Advance Health Monitoring System

ECG and Heart Rate Analysis for Early Detection and Diagnosis

Ashutosh Rajendra karve

Bits-ID: **2024ht01021**

2024ht01021@wilp.bits-pilani.ac.in

+91 9765541324

Pune, Maharashtra

14/09/2024

<< GitHub Project Link >>

clone: https://github.com/Ashutoshkarve007/HealthMonitoring_BITS_Pilani_2024ht01021.git

GUIDED FOR DETAILED UNDERSTANDING OF ECG AND WORKING

Dr. Prachi Pawar (BHMS)
Reg No:(85206)
Pune, Maharashtra

Abstract

This project develops an advanced health monitoring system that uses the STM32 Blue Pill microcontroller and AD8232 ECG sensor to capture comprehensive cardiovascular data. The system is designed not only to monitor heart rate but also to detect critical conditions such as heart attacks, myocardial infarction, and electrolyte imbalances through sophisticated signal processing and diagnostic algorithms implemented in Python. Data captured in real-time is analyzed to identify abnormal rhythms and potential cardiovascular issues, providing detailed health reports within minutes.

1 Introduction

The **Advance Health Monitoring System** is a project aimed at developing a real-time ECG monitoring solution using the STM32 Blue Pill microcontroller and AD8232 ECG sensor. The primary objective of this project is to capture ECG data from a user, transmit it via USB to a Python-based system for analysis, and visualize the ECG signal. The current version of the project is focused on raw ECG data acquisition and visualization, with plans for further development, such as AI-based analysis, in the future.

2 System Design

The system is designed to acquire ECG data from a human subject using a combination of hardware and software components:

Hardware:

- **STM32 Blue Pill Microcontroller:** Used for data acquisition via its analog-to-digital converter (ADC) and communication over USB (virtual COM port).
- **AD8232 ECG Sensor:** Measures the electrical activity of the heart and outputs analog signals, which are fed into the STM32 for processing.

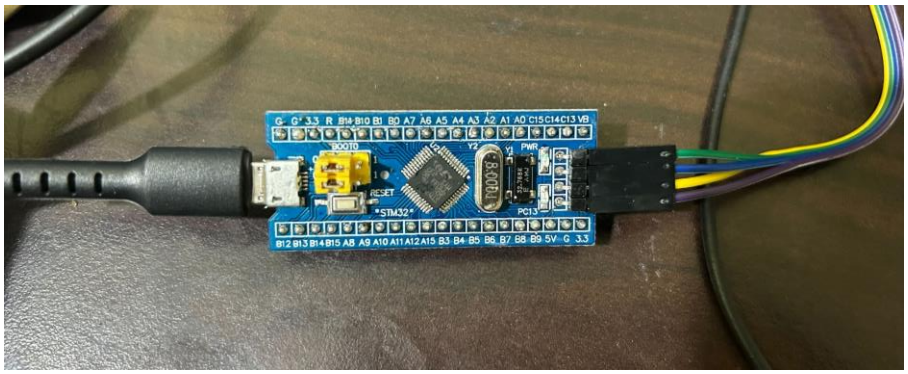
Software:

- **STM32 Firmware:** The firmware reads raw ADC data from the ECG sensor connected to the STM32's PA1 (A1) pin. The raw data (0-4095 for 12-bit ADC) is transmitted to a PC via USB.
- **Python Interface:** The Python code reads the raw ECG data from the STM32 and plots the real-time ECG signal. The data is visualized to observe the heart's electrical activity over time.

3 Hardware Integration

Step 1: Assembling the Hardware

- **Components Required**:
    - STM32 Blue Pill microcontroller



STM32(Blue Pill) Development Board

Microcontroller:

Part	:	STM32F103C8T6
Manufacturer	:	ST-Microelectronics
Core	:	Arm Cortex-M3
Max. Clock	:	Speed 72MHz
Package	:	LQFP 48 pins

Internal memories:

FLASH	:	64KiB
SRAM	:	20KiB

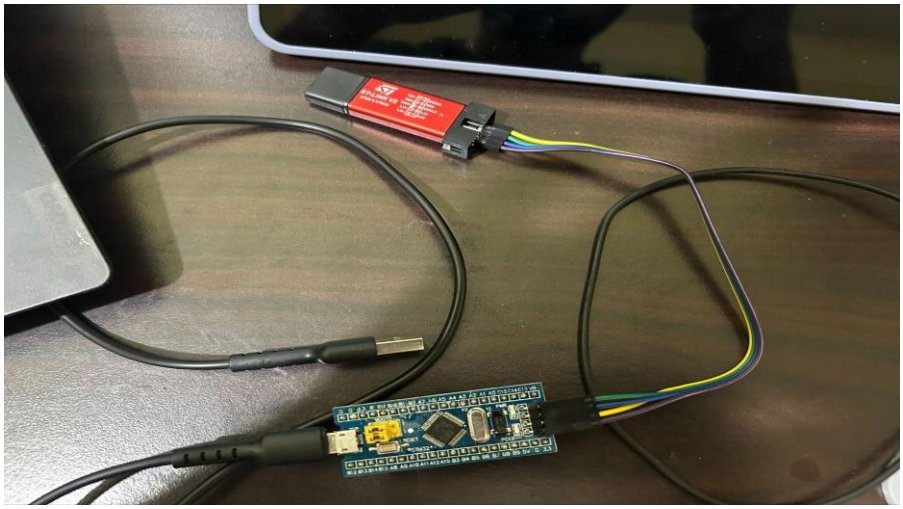
Oscillators:

HSI	:	8MHz
LSI	:	40kHz
HSE	:	8MHz
LSE	:	32.768kHz

Power:

Sources	:	Any +3.3V pin (+3.3V) Any +5V pin (+5V) USB connector (+5V)

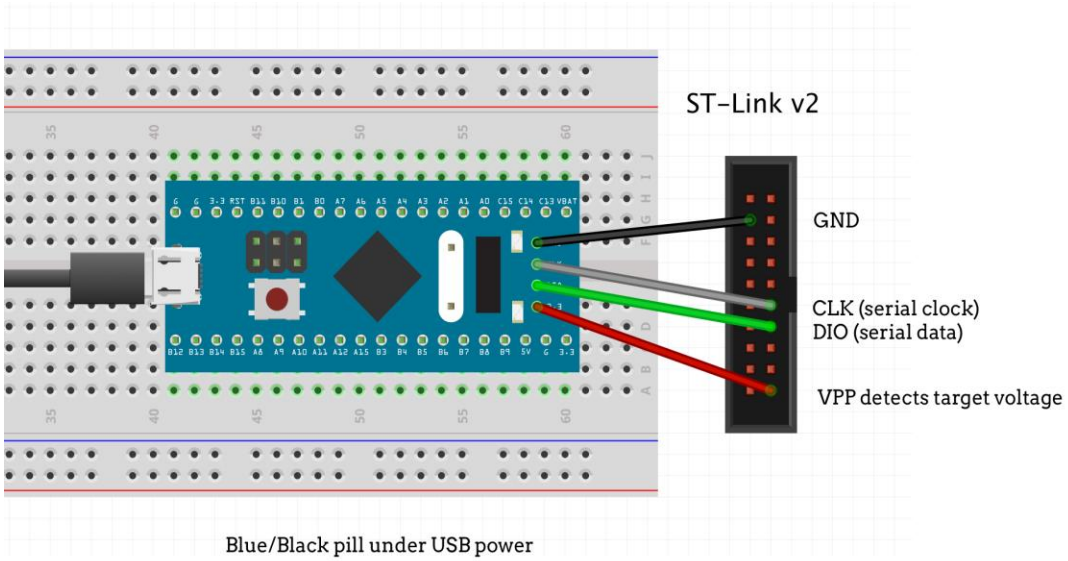
- USB cables & ST Link V2 for Programming



Possible Connections

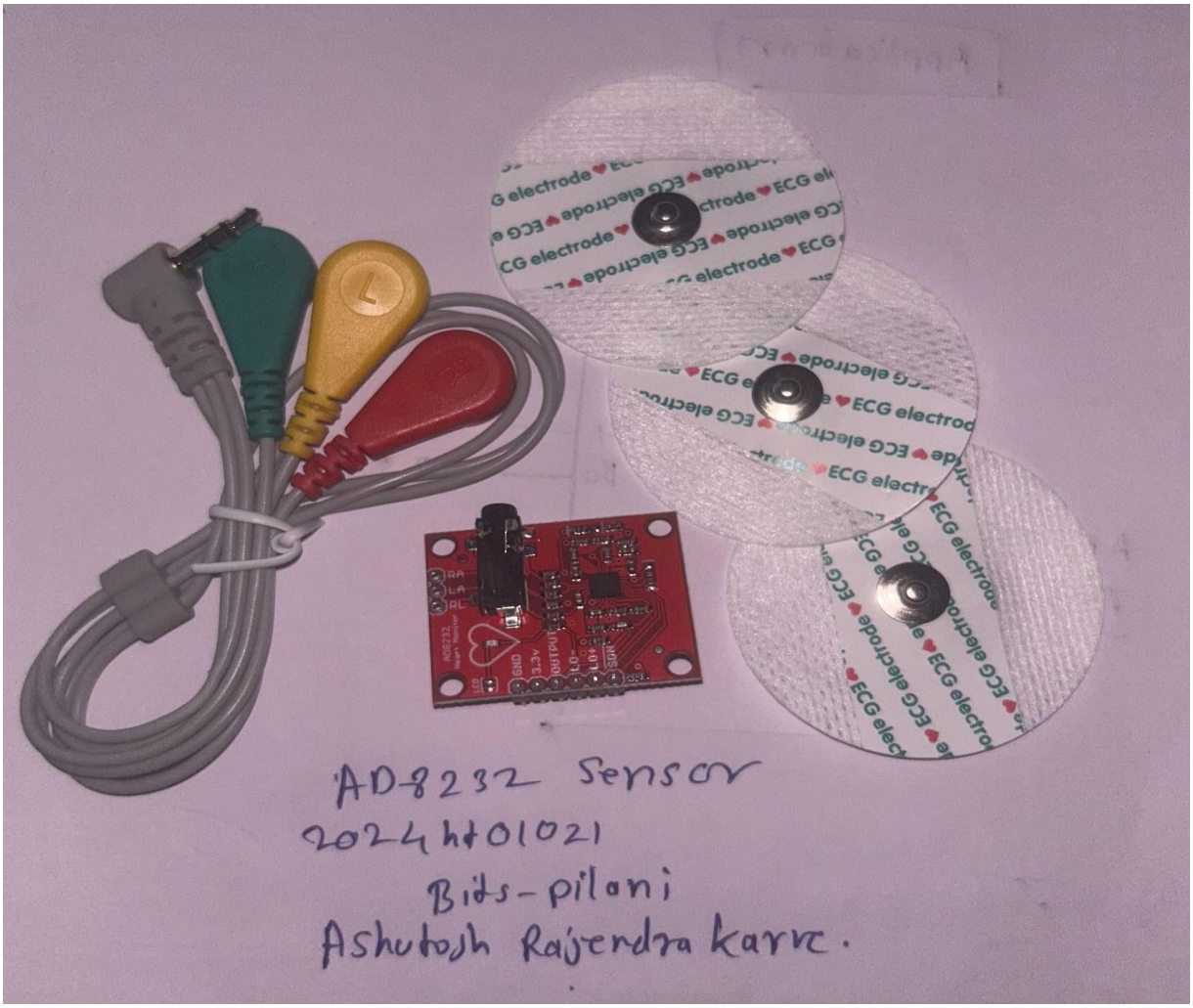
To use a ST-Link to program a blue/black pill (BP) under external power, connect it like this:

ST-Link GND (even-numbered pins from 4 to 20) to BP Ground
ST-Link SWDIO (pin 7) to BP DIO
ST-Link SWCLK (pin 9) to BP CLK
ST-Link VAPP (pins 1 or 2) to BP 3V3



Blue/Black pill under USB power

- AD8232 ECG Sensor



The **AD8232 ECG sensor** is designed for measuring heart activity by capturing, amplifying, and filtering ECG signals. It removes noise and interference, providing a clean analog signal for further processing.

Key Features:

- **Low Power**: Ideal for portable health monitoring.
- **Noise Filtering**: Removes motion artifacts and electrical interference for cleaner signals.
- **Electrode Inputs**: Requires three electrodes (right arm, left arm, right leg) for accurate readings.
- **Analog Output**: Provides analog ECG signals to the STM32's ADC for digitization.

In this project, the sensor's output is connected to **PA1 (A1)** on the STM32, which digitizes and transmits the ECG data over USB for real-time analysis and visualization.

- Heartbeat Pulse Sensor (For Future scope)

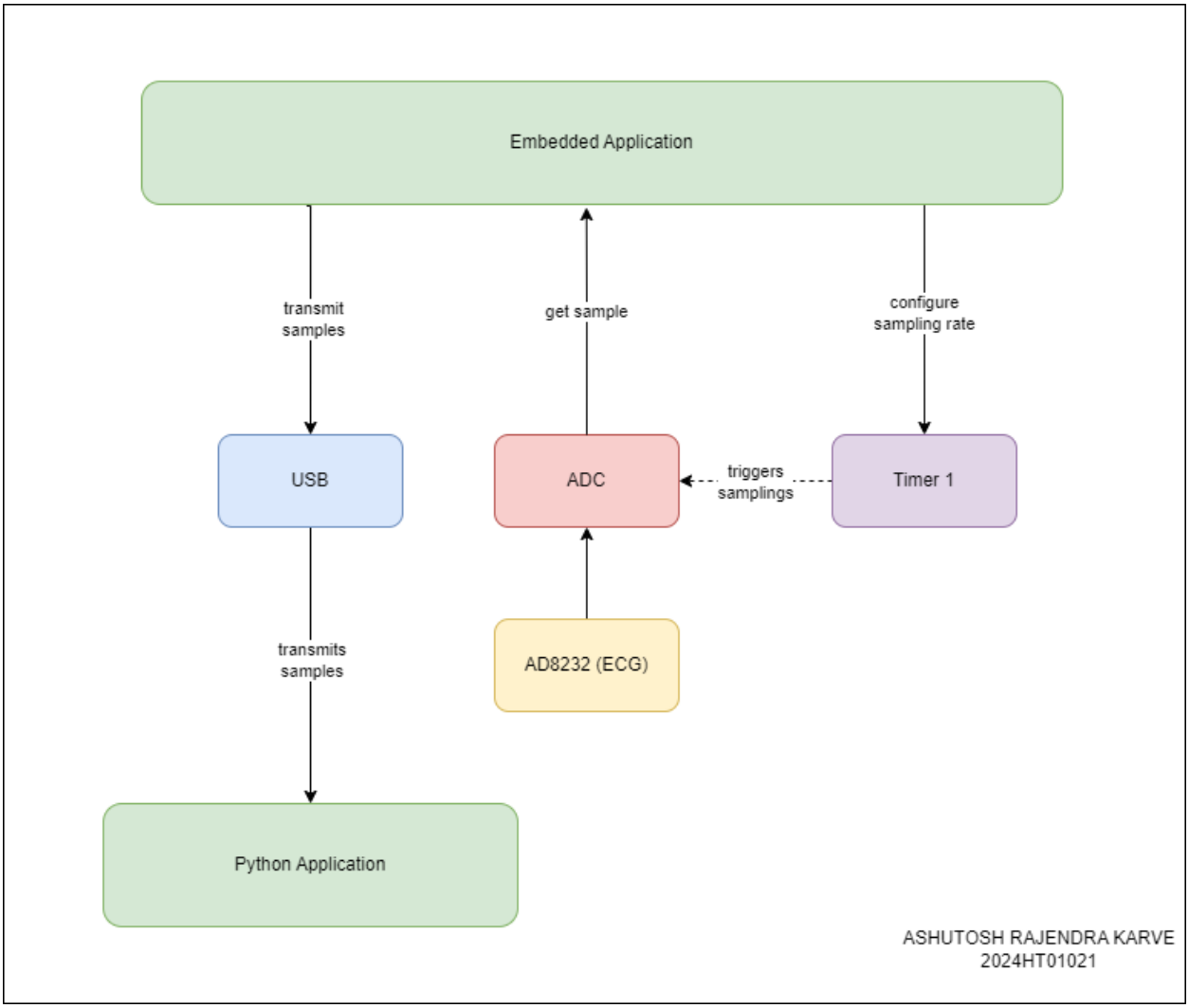
The **Heartbeat Pulse Sensor** is designed to measure heart rate by detecting the blood flow through the skin using photoplethysmography (PPG). It provides an easy way to monitor pulse activity by converting the changes in light absorption, caused by blood flow, into an electrical signal.

