

# **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](mailto: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](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:

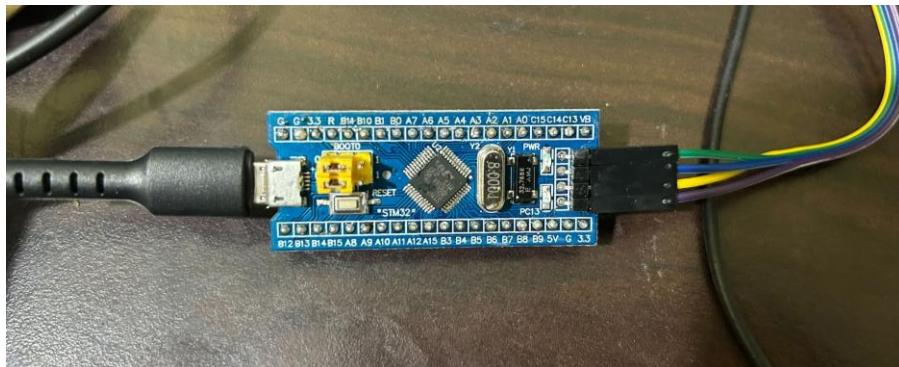
- **STM32 Firmware:** The firmware reads raw ADC data from the ECG sensor connected to the STM32's PA1 (A1) pin. The raw data (0-4095 for 12-bit ADC) is transmitted to a PC via USB.
- **Python Interface:** The Python code reads the raw ECG data from the STM32 and plots the real-time ECG signal. The data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

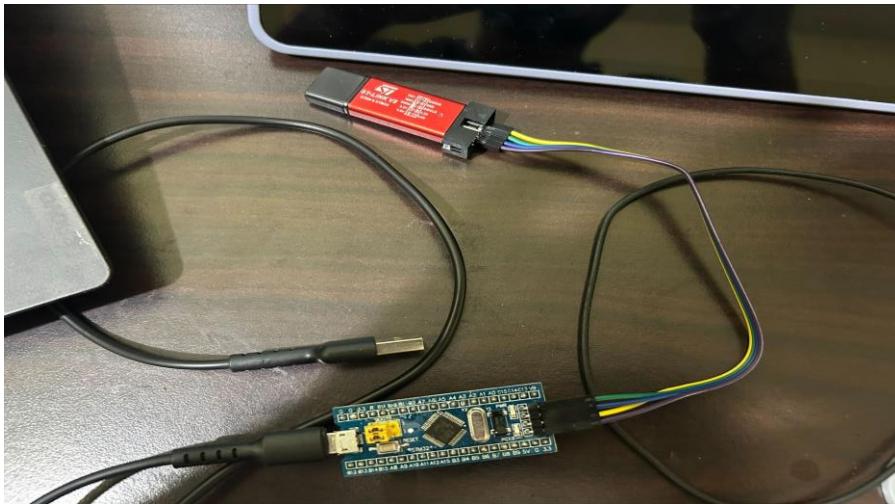
Oscillators:

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

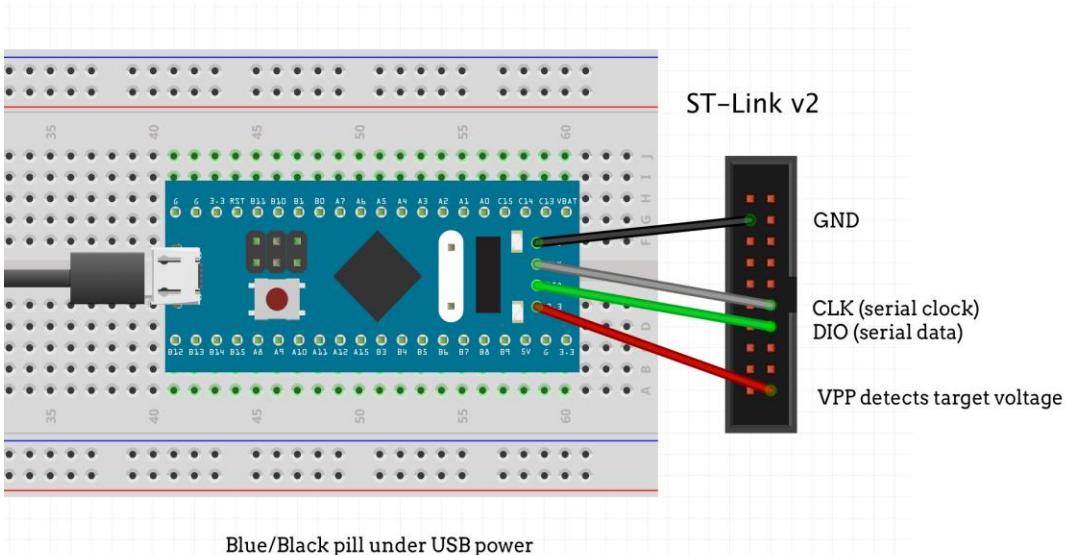
To use a ST-Link to program a blue/black pill (BP) under external power, connect it like this:

ST-Link GND (even-numbered pins from 4 to 20) to BP Ground

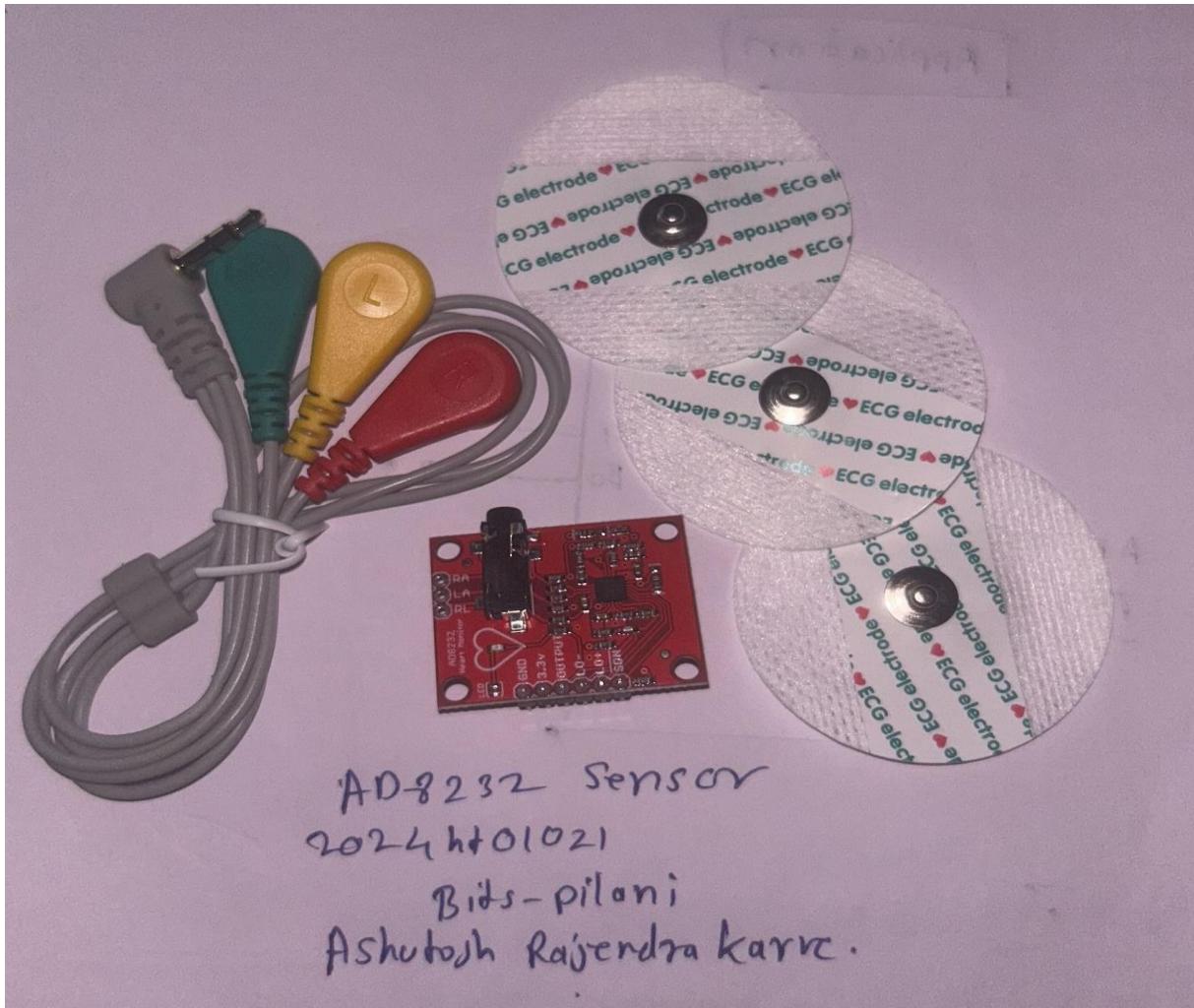
ST-Link SWDIO (pin 7) to BP DIO

ST-Link SWCLK (pin 9) to BP CLK

ST-Link VAPP (pins 1 or 2) to BP 3V3



- 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.

