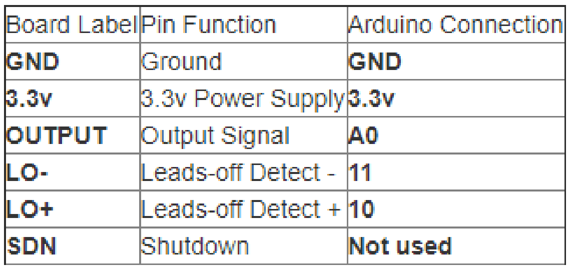


Figure 2 Circuit Diagram



Board Label	Pin Function	Arduino Connection
GND	Ground	**GND**
3.3v	3.3v Power Supply	**3.3v**
OUTPUT	Output Signal	**A0**
LO-	Leads-off Detect -	**11**
LO+	Leads-off Detect +	**10**
SDN	Shutdown	**Not used**

Figure 3 Connection of the ECG Sensor with Arduino Uno

ECG Sensor Placement on Body:

It is recommended to snap the sensor pads on the leads before application to the body. The closer to the heart the pads are, the better the measurement. The cables are color-coded to help identify proper placement.

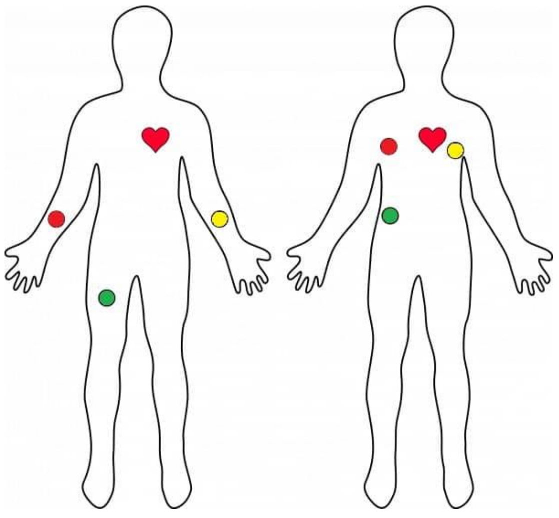


Figure 2 ECG Sensor Placement on Body

Software Explain:

In this project, Arduino serves as the backbone for capturing and transmitting ECG data from sensors to a computer. Its role involves:

1. Data Acquisition:

Arduino interfaces with ECG sensors to capture raw electrical signals representing cardiac activity. It converts analog signals to digital data, ensuring accurate representation of ECG waveforms.

2. Data Transmission:

Arduino facilitates the transmission of captured data to a computer for further analysis. Through serial communication, it sends the data in real-time to software like CoolTerm, which saves it in a text file format.

3. Arduino Data Capture:

Arduino is utilized to capture data from the ECG sensor. The Arduino board interfaces with the sensor to retrieve electrical signals representing cardiac activity. These signals are then transmitted to a computer and saved in a text file format using software like CoolTerm. This step ensures the raw ECG data is recorded for further analysis.

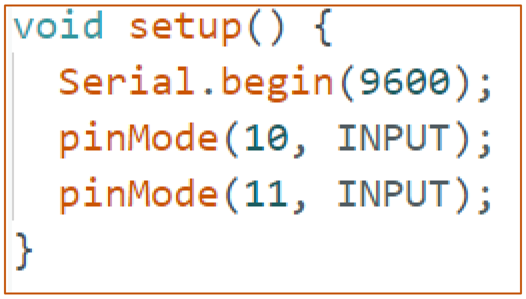
4. MATLAB Data Analysis:

Following data capture, MATLAB is employed for comprehensive data analysis and visualization. The captured ECG data is imported into MATLAB for processing. In the time domain, the data is plotted to visualize the waveform, enabling examination of the heart's electrical activity over time. Additionally, advanced analysis techniques are applied to explore the frequency domain characteristics of the ECG signals. This may involve techniques such as Fourier Transform to examine frequency components and spectral analysis to identify specific frequency bands of interest.

Arduino Code Explain :

6. Setup Function:

First, we initialize the serial communication at baud rate equals 9600. Then we define the input pins For receiving signals from the ECG Sensor.
It only runs one time at the beginning of the program.