- Connecting wires

- **Procedure:**

1. **Power Connections**:
    - Connect the 3.3V and GND pins of the STM32 to the respective pins on the AD8232 and Heart Pulse Sensor.
2. **Signal Connections**:
    - Connect the OUTPUT of the AD8232 to PA1 on STM32 for ECG data capture.
    - Connect the OUTPUT of the Heart Pulse Sensor to PA0 on STM32 for heart rate measurement. (Future Scope)
3. **USB Configuration**:
    - Ensure PA11 (USB_DM) and PA12 (USB_DP) are connected for USB functionality.
4. **Configuring Leads-Off Detection:**
    - Use PB0 and PB1 on STM32 to connect to LO- and LO+ on the AD8232 for detecting if the leads are properly attached.

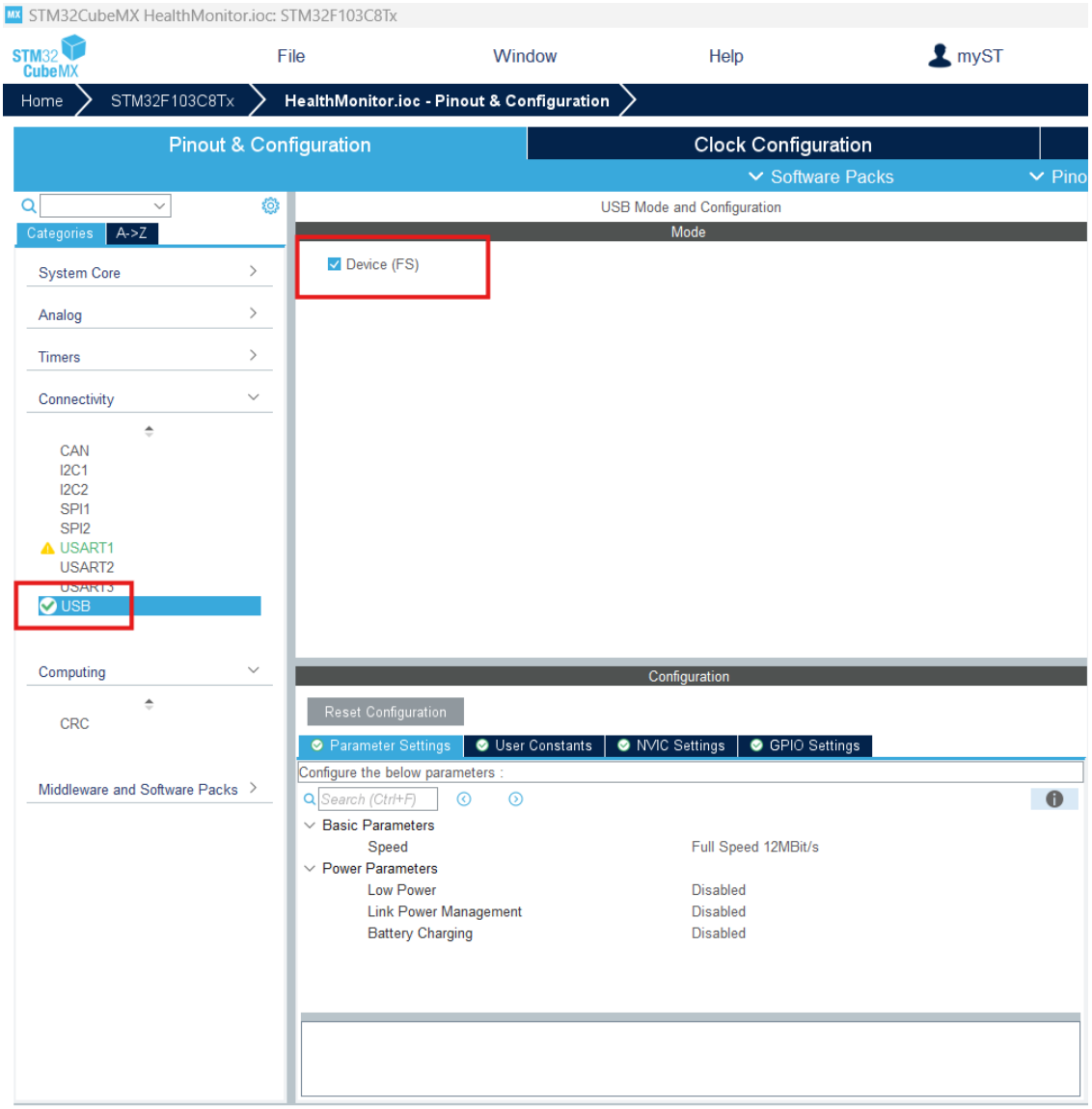
- **Hardware Architecture**



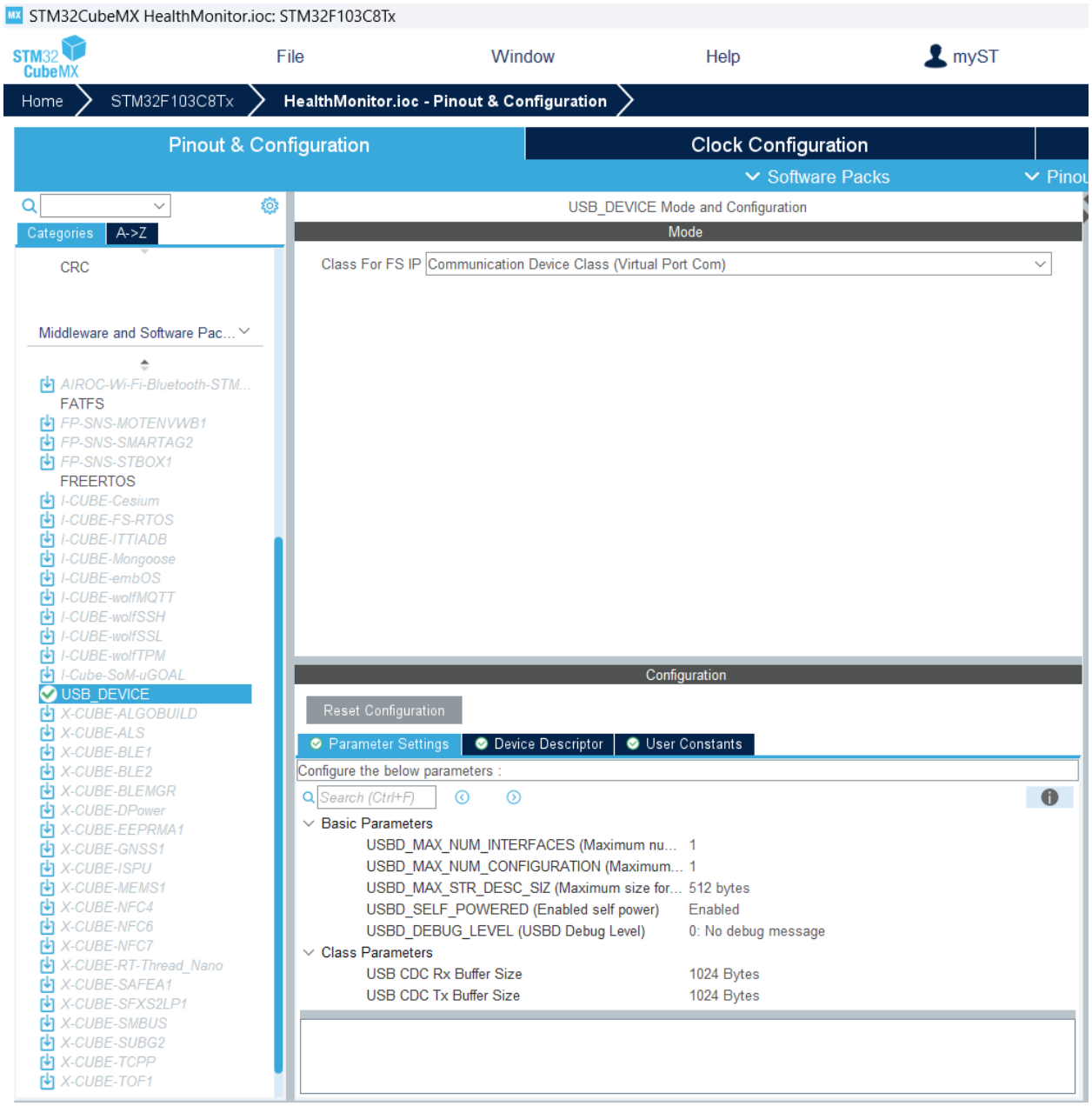
The application was developed using Keil MDK Software for stm32f103c8 microcontroller. And pin configuration was done on STM32 CUBEMX, we are collecting data from AD8232 sensor from ADC to STM32F103 and then data is transmitted to any PC/Raspberry pi.

Step 2: STM32 CUBE MX Pin Configuration

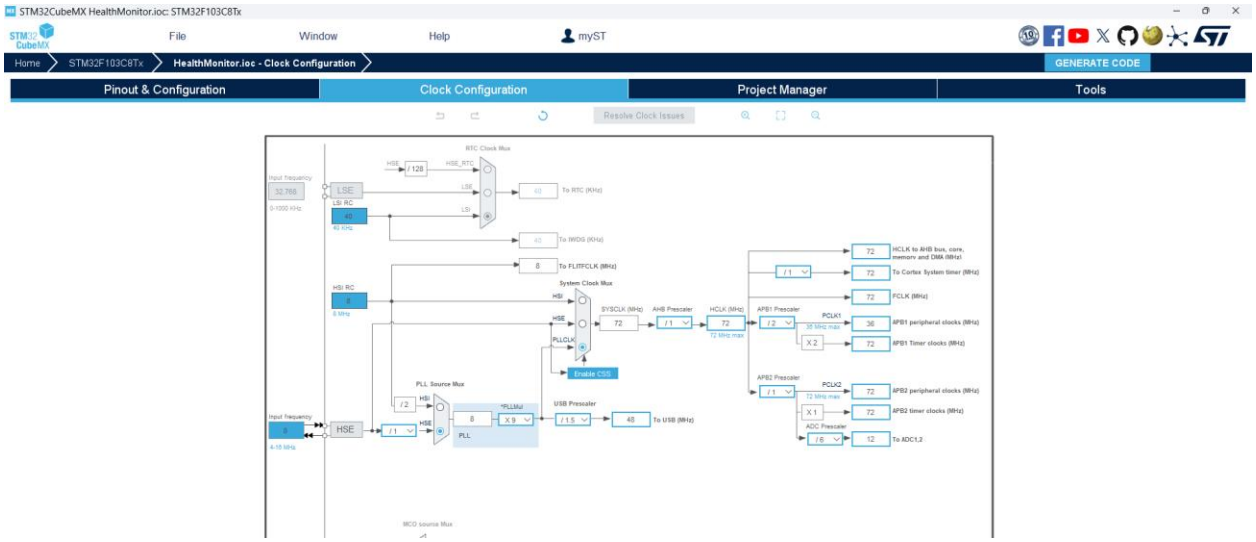
- **USB Configuration on STM32 Blue pill**
  I will use the STM32F103C8 controller in the USB DEVICE mode, and the Communication Device class (Virtual Com Port). The setup for the same is shown below screen shot



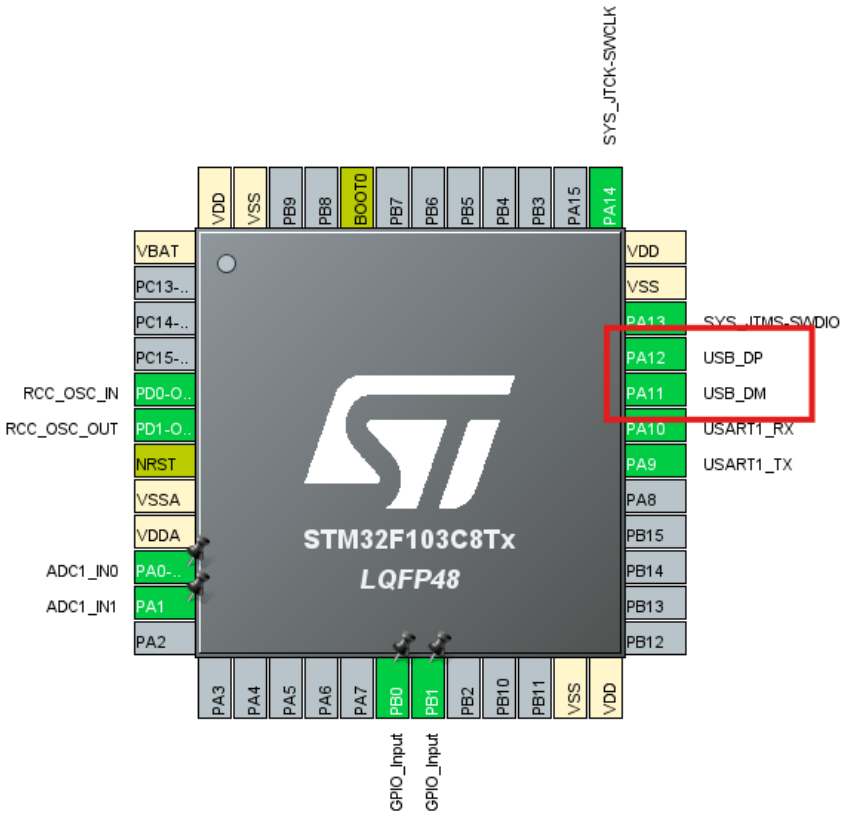
First, I am selecting the USB in DEVICE only MODE, as shown above



in the USB Device, Select the class as Communication Device Class (Virtual Port Com)

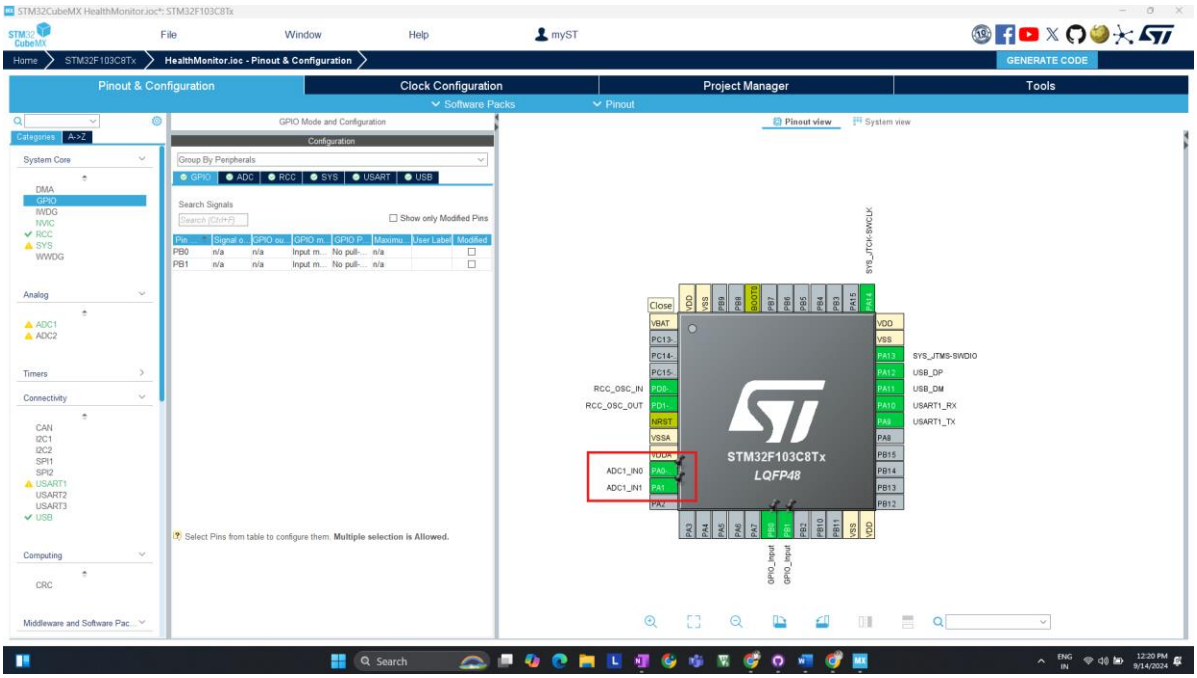


Finally, the clock is set to maximum here. As you can see above, the USB clock is automatically adjusted to 48 MHz