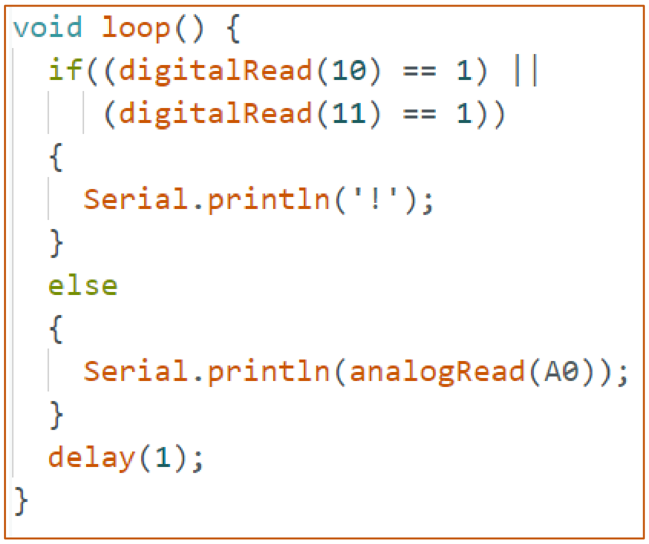


```
void setup() {
  Serial.begin(9600);
  pinMode(10, INPUT);
  pinMode(11, INPUT);
}
```

Figure 3 Setup Function

7. Loop Function:

The loop function is the main function in Arduino sketches. Whatever code is inside loop will execute repeatedly as long as the Arduino is powered on.
Inside the loop, there's an if statement that checks if either digital pin 10 or digital pin 11 is HIGH (has a value of 1). If either of them is HIGH, it will print a '!' character to the serial monitor. If neither pin is HIGH, it will read the analog input from pin A0 using analogRead(A0) and print that value to the serial monitor instead.



```
void loop() {
  if((digitalRead(10) == 1) ||
     (digitalRead(11) == 1))
  {
    Serial.println('!');
  }
  else
  {
    Serial.println(analogRead(A0));
  }
  delay(1);
}
```

Figure 4 Loop Function

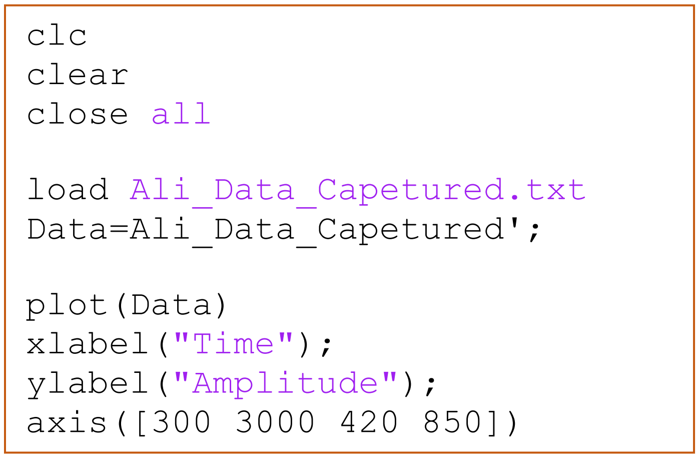
MATLAB Code Explain :

1. Plot the captured data from the ECG sensor

Clearing Environment: Clears the command window, clears all variables from the workspace, and closes all figure windows.

Loading Data: Loads data from the file "Ali_Data_Captured.txt" into the variable Data. The data is transposed so that each column represents a variable.

Plotting: Plots the data stored in Data using the plot function.



```
clc
clear
close all

load Ali_Data_Capetured.txt
Data=Ali_Data_Capetured';

plot(Data)
xlabel("Time");
ylabel("Amplitude");
axis([300 3000 420 850])
```

Figure 5 Loading and plotting the captured data from ECG Sensor

Labels and Axis Limits: Sets the x-axis label to "Time", the y-axis label to "Amplitude", and restricts the plot's axis limits to the range [300, 3000] for the x-axis and [420, 850] for the y-axis.

Furthermore, to enhance the analysis and compression of the ECG data, sophisticated algorithms like Huffman Coding are implemented. Huffman Coding is utilized for data compression, reducing redundancy and minimizing storage requirements without compromising essential information. By applying Huffman Coding to the ECG data, we can achieve efficient storage and transmission of cardiac information, optimizing resources and enabling streamlined data management.