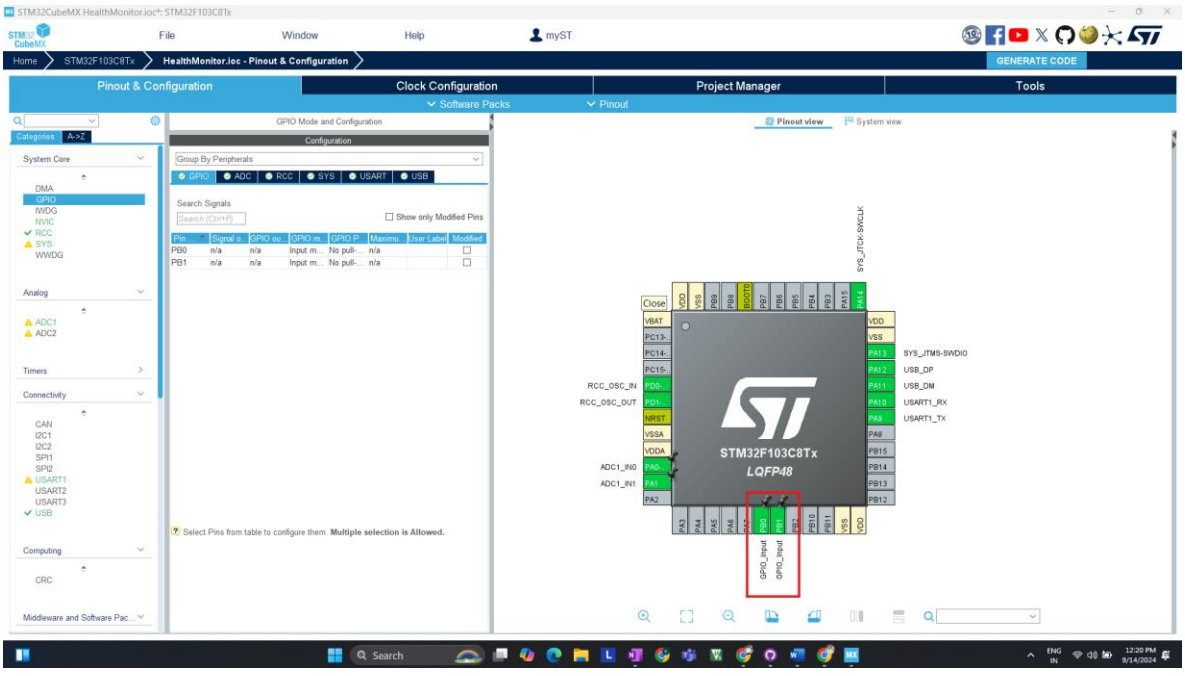


Ensure PA11 (USB_DM) and PA12(USB_DP) are connected for USB functionality

- **Sensor connection and configuration:**



ADC Configuration is done for AD8283 & Heart Pulse detector sensor, AD8232 to **PA1** on STM32 for ECG data capture, Heart Pulse Sensor to **PA0** on STM32 for heart rate measurement.



Use PB0 and PB1 on STM32 to connect to LO- and LO+ on the AD8232 for detecting if leads are properly attached.

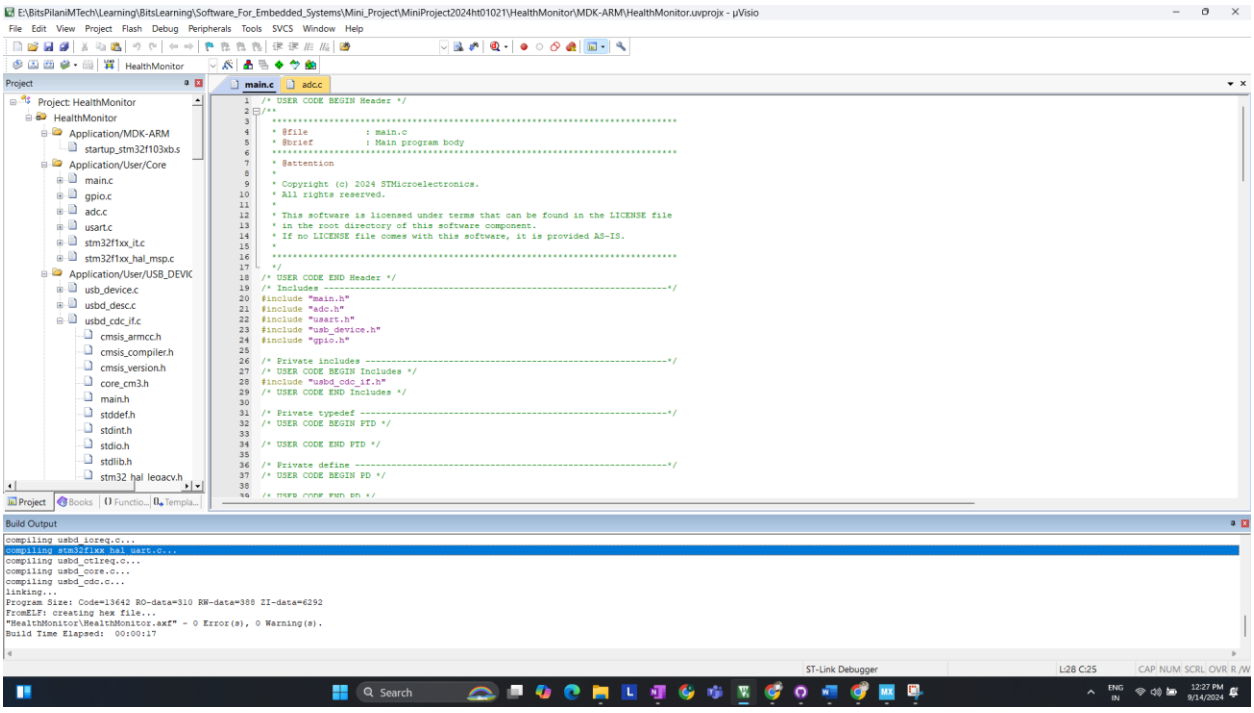
4 Software Integration (Firmware Development & Testing STM32)

Step 1: Environment Setup

- **Tools Required:**
    - Keil V5 for programming and debugging.

- **Procedure:**

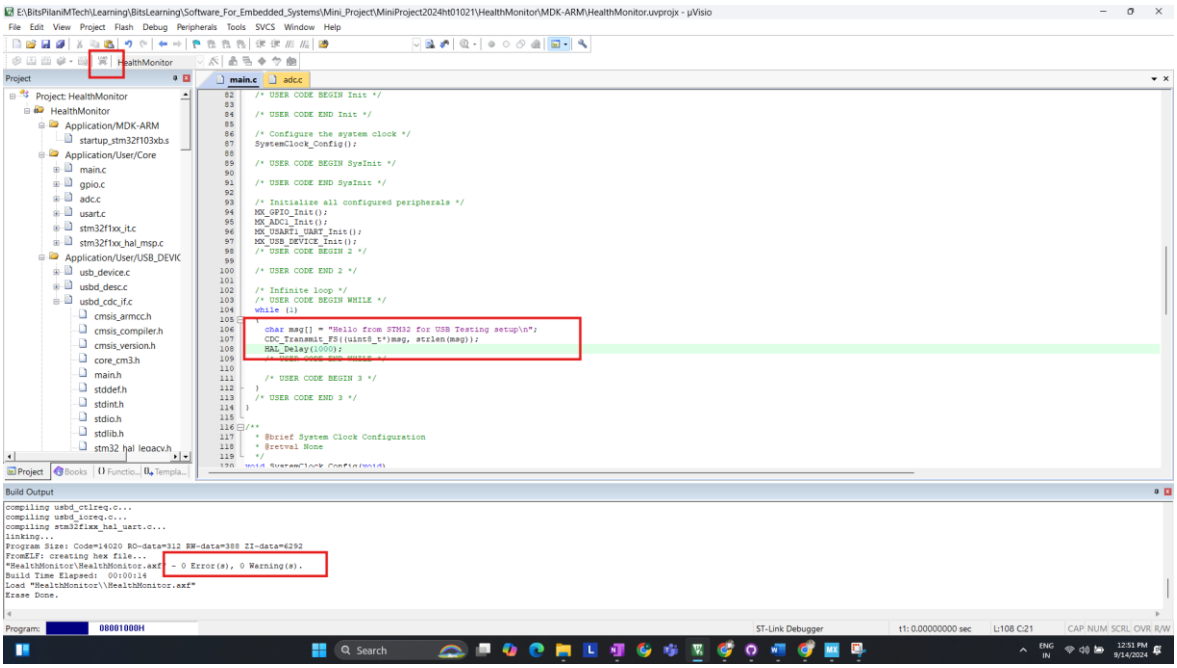
1. Install Keil V5 and configure the project for STM32F103C8Tx.
2. Set up project directories and necessary libraries (HAL, USB CDC).



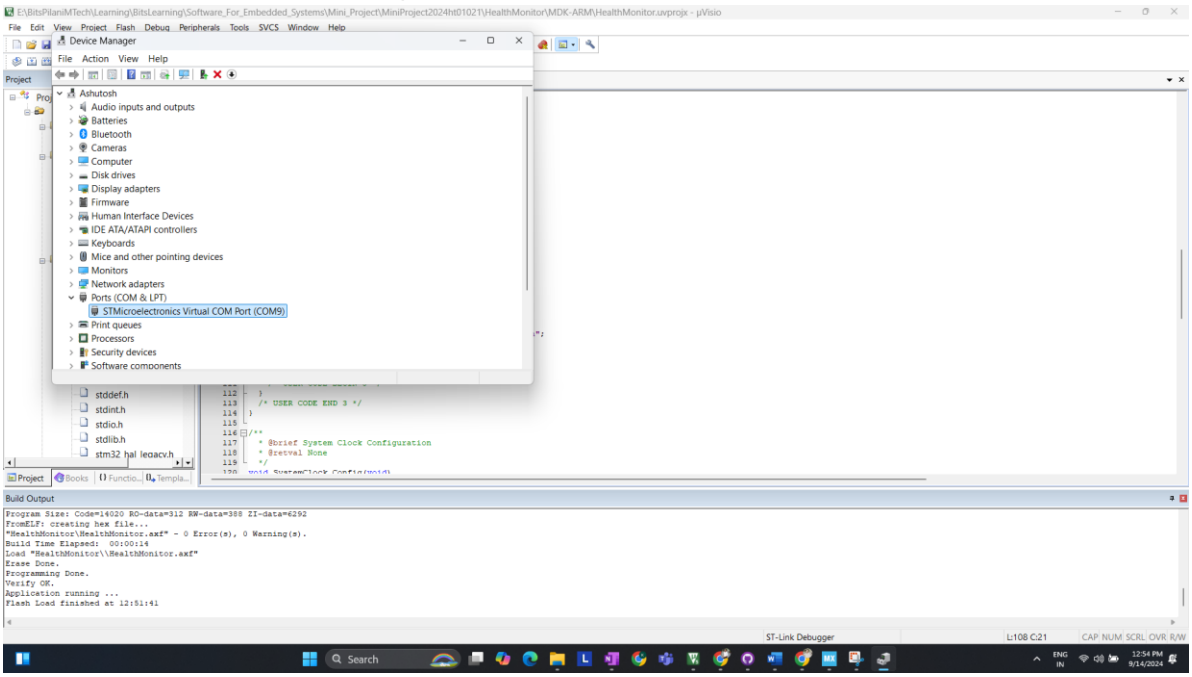
Screen shot of error less code after generating code from STM32 Cube MX.

Step 2: Coding (Initial Basic Setup & Testing)

1. **USB Communication Setup**:
   Implement USB CDC for data transmission to Ubuntu.



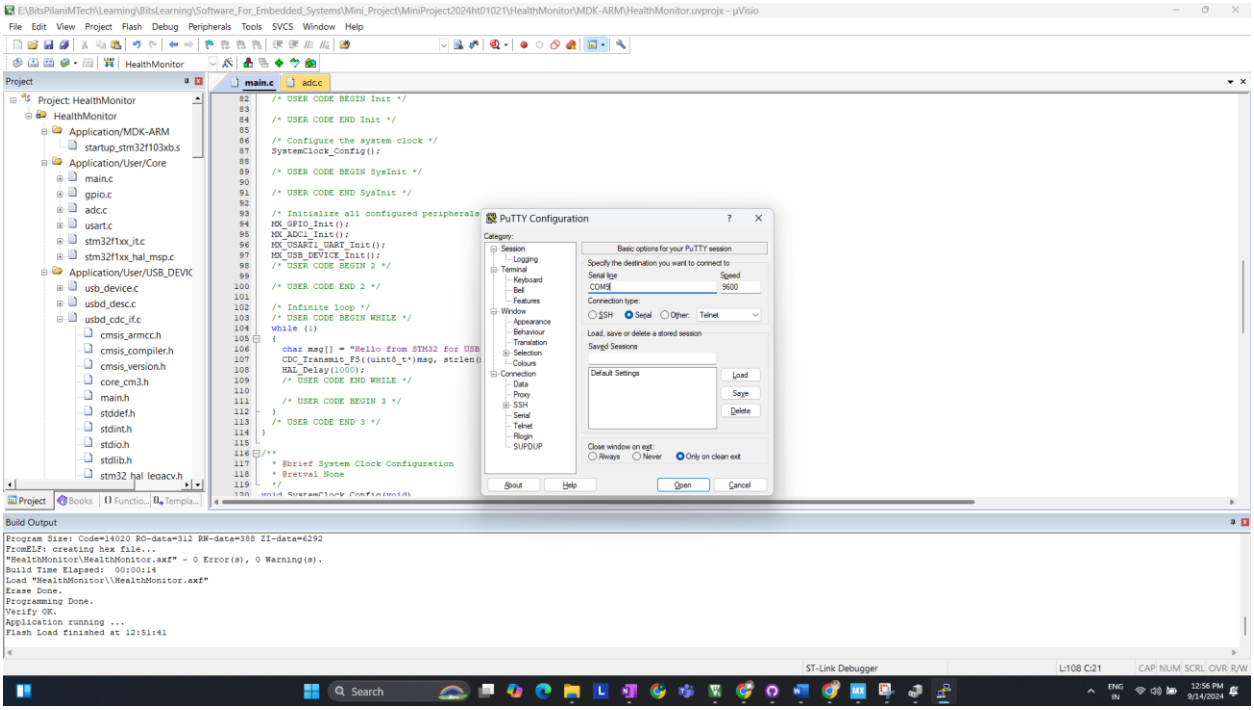
In above screenshot I have compiled and flash code for USB CDC.



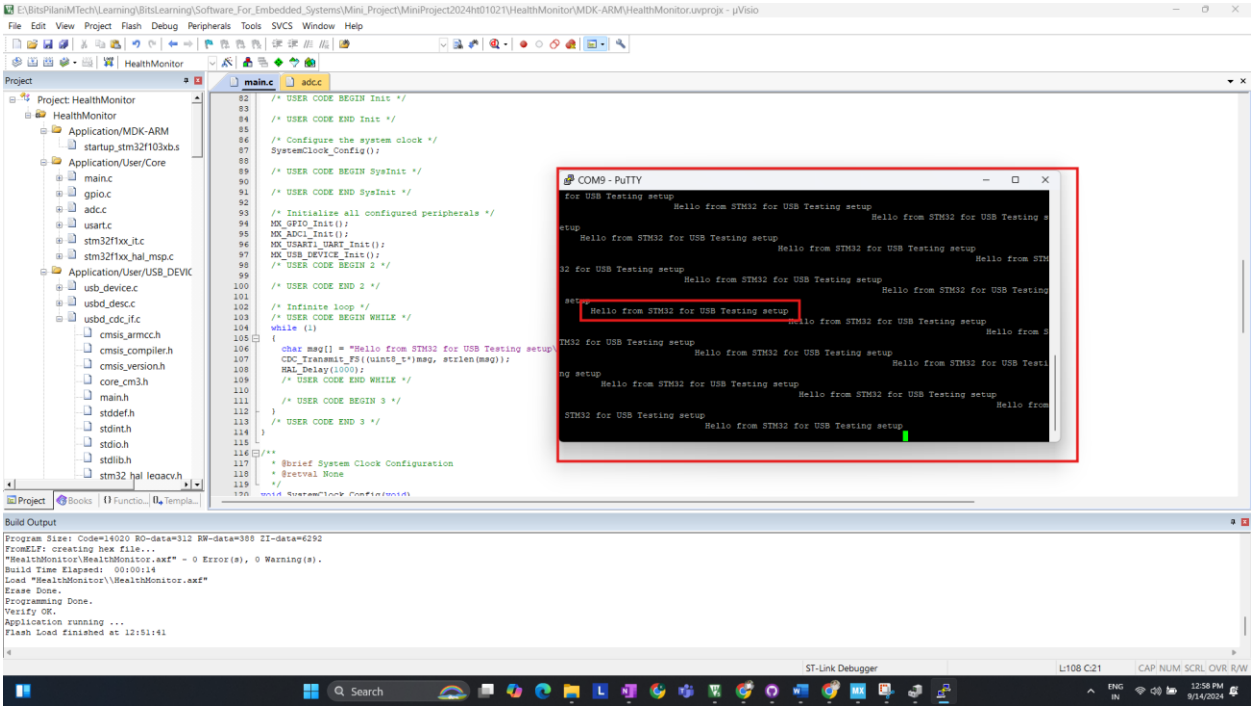
We can detect COM Port (COM9) for Our device (STM32 Blue Pill)

2. Checking with putty:

Configuring Serial to COM9 and setting baud rate to 9600

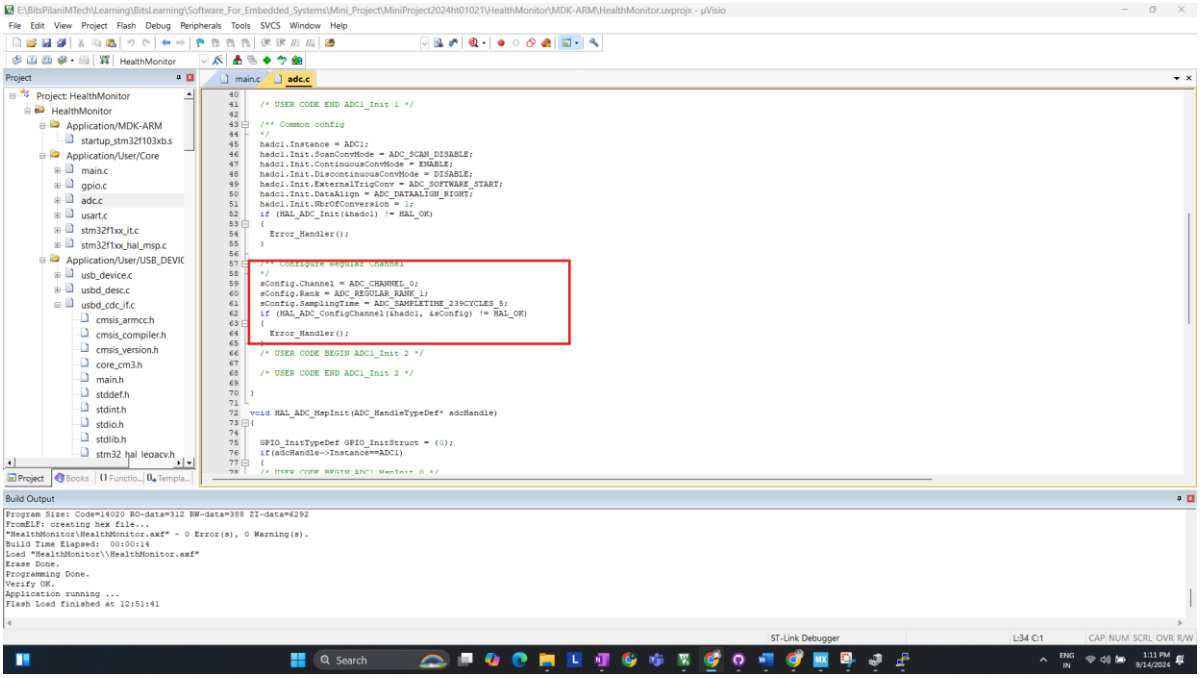


Checking Serial Terminal We are receiving the DATA

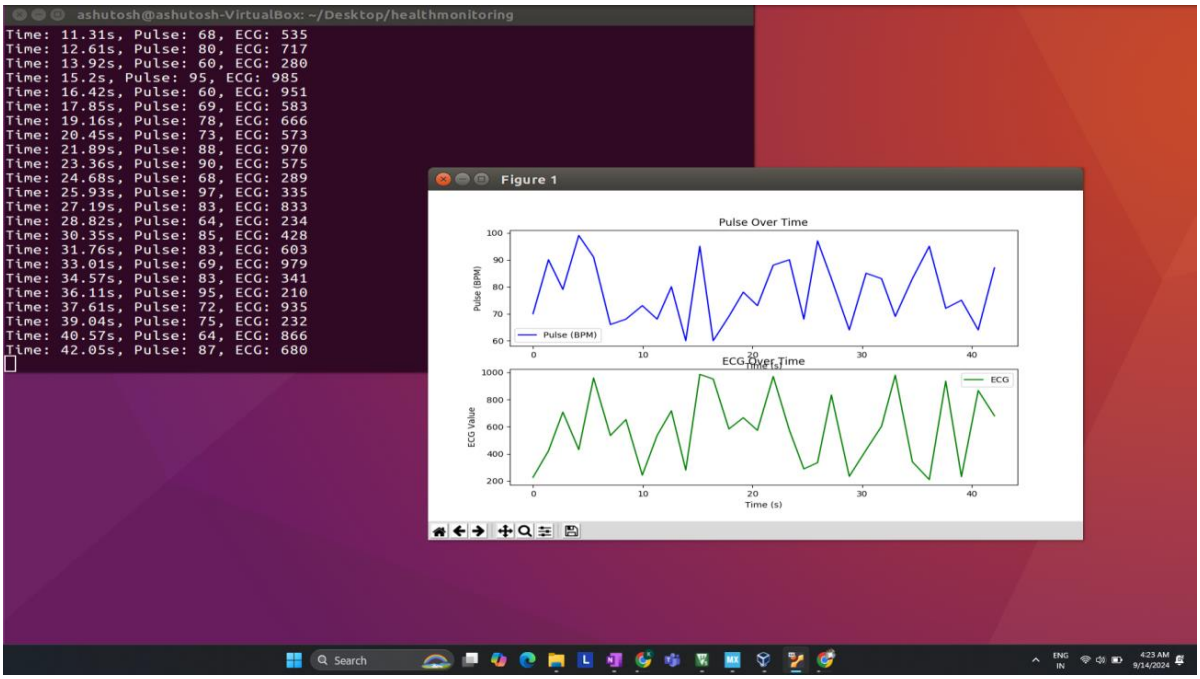


3. ADC Configuration:

The ADC is configured to read analog values from the ECG sensor connected to PA1 and the Heart Pulse sensor connected to PA0. The following code snippet demonstrates how to configure the ADC



4. Testing and debugging on Ubuntu



Initial Fetching of RAW DATA for ECG and PULSE ON UBUNTU

Step 3: Software Development on Ubuntu for Data Processing (Testing)

After completing the firmware development and testing the STM32's ability to transmit data via USB, the next step involves setting up the Python-based system on Ubuntu.

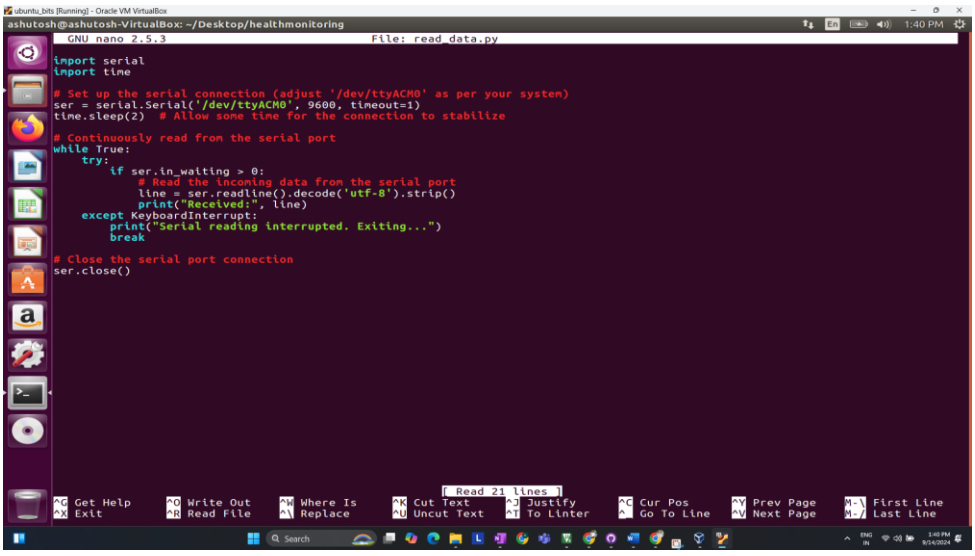
1: Setting Up the Python Environment

Tools Required:

- Python 3.6 or higher
- Libraries:
    - **pyserial** for USB communication
    - **matplotlib** for visualizing ECG and pulse data
    - **numpy** for data handling and signal processing
    - **scipy** (optional) for more advanced signal processing and peak detection

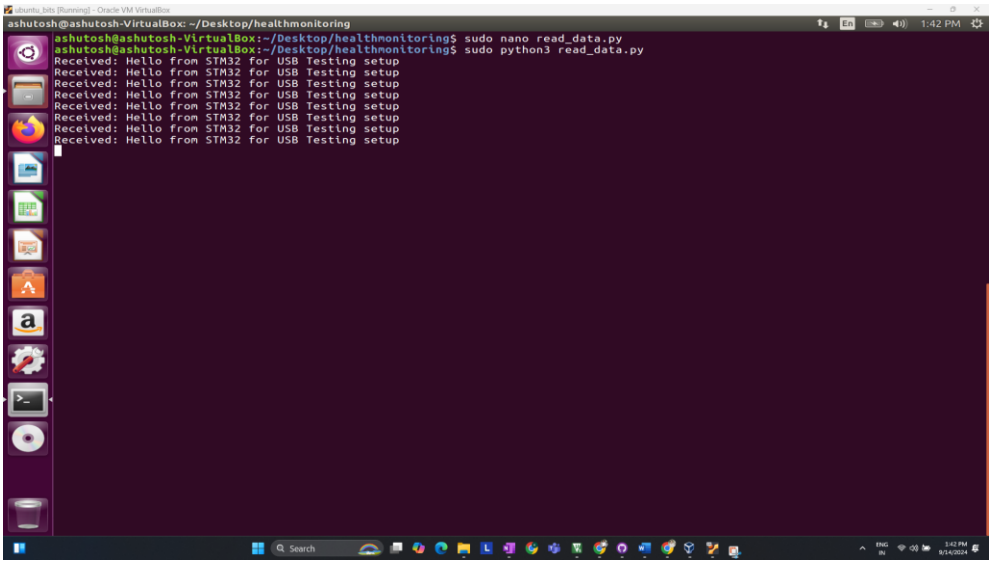
2: Writing Python Code to Read Data from STM32

The Python script will connect to the STM32 via the virtual COM port over USB, read the incoming data, and then process it. Below is a simple script to start reading the data. (Basic program to check connection with STM32 USB CDC)



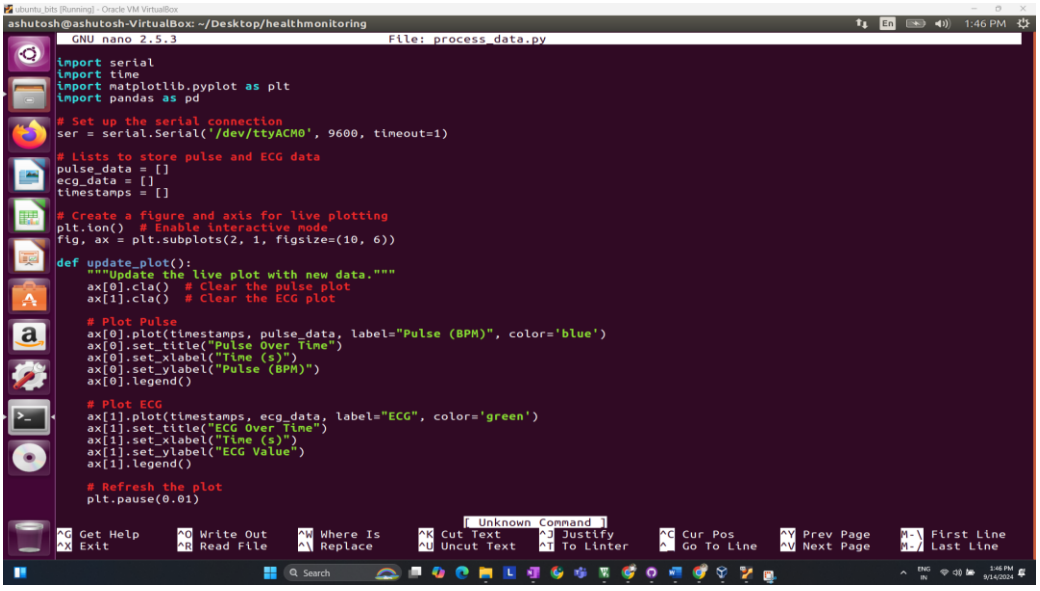
Explanation:

- The serial.Serial() function opens the USB serial port.
- The script reads incoming data from the STM32, which is sent every second, and prints it to the console.



3: Visualizing the Data (Basic Visualization) (Testing)

Once the data is received from STM32, it is important to visualize it. Using matplotlib, you can plot the ECG and heart pulse data in real-time.



```
start_time = time.time()

while True:
    # Read data from the serial port
    try:
        data = ser.readline().decode('utf-8').strip()
        if data:
            # Parse pulse and ECG values from the data string
            parts = data.split(", ")
            pulse = int(parts[0].split(": ")[1])
            ecg = int(parts[1].split(": ")[1])

            # Get the current timestamp
            elapsed_time = round(time.time() - start_time, 2)

            # Append data to the lists
            pulse_data.append(pulse)
            ecg_data.append(ecg)
            timestamps.append(elapsed_time)

            # Update the plot
            update_plot()

            # Print the received data for logging
            print("Time: {}s, Pulse: {}, ECG: {}".format(elapsed_time, pulse, ecg))

        # Add a small delay
        time.sleep(1)

    except KeyboardInterrupt:
        # Save the data to a CSV file on exit
        df = pd.DataFrame({
            "Time (s)": timestamps,
            "Pulse (BPM)": pulse_data,
            "ECG": ecg_data
        })
        df.to_csv("sensor_data.csv", index=False)
        print("Data saved to sensor_data.csv.")
        break
```

Result:

```
Time: 11.31s, Pulse: 68, ECG: 535
Time: 12.61s, Pulse: 80, ECG: 717
Time: 13.92s, Pulse: 60, ECG: 280
Time: 15.2s, Pulse: 95, ECG: 985
Time: 16.42s, Pulse: 60, ECG: 951
Time: 17.85s, Pulse: 69, ECG: 583
Time: 19.16s, Pulse: 78, ECG: 666
Time: 20.45s, Pulse: 73, ECG: 573
Time: 21.89s, Pulse: 88, ECG: 970
Time: 23.36s, Pulse: 90, ECG: 575
Time: 24.68s, Pulse: 68, ECG: 289
Time: 25.93s, Pulse: 97, ECG: 335
Time: 27.19s, Pulse: 83, ECG: 833
Time: 28.82s, Pulse: 64, ECG: 234
Time: 30.35s, Pulse: 85, ECG: 428
Time: 31.76s, Pulse: 83, ECG: 603
Time: 33.01s, Pulse: 69, ECG: 979
Time: 34.57s, Pulse: 83, ECG: 341
Time: 36.11s, Pulse: 95, ECG: 210
Time: 37.61s, Pulse: 72, ECG: 935
Time: 39.04s, Pulse: 75, ECG: 232
Time: 40.57s, Pulse: 64, ECG: 866
Time: 42.05s, Pulse: 87, ECG: 680
```

Figure 1

Pulse Over Time

Pulse (BPM)