2. Encoding or compressing data using Huffman Coding Algorithm:

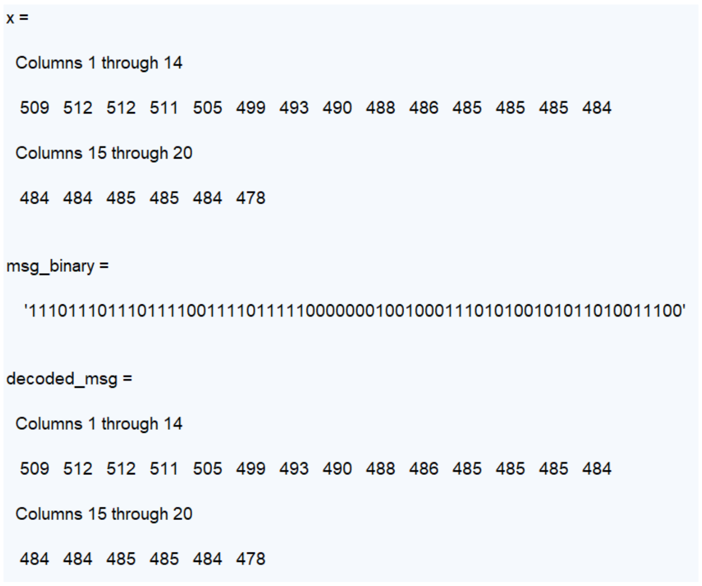
- **Clearing Environment :** Clears the command window, clears all variables from the workspace, and closes all figure windows.
- **Loading Data :** Loads data from the file "Ali_Data_Capetured.txt" into the variable x. The data is transposed so that each column represents a variable.
- **Data Truncation :** It selects the first 20 values of the loaded data and stores them in x. This step simplifies the process for demonstration purposes.

```
clc
clear
close all

load Ali_Data_Capetured.txt
x=Ali_Data_Capetured';

x = x(1:20); %% just taking the first 20 values of the data
for simplicity

[unique_chars,num_unique_chars] = findChars(x);

[chars_prop,chars_count] =
findMsgProps(x,length(x),unique_chars,num_unique_chars);

codebook=huffman_encoding(chars_prop);

msg_binary = msgInBinary(x,length(x),codebook,unique_chars)

decoded_msg = Recevier(msg_binary,codebook,unique_chars)
```

Figure 6 Source Coding For ECG Sensor

- **Finding Unique Characters** : The function findChars() identifys unique characters in the data.
- **Finding Message Properties** : The function findMsgProps() calculates the properties of the message, such as character frequency.
- **Huffman Encoding** : The huffman_encoding() function generates a Huffman codebook based on the properties of the message.
- **Converting Message to Binary** : The msgInBinary() function converts the message x into binary format using the Huffman codebook.
- **Decoding the Message** : The Recevier() function decodes the binary message using the Huffman codebook, reconstructing the original message.

In the next figure we can see the output of the previous code :



```
x =

  Columns 1 through 14

   509   512   512   511   505   499   493   490   488   486   485   485   485   484

  Columns 15 through 20

   484   484   485   485   484   478


msg_binary =

    '111011101110111100111101111100000001001000111010100101011010011100'


decoded_msg =

  Columns 1 through 14

   509   512   512   511   505   499   493   490   488   486   485   485   485   484

  Columns 15 through 20

   484   484   485   485   484   478
```

Figure 7 Encoding and Decoding of ECG signal

3. Plotting the data in time and frequency domain:

- **Clearing Environment**: Clears the command window, clears all variables from the workspace, and closes all figure windows.

- **Loading Data**: Loads data from the file "Ali_Data_Capetured.txt" into the variable data. The data is transposed so that each column represents a variable.

```
clc,clear,close all
load Ali_Data_Capetured.txt
data=Ali_Data_Capetured';
```

Figure 8 Loading Captured Data To MATLAB

- **FFT of ECG Signal with Filtering** : in this part we move through some steps to get the Fast Fourier Transform of the ECG Signal .

```
Fs = 360;                   % Sampling frequency
T = 1/Fs;                   % Sampling period
t = (0:1/Fs:10-1/Fs);       % Time vector
data = normalize(data);
n = length(data);           % number of samples

%% frequency
f = (0:n-1)*(Fs/n); % frequency range
L = 1:floor(n/2);   % plot the first half of freqs
```