ECG Over Time

ECG Value

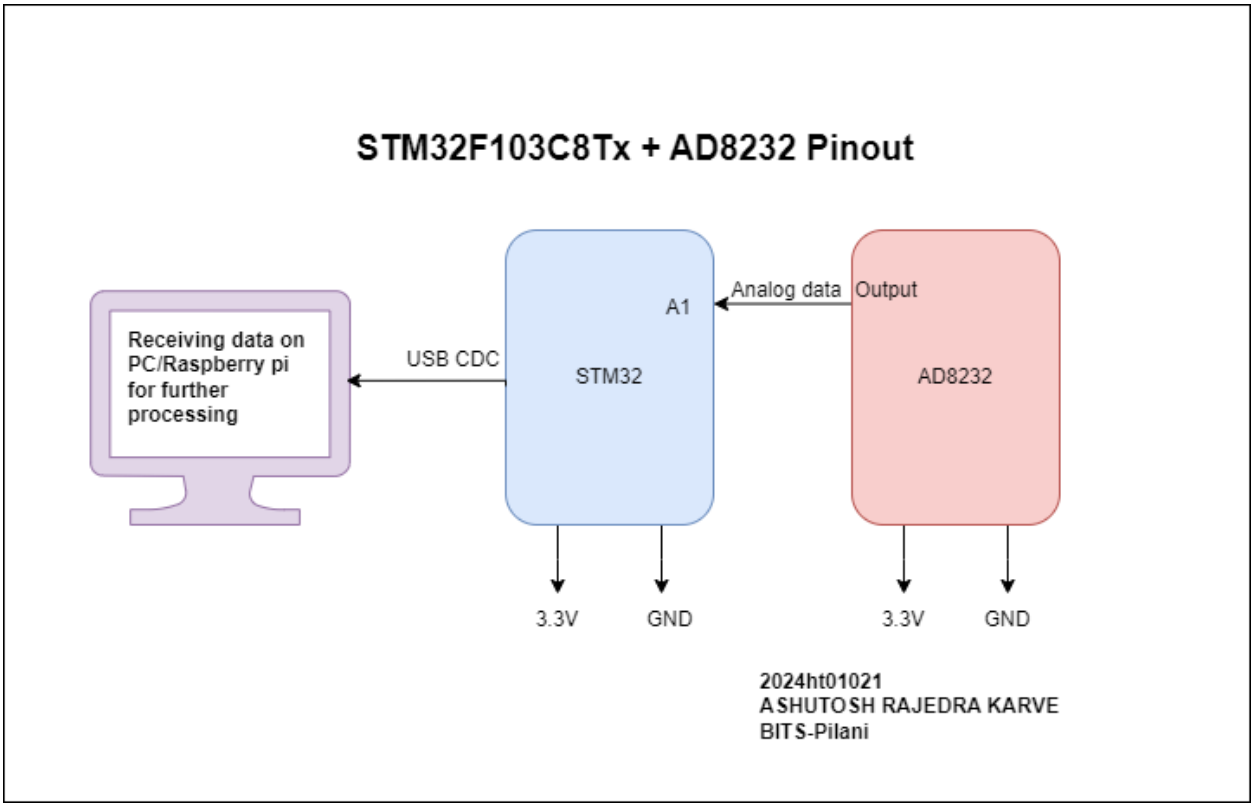
Time (s)

Real-time ECG Plot Testing...

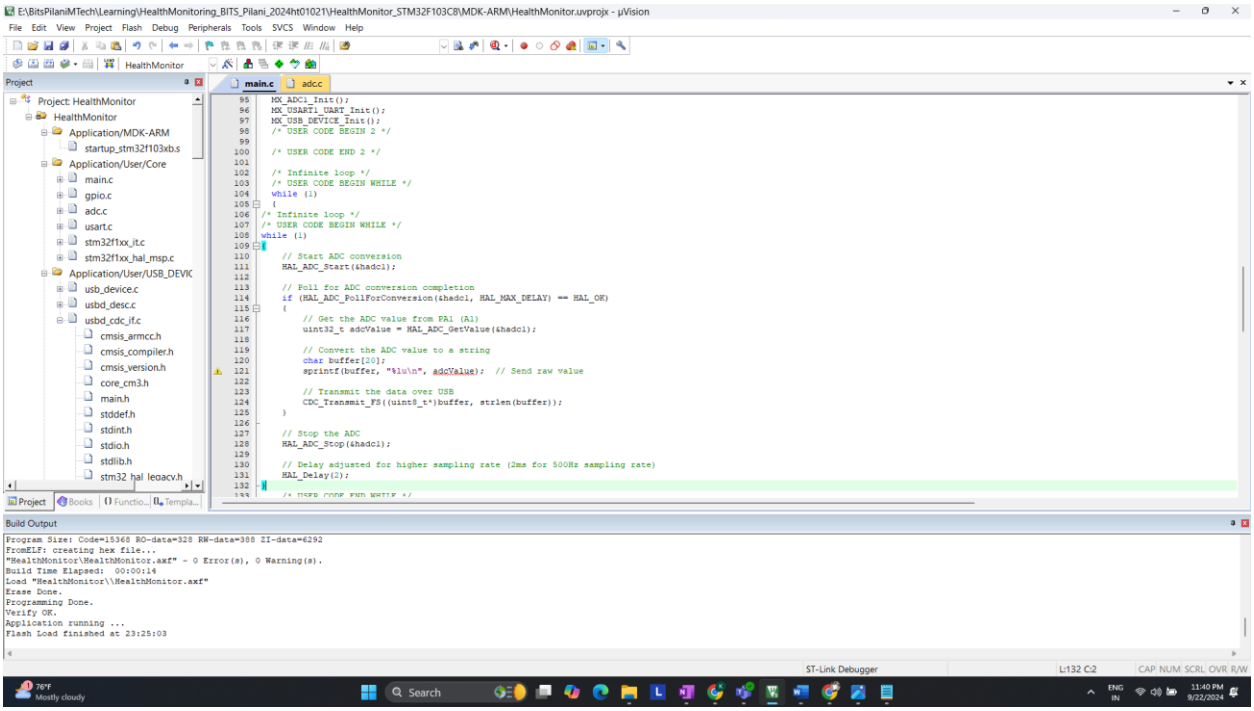
Once the raw data is captured, you can move on to basic data processing and visualization to analyze ECG signals and detect abnormalities.

5 Setting up actual sensor (ECG Sensor) & Final processing code

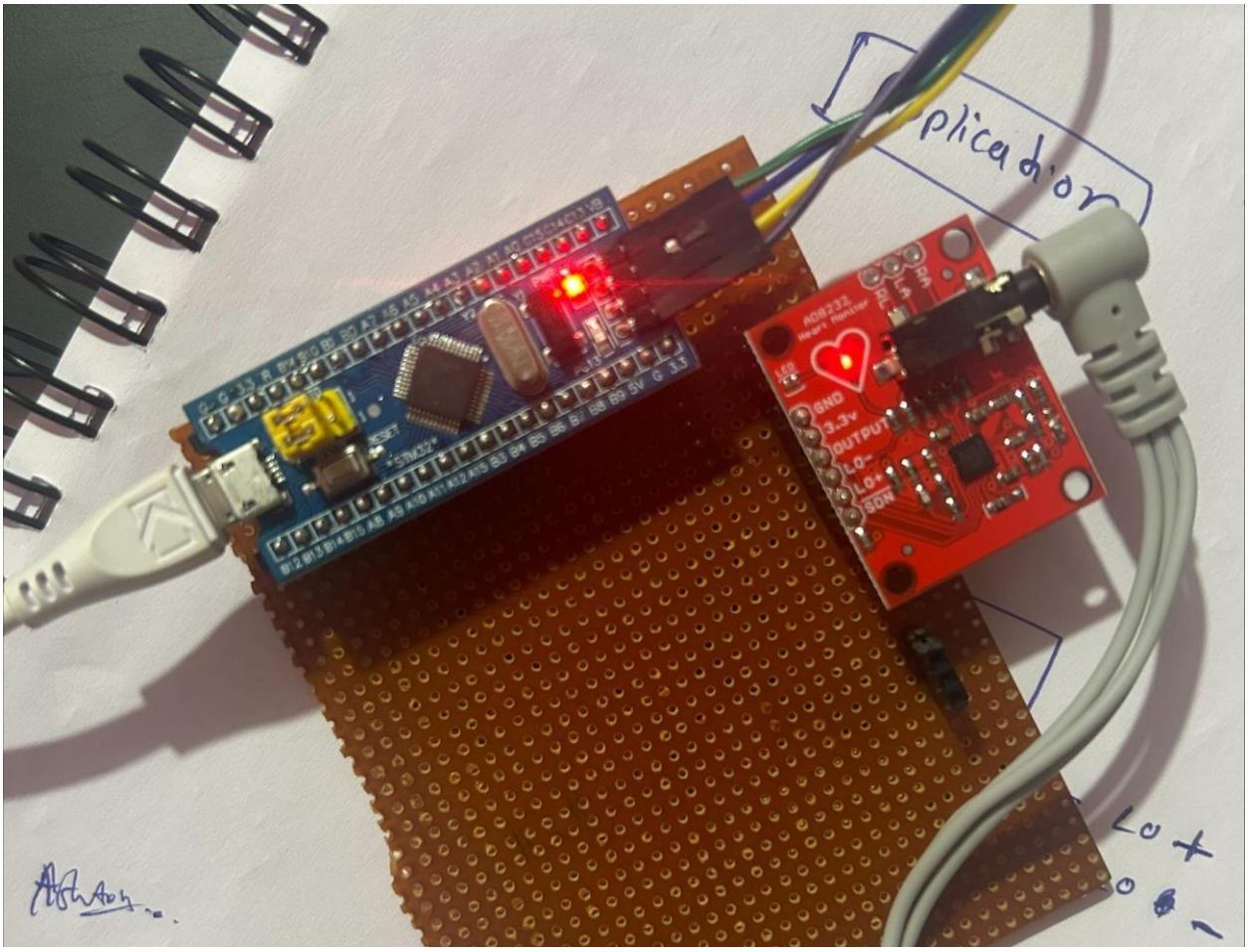
Pin-Configuration Block Diagram of STM32 + AD8232



Code//

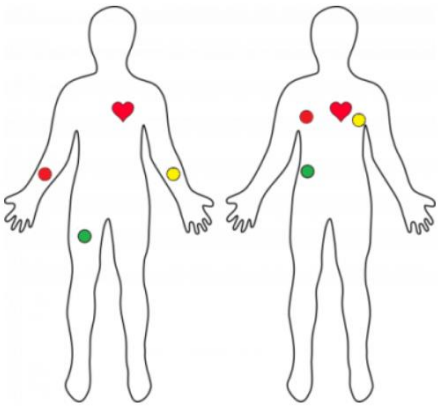


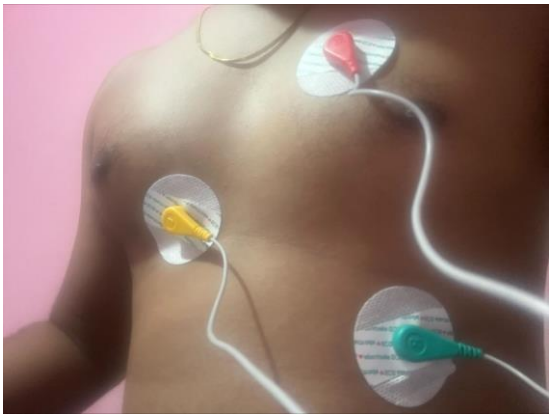
Actual Hardware setup



ECG Sensor Placement on Body

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





Before moving Ahead Let's understand what is ECG?

ECG is the abbreviated term for an electrocardiogram. It is used to record the electrical activity of the heart from different angles to both identify and locate pathology. Electrodes are placed on different parts of a patient's limbs and chest to record the electrical activity.

Parts of the ECG explained

- **P waves**

  P waves represent **atrial depolarization**
  In healthy individuals, there should be a **P wave** preceding each **QRS** complex.

- **PR interval**

  The **PR interval** begins at the **start of the P wave** and ends at **the beginning** of the **Q wave**.
  It **represents** the **time** for electrical activity to move **between the atria and the ventricles**.

- **QRS complex**

  The **QRS complex** represents the **depolarisation of the ventricles**.
  It appears as three closely related waves on the ECG (the Q, R and S wave).

- **ST segment**

  The **ST segment starts** at the **end** of the **S wave** and **ends** at the **beginning** of the **T wave**.
  The **ST segment** is an isoelectric line representing the
  time **between depolarisation** and **repolarisation** of the **ventricles** (i.e. ventricular contraction).

- **T wave**

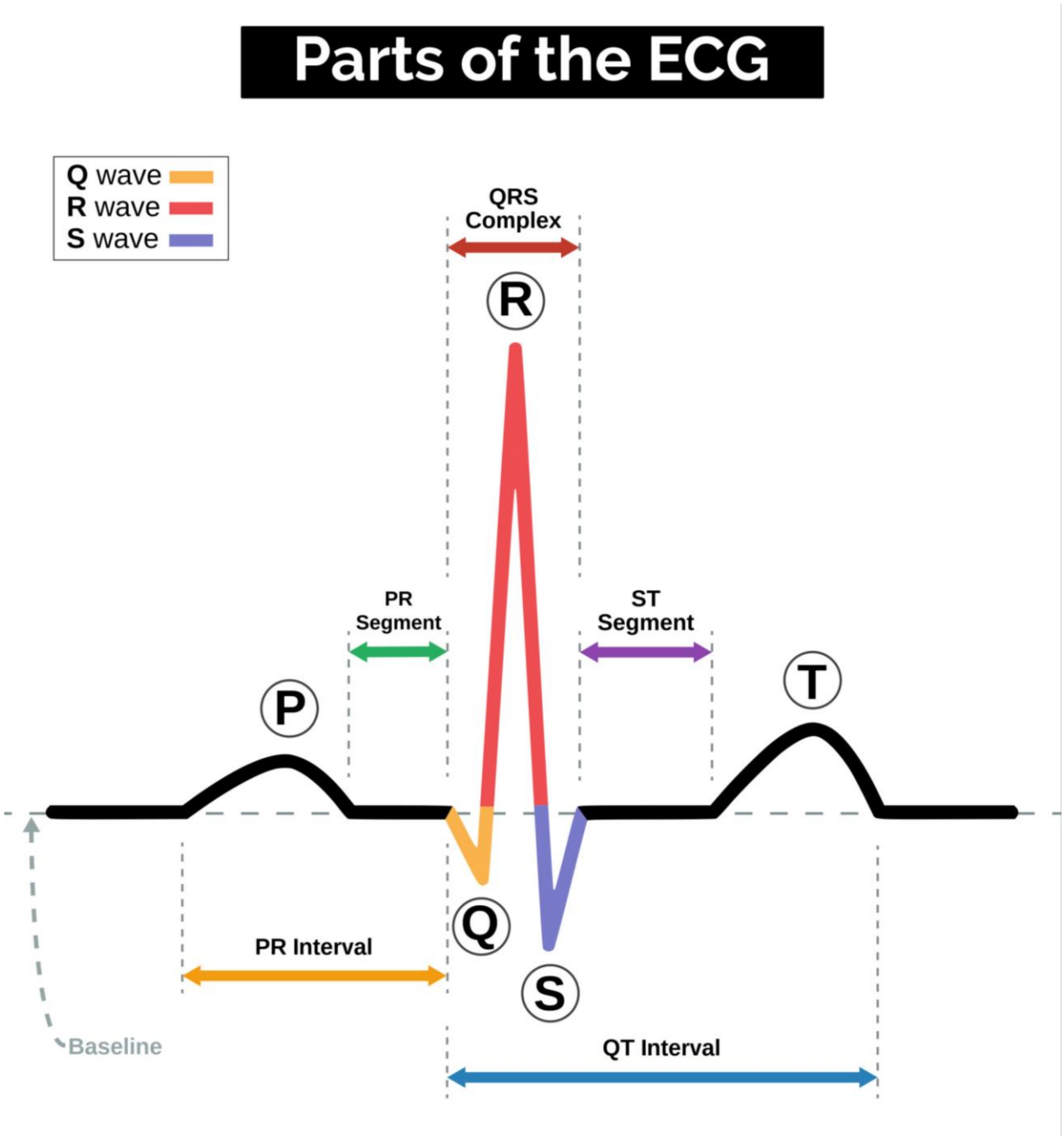
  The **T wave** represents **ventricular repolarisation**.
  It appears as a small wave **after** the **QRS complex**.

- **RR interval**

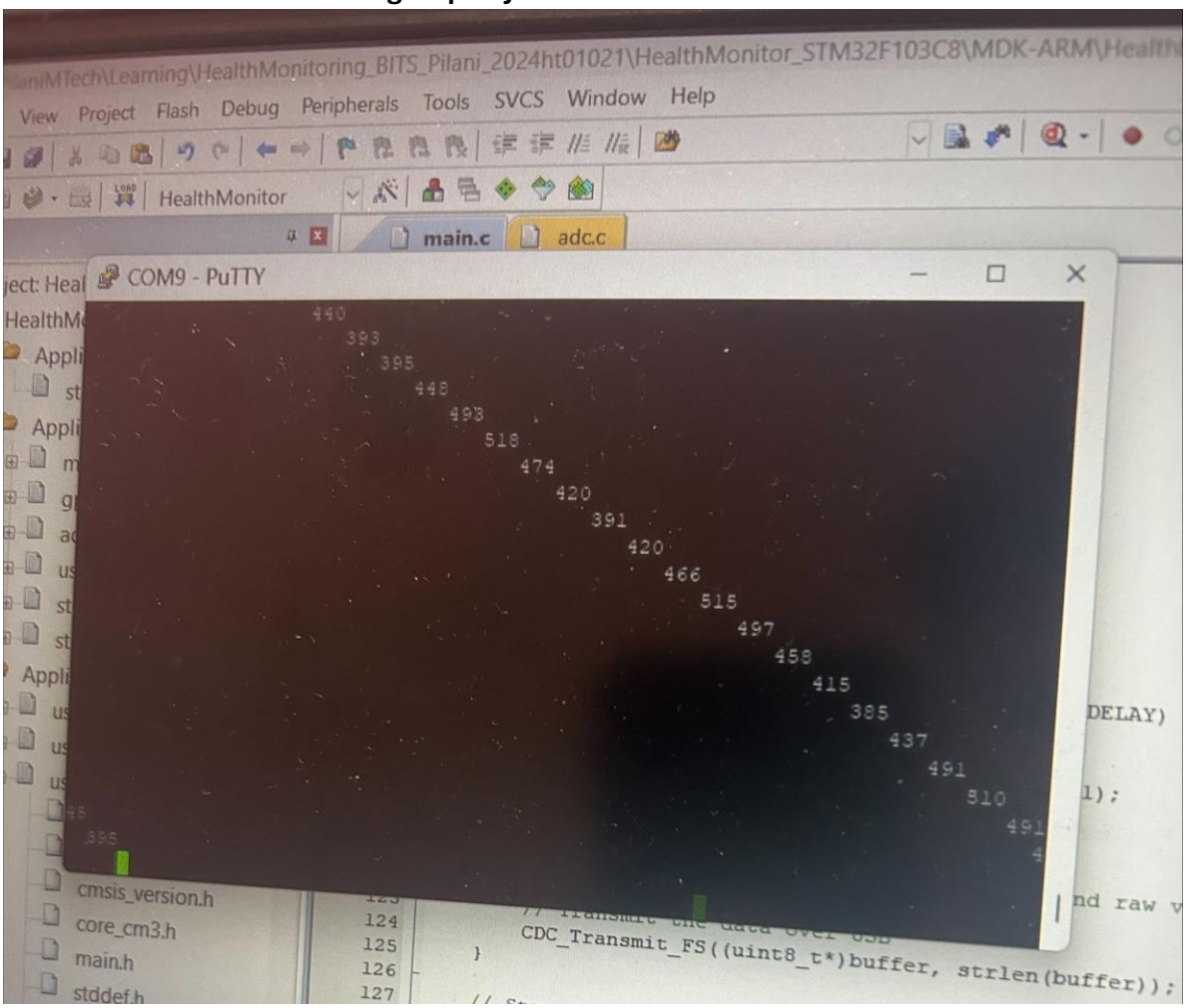
  The **RR interval begins** at the **peak of one R wave** & ends at the **peak of the next R wave**.
  It **represents** the **time between two QRS complexes**.

- **QT interval**

  The **QT interval** begins at the start of the **QRS complex** & finishes at the end of the **T wave**.
  It **represents** the time taken for the **ventricles** to **depolarise** and then **repolarise**.


Parts of the ECG
Q wave
R wave
S wave
QRS Complex
R
PR Segment
ST Segment
P
T
Q
S
PR Interval
Baseline
QT Interval

Raw data which I am receiving on putty



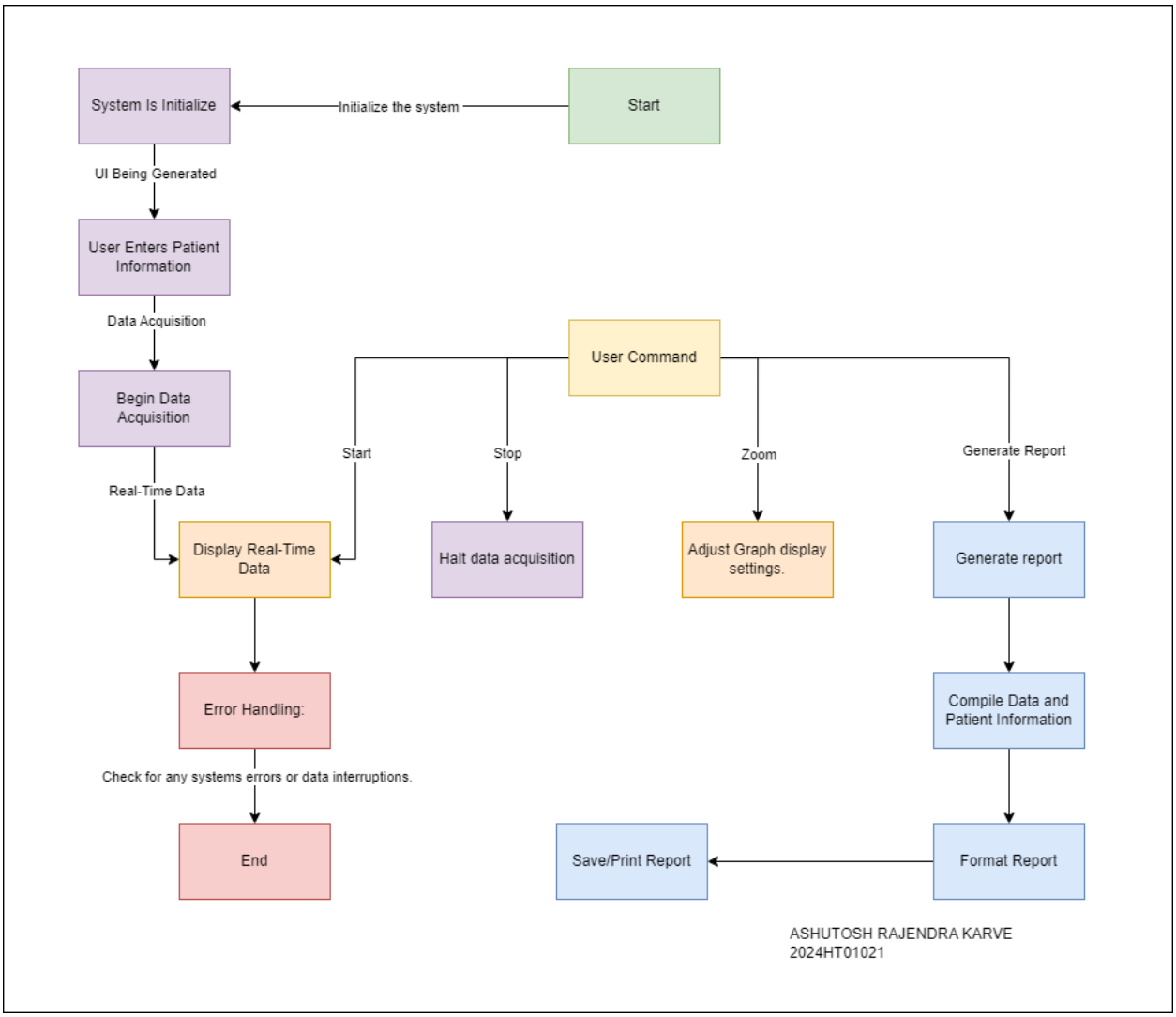
Observations from Putty Display of ECG Data

Interpreting the Data:

- **Fluctuations in the range of 390–530:** The readings captured in the Putty terminal indicate fluctuations between 390 and 530. These fluctuations represent various components of the ECG signal, primarily depicting the heart's electrical activity.
- **QRS Complex:** Values near the higher end of this range, particularly those approaching 530, are likely associated with the QRS complex, indicating ventricular contraction. This is the most prominent and critical part of an ECG trace.
- **P-Wave and T-Wave:** Lower values in this range, closer to 390, could potentially represent the P-wave or T-wave, which are less pronounced than the QRS complex and signify atrial depolarization and ventricular repolarization, respectively.

Periodic Patterns: The data appears to be periodically fluctuating, consistent with the expected rhythmic pattern of a normal ECG, indicating the regular beating of the heart. This rhythmic pattern is essential for analyzing cardiac function and identifying potential abnormalities.

Software Architecture for Data Processing (FLOW CHART)



Data Acquisition: Continuous reading and real-time display of ECG data.

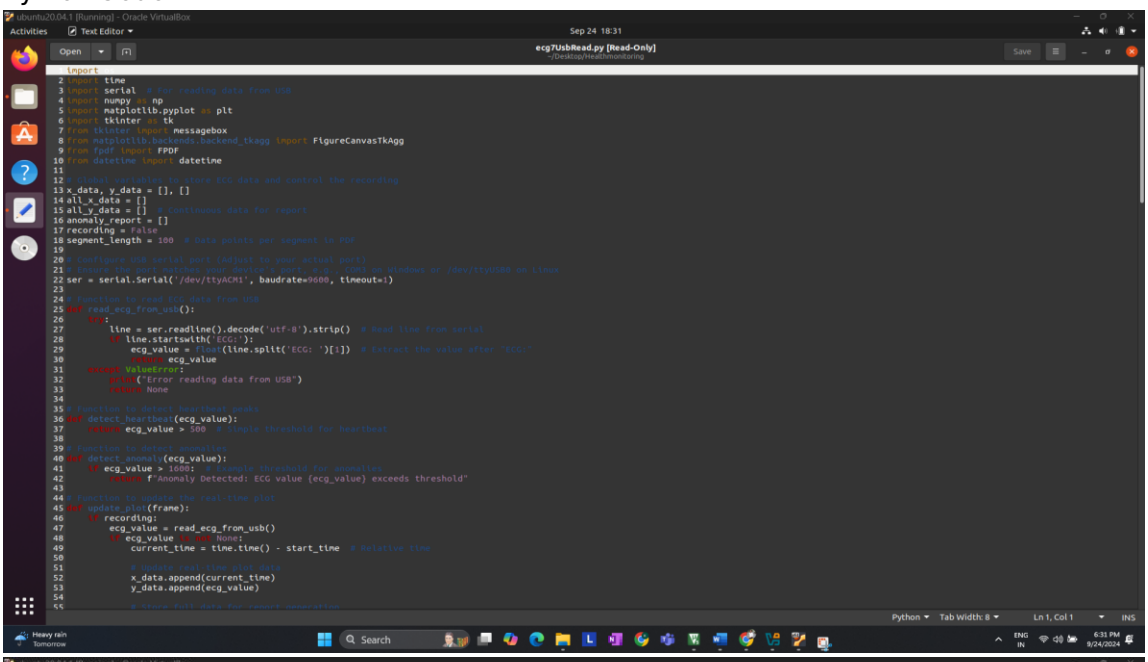
User Interaction: Users can start/stop data acquisition, adjust graph settings, or initiate report generation.

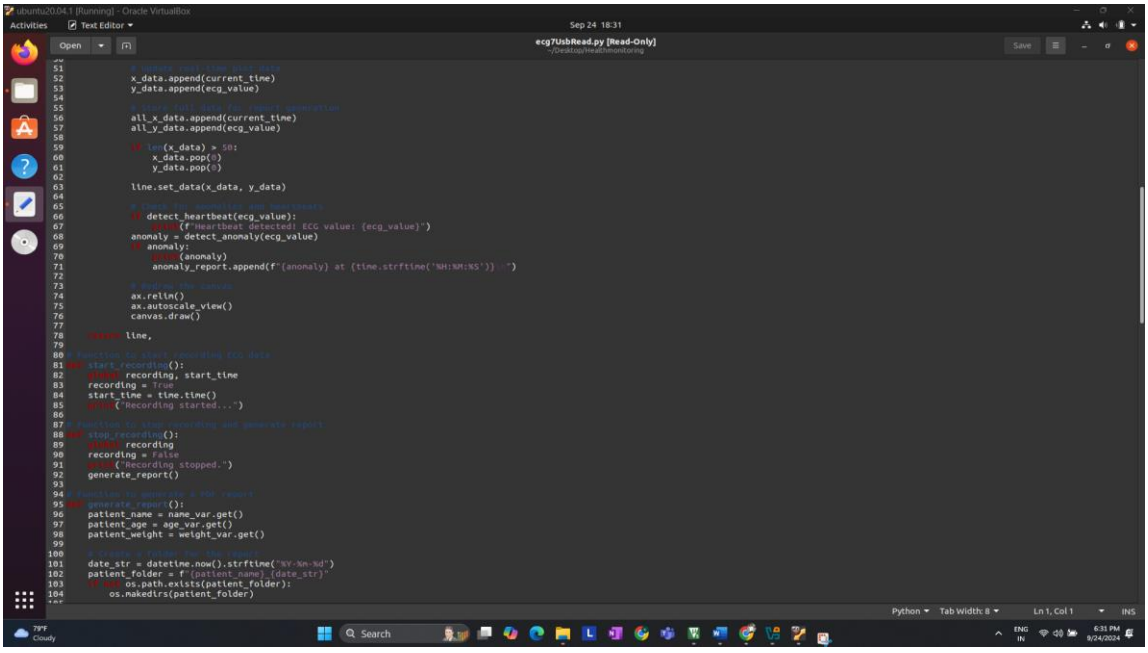
Report Generation: Compiles data and patient information, formats it, and allows for saving or printing.

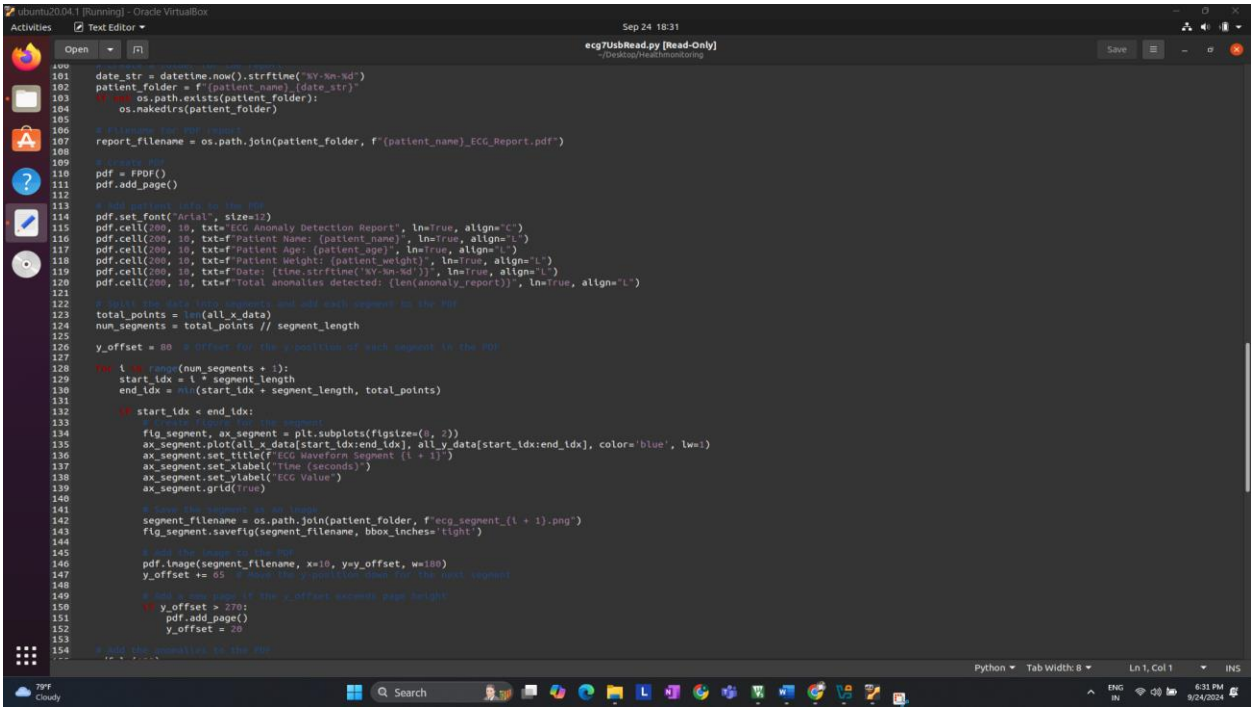
Python Program for further processing and report generation (Enhanced) (FINAL)
Install the following libraries which are crucial for data handling, real-time plotting, and report generation:

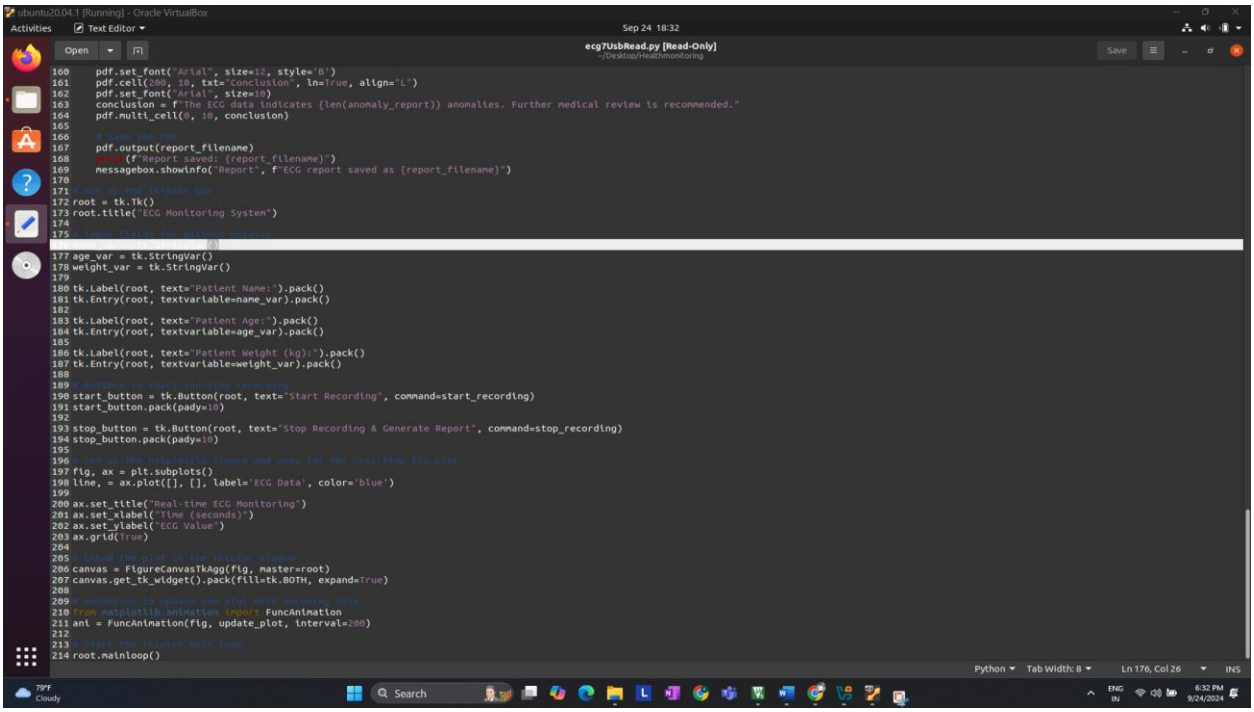
- **matplotlib for data visualization.**
- **numpy for numerical operations.**
- **pyserial for communication with the STM32 device over USB.**
- **fpdf or reportlab for generating PDF reports.**

Python Code>>

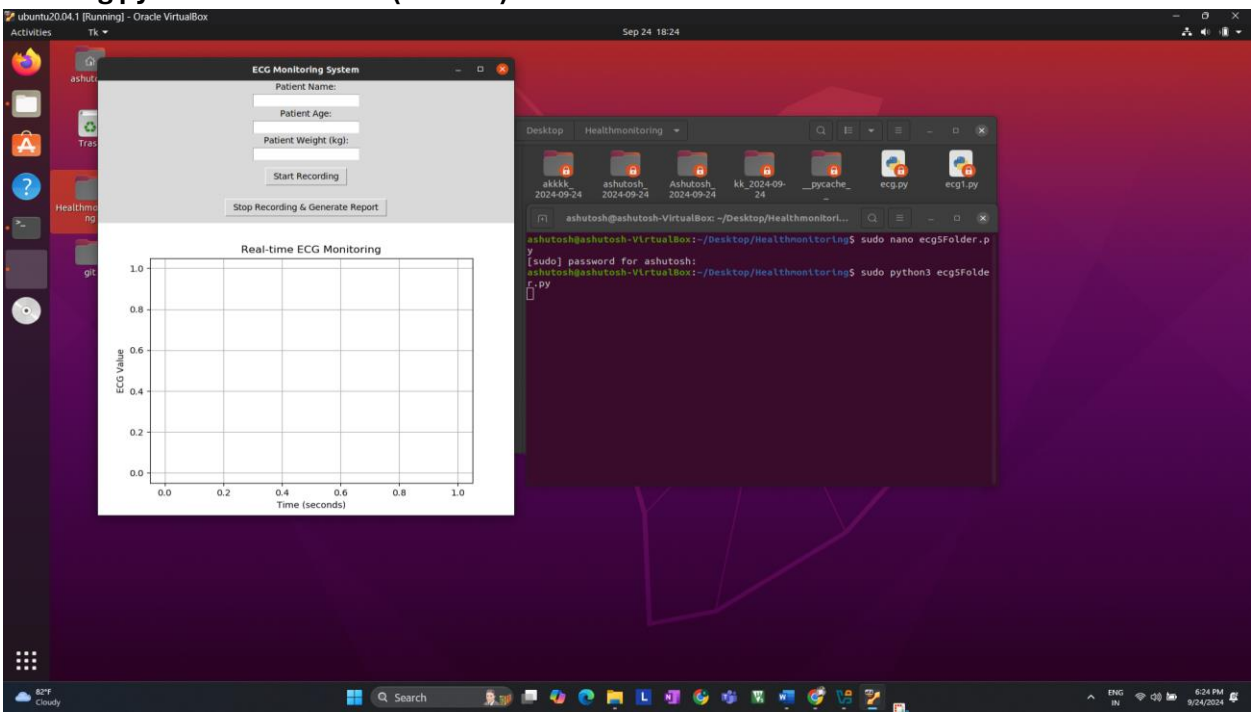




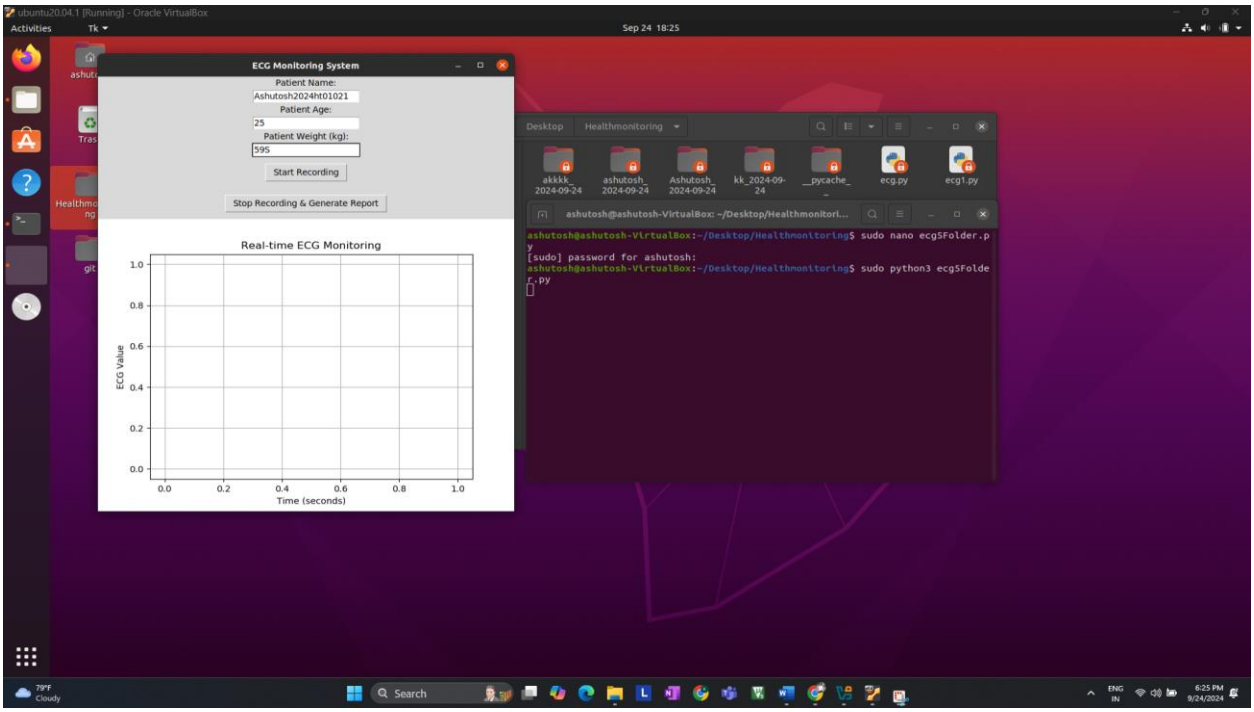

ubuntu20.04.1 [Running] - Oracle VirtualBox
Activities
Text Editor
Sep 24 18:31
Open
ecg7UsbRead.py [Read-Only]
~/Desktop/Healthmonitoring
Save
date_str = datetime.now().strftime("%Y-%m-%d")
patient_folder = f"{patient_name}_{date_str}"
if not os.path.exists(patient_folder):
os.makedirs(patient_folder)
# Filename for PDF report
report_filename = os.path.join(patient_folder, f"{patient_name}_ECG_Report.pdf")
# Create PDF
pdf = FPDF()
pdf.add_page()
# Add patient info to the PDF
pdf.set_font("Arial", size=12)
pdf.cell(200, 10, txt="ECG Anomaly Detection Report", ln=True, align="C")
pdf.cell(200, 10, txt=f"Patient Name: {patient_name}", ln=True, align="L")
pdf.cell(200, 10, txt=f"Patient Age: {patient_age}", ln=True, align="L")
pdf.cell(200, 10, txt=f"Patient Weight: {patient_weight}", ln=True, align="L")
pdf.cell(200, 10, txt=f"Date: {time.strftime('%Y-%m-%d')}", ln=True, align="L")
pdf.cell(200, 10, txt=f"Total anomalies detected: {len(anomaly_report)}", ln=True, align="L")
# Split the data into segments and add each segment to the PDF
total_points = len(all_x_data)
num_segments = total_points // segment_length
y_offset = 80 # Offset for the y-position of each segment in the PDF
for i in range(num_segments + 1):
start_idx = i * segment_length
end_idx = min(start_idx + segment_length, total_points)
if start_idx < end_idx:
# Create figure for the segment
fig_segment, ax_segment = plt.subplots(figsize=(8, 2))
ax_segment.plot(all_x_data[start_idx:end_idx], all_y_data[start_idx:end_idx], color='blue', lw=1)
ax_segment.set_title(f"ECG Waveform Segment {i + 1}")
ax_segment.set_xlabel("Time (seconds)")
ax_segment.set_ylabel("ECG Value")
ax_segment.grid(True)
# Save the segment as an image
segment_filename = os.path.join(patient_folder, f"ecg_segment_{i + 1}.png")
fig_segment.savefig(segment_filename, bbox_inches='tight')
# Add the image to the PDF
pdf.image(segment_filename, x=10, y=y_offset, w=180)
y_offset += 65 # Move the y-position down for the next segment
# Add a new page if the y_offset exceeds page height
if y_offset > 270:
pdf.add_page()
y_offset = 20
# Add the anomalies to the PDF
Python
Tab Width: 8
Ln 1, Col 1
INS
79°F
Cloudy
Search
ENG
IN
6:31 PM
9/24/2024


ubuntu20.04.1 [Running] - Oracle VirtualBox
Activities
Text Editor
Sep 24 18:32
Open
ecg7UsbRead.py [Read-Only]
~/Desktop/Healthmonitoring
Save
pdf.set_font("Arial", size=12, style='B')
pdf.cell(200, 10, txt="Conclusion", ln=True, align="L")
pdf.set_font("Arial", size=10)
conclusion = f"The ECG data indicates {len(anomaly_report)} anomalies. Further medical review is recommended."
pdf.multi_cell(0, 10, conclusion)
# Save the PDF
pdf.output(report_filename)
print(f"Report saved: {report_filename}")
messagebox.showinfo("Report", f"ECG report saved as {report_filename}")
# Set up the Tkinter GUI
root = tk.Tk()
root.title("ECG Monitoring System")
# Input fields for patient details
age_var = tk.StringVar()
weight_var = tk.StringVar()
tk.Label(root, text="Patient Name:").pack()
tk.Entry(root, textvariable=name_var).pack()
tk.Label(root, text="Patient Age:").pack()
tk.Entry(root, textvariable=age_var).pack()
tk.Label(root, text="Patient Weight (kg):").pack()
tk.Entry(root, textvariable=weight_var).pack()
# Buttons to start and stop recording
start_button = tk.Button(root, text="Start Recording", command=start_recording)
start_button.pack(pady=10)
stop_button = tk.Button(root, text="Stop Recording & Generate Report", command=stop_recording)
stop_button.pack(pady=10)
# Set up the Matplotlib figure and axes for the real-time ECG plot
fig, ax = plt.subplots()
line, = ax.plot([], [], label='ECG Data', color='blue')
ax.set_title("Real-time ECG Monitoring")
ax.set_xlabel("Time (seconds)")
ax.set_ylabel("ECG Value")
ax.grid(True)
# Embed the plot in the Tkinter window
canvas = FigureCanvasTkAgg(fig, master=root)
canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
# Animation to update the plot with incoming data
from matplotlib.animation import FuncAnimation
ani = FuncAnimation(fig, update_plot, interval=200)
# Start the Tkinter main loop
root.mainloop()
Python
Tab Width: 8
Ln 176, Col 26
INS
79°F
Cloudy
Search
ENG
IN
6:32 PM
9/24/2024

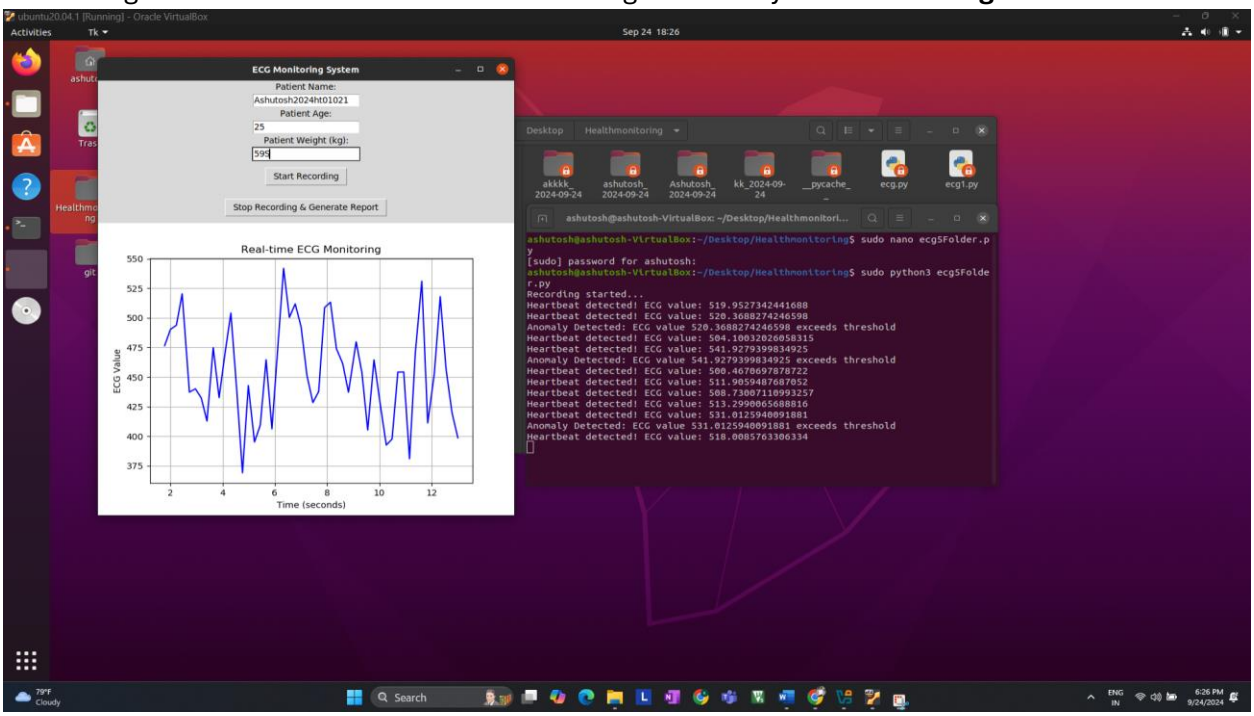
Running python code in Linux(Ubuntu) Machine