Figure 9 Define Time Vector and Sampling Frequency

1. Defines the sampling frequency Fs, sampling period T, and time vector t. Normalizes the data.
2. Calculates the number of samples n.
3. Defines the frequency range f.
4. Divides the frequency range f into half L.
5. Computes the FFT of the raw signal and plots it.
6. Applies a low-pass filter (LPF) using a Butterworth filter, then computes the FFT of the LPF signal and plots it.
7. Applies a high-pass filter (HPF) using another Butterworth filter on the LPF signal, then computes the FFT of the HPF signal and plots it.

```
%% FFT of raw signal
figure(1)
dataF = fft(data);
subplot(311)
plot(f(L),abs(dataF(L))),grid on;
title('FFT of raw ECG signal')

%% LPF filter
fc = 50;
Wn=2*fc/Fs; % normalized frequency
order=6; % the order
[b,a] = butter(order,Wn,'low');
nv=filter(b,a,data);
vv=fft(nv);
power1 = abs(vv).^2/n;
subplot(312)
plot(f(L),abs(power1(L))),grid on;
title('FFT of LPF')

%% HPF filter
fc1 = 10;
Wn1=2*fc1/Fs; % normalizedfrequency
order=6; % the order
[b1,a1] = butter(order,Wn1,'high');
nv1=filter(b1,a1,nv);
vv1=fft(nv1);
power2 = abs(vv1).^2/n;
subplot(313)
plot(f(L),abs(power2(L))),grid on;
title('FFT of HPF')
```

Figure 10 Plotting FFT For Raw Data and Filtered Versions

- **Plotting:**

Plots the raw ECG signal and its filtered versions (LPF and HPF) in separate subplots.

This code analyzes an ECG signal by applying a low-pass and high-pass Butterworth filter and visualizes the frequency content before and after filtering using the FFT. Finally, it plots the original signal along with its LPF and HPF versions.

```
figure(2);
subplot(311)
plot(data);
title('Signal')
xlabel('t (milliseconds)')
ylabel('data(t)')
subplot(312)
plot(nv);
title('LPF of ECG signal')
subplot(313)
plot(nv1);
title('HPF of ECG signal')
```

Figure 11 Plotting The Raw ECG Signal and its Filtered Versions

Software Output:

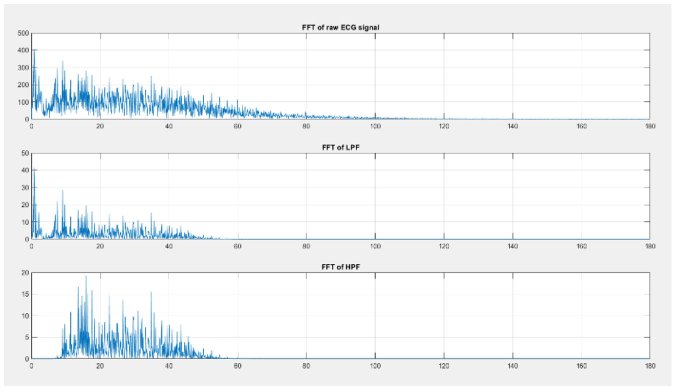


Figure 13 FFT for Raw data with LPF and HPF are Applied on It

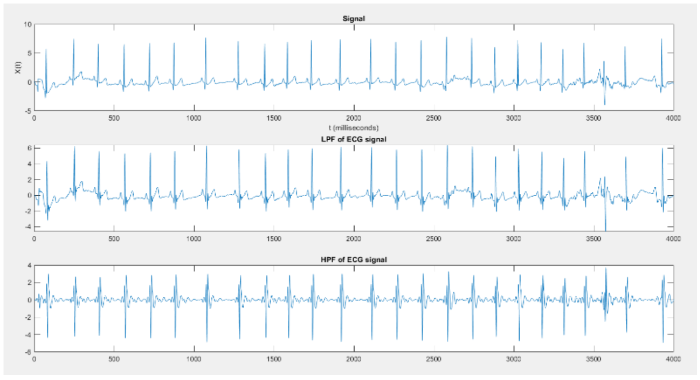


Figure 12 Original Signal with LPF and HPF are Applied on It

Overall, the integration of Arduino for data capture and MATLAB for analysis and visualization forms a robust framework for ECG data analysis. This approach facilitates not only real-time monitoring but also in-depth examination and interpretation of cardiac signals, paving the way for enhanced diagnostic capabilities and personalized healthcare interventions.

Conclusion:

In summary, the ECG data analysis project combines Arduino and MATLAB to capture, analyze, and interpret electrocardiogram (ECG) signals. Arduino captures raw data from ECG sensors, which is then saved and transmitted to MATLAB for comprehensive analysis. MATLAB enables visualization of ECG waveforms in both time and frequency domains, facilitating detailed examination of cardiac activity. Advanced algorithms like Huffman Coding optimize data compression for efficient storage and transmission. This project offers a cost-effective, accessible solution for cardiac health monitoring, with potential applications in remote patient care and personalized health management. Overall, it represents a promising convergence of electronics, data science, and healthcare, with the aim of improving cardiovascular well-being and empowering individuals to take proactive steps towards better health.