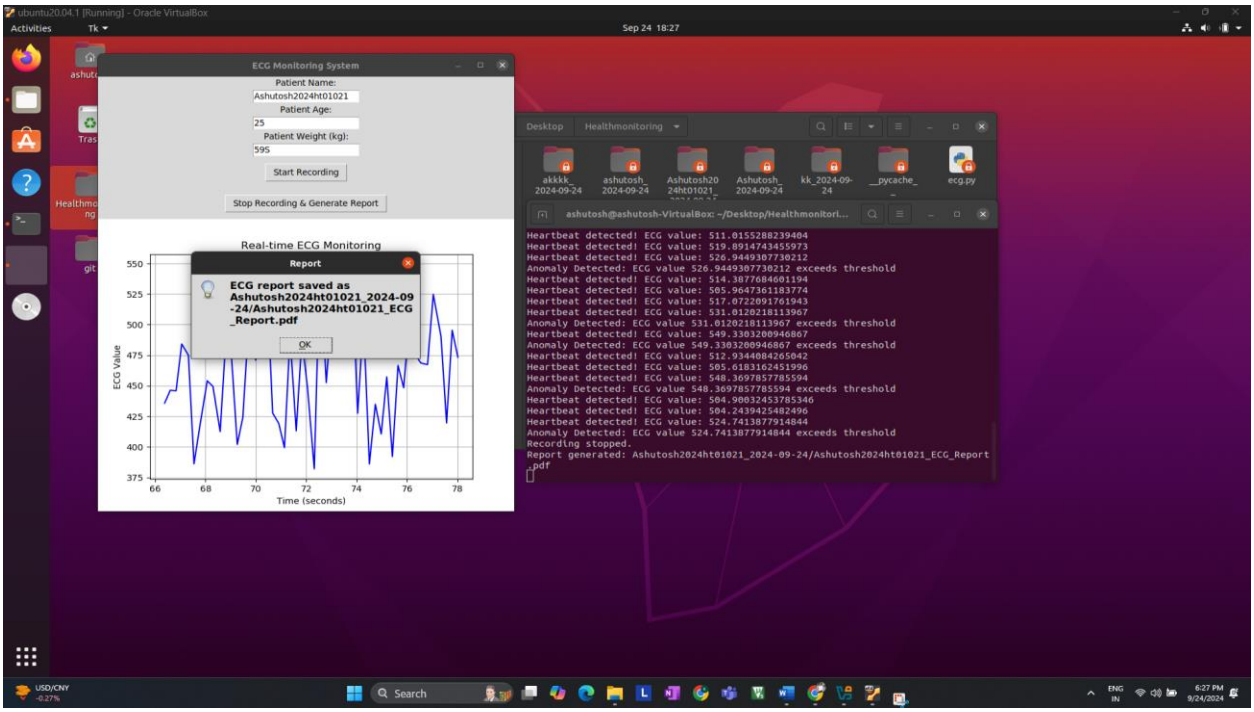
After Running the code, Fill the container with the patient/user details.



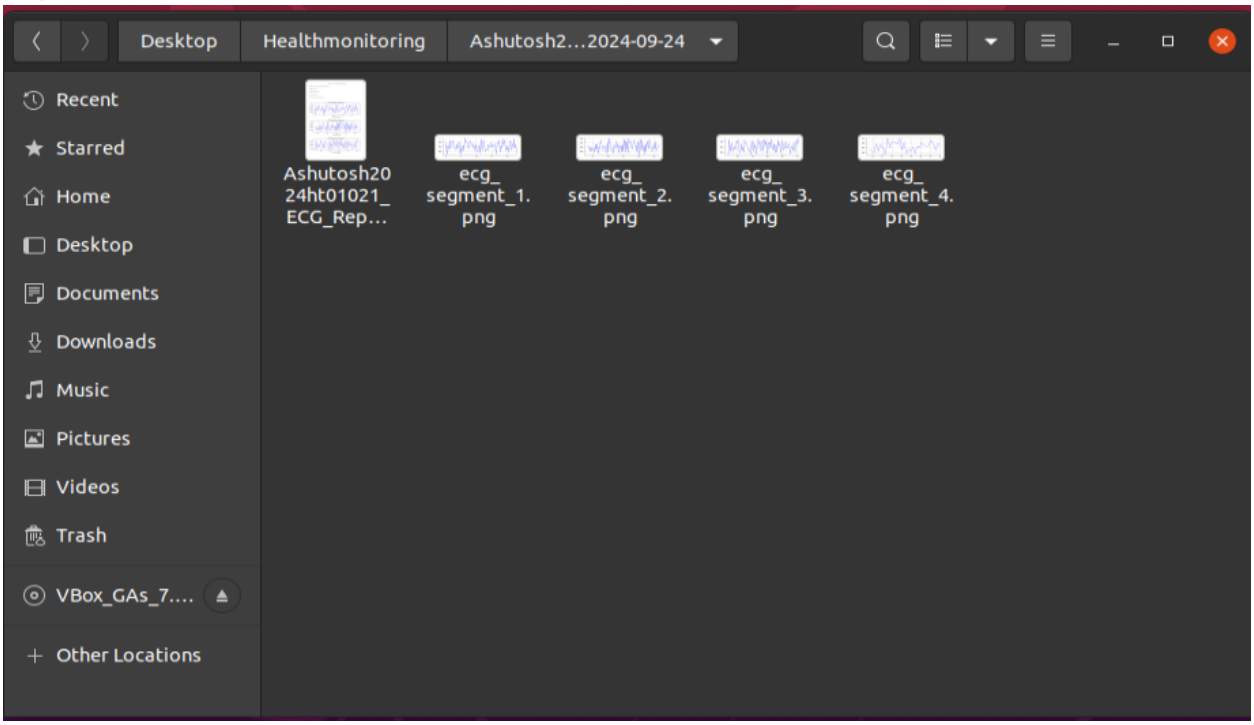
After filling the details and ECG Diodes connecting to the body **start Recording**



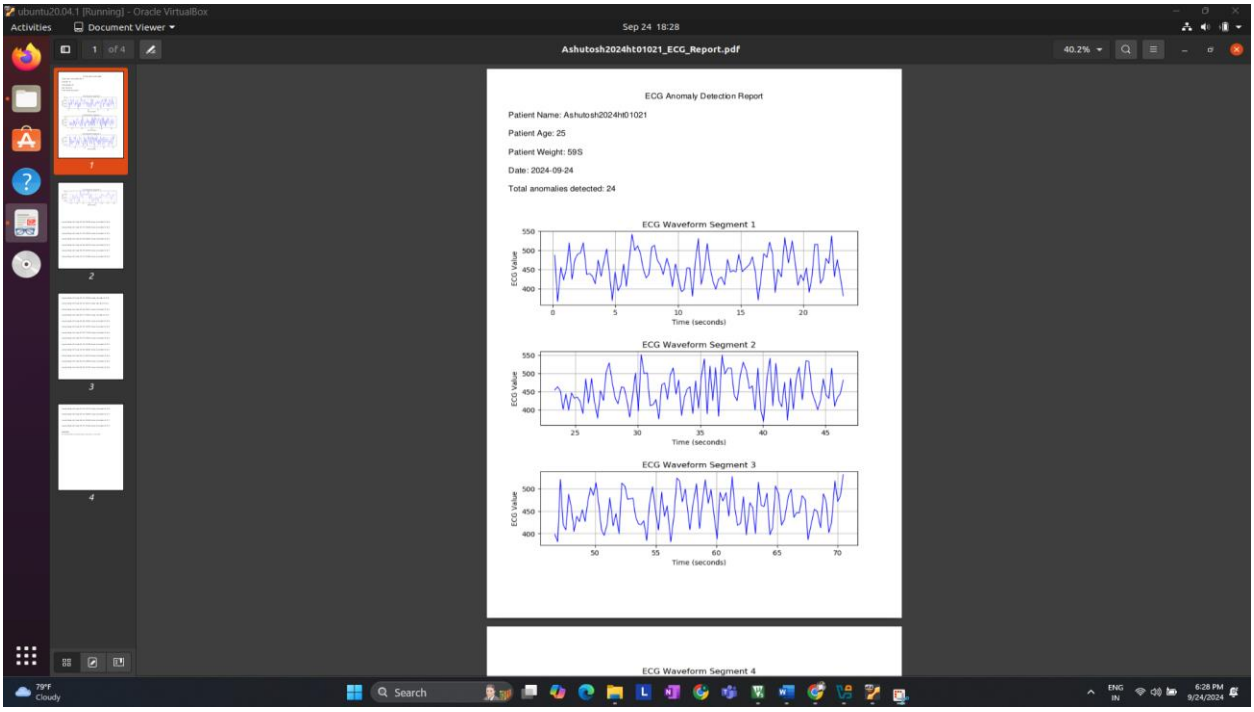
Click **Stop Recording to Generate Report,** then you will ask to save report, click **OK**

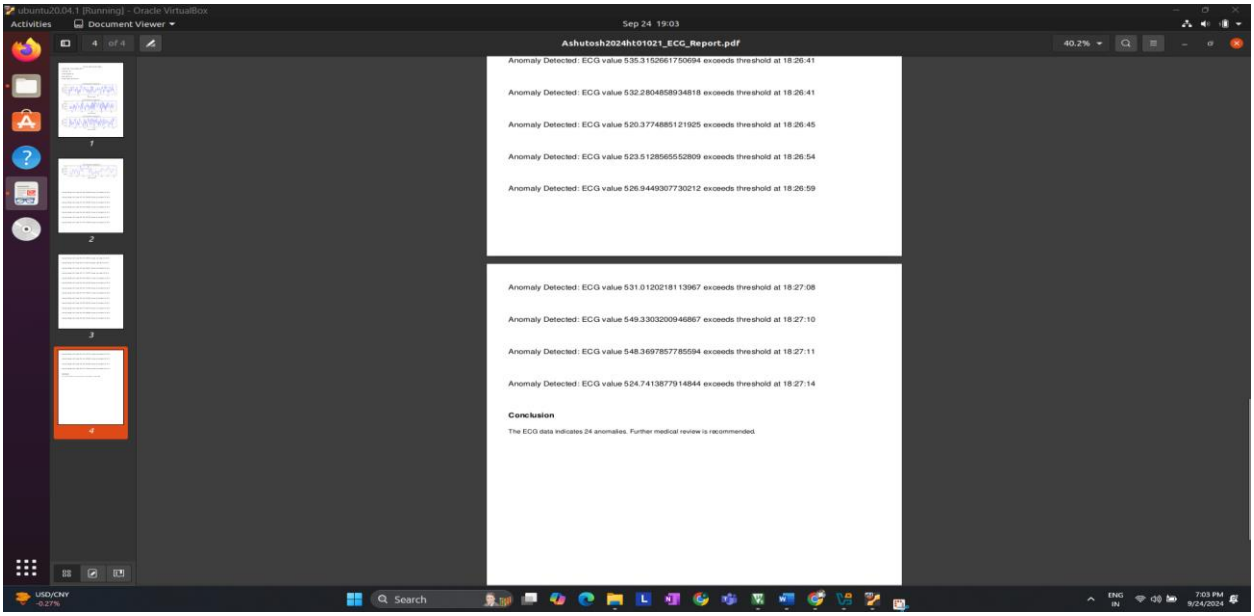


Report will be **saved** on same location in **New Folder** with Patient/User name.



Final Report





6 Challenges and Solutions

Sensor Noise:

One of the major challenges encountered during the development of the ECG monitoring system is the presence of noise and motion artifacts in the ECG signal. These interferences can distort the true heart signal and arise from various sources such as muscle contractions, electrode movement, or external electrical noise. Ensuring proper electrode placement and maintaining steady contact with the skin is essential to mitigate this issue. However, in future versions of the system, software-based filtering will be incorporated to further reduce noise and improve signal quality.

Raw Data Accuracy:

Currently, the raw data captured from the AD8232 ECG sensor is transmitted as unprocessed ADC values. While this provides a direct representation of the analog signal, it does not correspond to standard voltage units (mV) commonly used in ECG analysis. As a result, the raw ADC values can be difficult to interpret without proper scaling. Moving forward, converting these values into meaningful voltage levels will be necessary to facilitate better analysis and comparison of ECG signals.

Data Filtering (Future Plan):

To enhance the quality of the ECG signal, future updates to the system will include the implementation of a smoothing algorithm such as a moving average filter. This filter will help in reducing short-term fluctuations and noise, resulting in a cleaner and more accurate signal for analysis. Additionally, the raw ADC values will be scaled to represent the actual voltages in millivolts (mV), which will make the data more interpretable and allow for easier identification of key ECG components, such as the P-wave, QRS complex, and T-wave.

7 Future Scope

AI-based ECG Analysis:

In future iterations of the system, AI models could be integrated to automatically analyze ECG signals and detect abnormalities like arrhythmias, ischemia, and other heart conditions. Machine learning algorithms, such as convolutional neural networks (CNNs), can be trained on large datasets of ECG signals to identify these irregularities. This would allow for real-time diagnostic feedback, potentially alerting users to abnormal heart rhythms or conditions that require medical attention.

Pulse Sensor Integration:

Another enhancement is the integration of a **Heartbeat Pulse Sensor** to provide additional heart rate monitoring through photoplethysmography (PPG). The combination of both ECG and pulse data would offer a more comprehensive view of cardiovascular health. By analyzing the synchronization between the pulse and ECG signals, future versions of the system could improve the detection of arrhythmias and other irregularities.

Advanced Data Logging:

Future versions will support comprehensive data logging, enabling the storage of ECG signals over time. This would allow for detailed analysis of a patient's heart activity across multiple sessions. The system will generate reports containing graphs, patient information (e.g., name, age, weight), and any detected abnormalities. This data could be stored locally or in the cloud for remote monitoring by healthcare providers.

Wireless Communication:

To enhance portability and usability, future updates may include wireless communication using Bluetooth or Wi-Fi. This will allow the ECG data to be transmitted wirelessly to a smartphone or cloud platform, enabling real-time monitoring without the need for wired connections. This feature would make the system more convenient for patients and medical professionals.

8 Conclusion

Current Status:

The **Advance Health Monitoring System** has successfully demonstrated real-time ECG signal acquisition and basic visualization. The STM32 Blue Pill microcontroller captures raw ECG data from the AD8232 sensor, which is then transmitted over USB and processed using Python. Initial tests show promising results in capturing and displaying ECG waveforms.

Next Steps:

Future development will focus on improving the signal quality through data filtering and scaling, integrating a pulse sensor, and adding advanced features like AI-based ECG analysis. This will allow the system to detect heart abnormalities automatically. Additionally, the system will evolve to include wireless communication for better portability and ease of use.

Impact:

This project has the potential to develop into a low-cost, portable, and comprehensive health monitoring system. It can be used to assist in early detection of heart issues, helping individuals and healthcare providers track cardiovascular health more effectively. By continuously expanding its functionality, the system could become a valuable tool for non-invasive heart monitoring in both clinical and personal health settings.

9 References

1. STMicroelectronics. STM32 Blue Pill Documentation: https://www.st.com/en/microcontrollers-microprocessors/stm32-bluepill.html
2. Analog Devices. AD8232 Single-Lead Heart Rate Monitor Front End: https://www.analog.com/en/products/ad8232.html
3. Python Software Foundation. Python Documentation: https://docs.python.org/3/
4. PySerial Library Documentation: https://pyserial.readthedocs.io/en/latest/
5. Matplotlib Library Documentation: https://matplotlib.org/stable/index.html
6. Heart Rate Variability and ECG Analysis using AI, IEEE Conference Paper: https://ieeexplore.ieee.org/document/9173269
7. "Photoplethysmography for Pulse Detection in Wearable Devices," Journal of Biomedical Engineering: https://www.jbiomecheng.org/ppg-sensors

GITHUB PROJECT LINK:
https://github.com/Ashutoshkarve007/HealthMonitoring_BITS_Pilani_2024ht01021.git

GITHUB PROFILE:
https://github.com/Ashutoshkarve007

LINKDIN LINK:
https://in.linkedin.com/in/ashutoshkarve

MY YOUTUBE CHANNEL:
https://www.youtube.com/@ashutoshkarve698