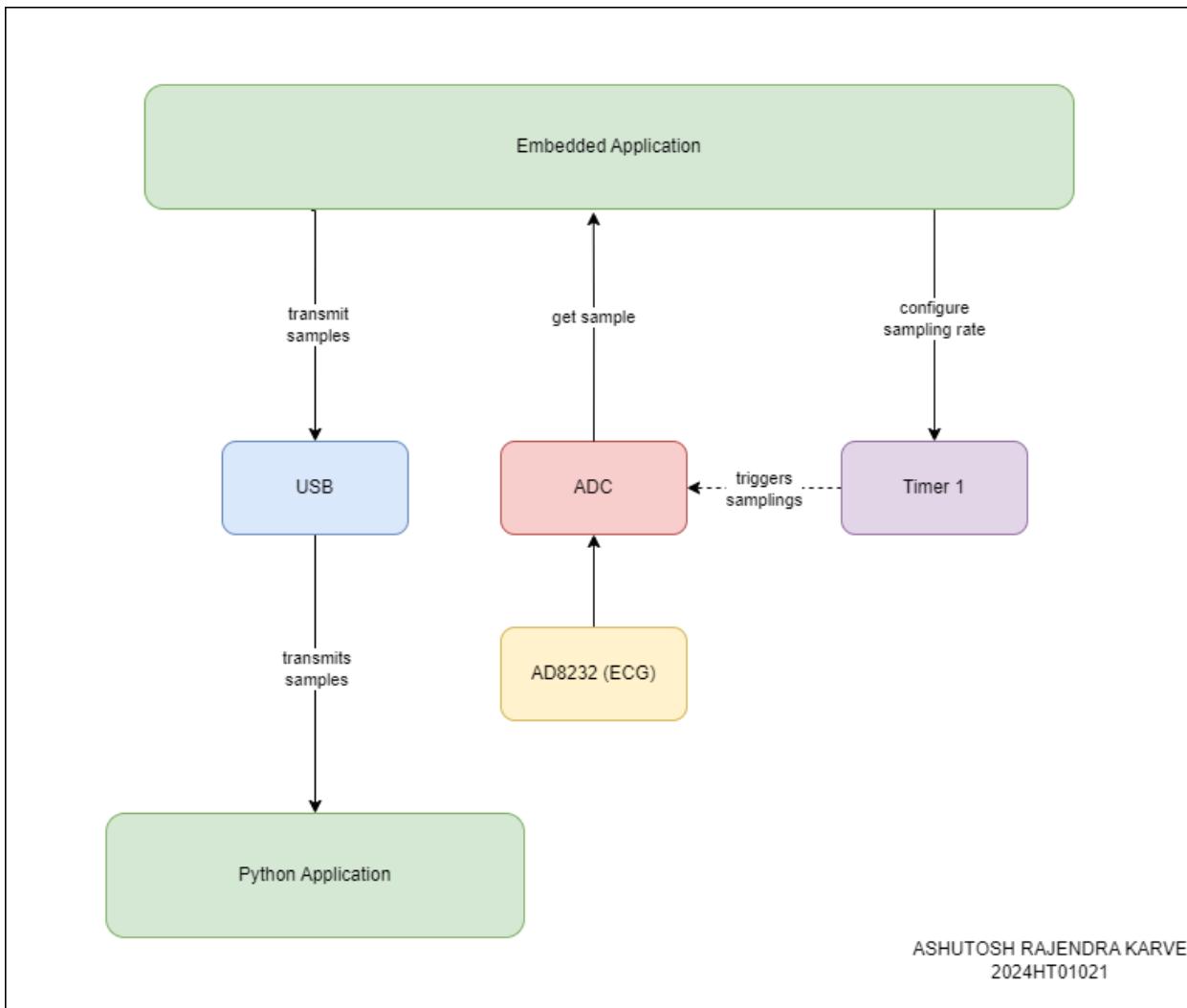
- Heartbeat Pulse Sensor (For Future scope)



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**

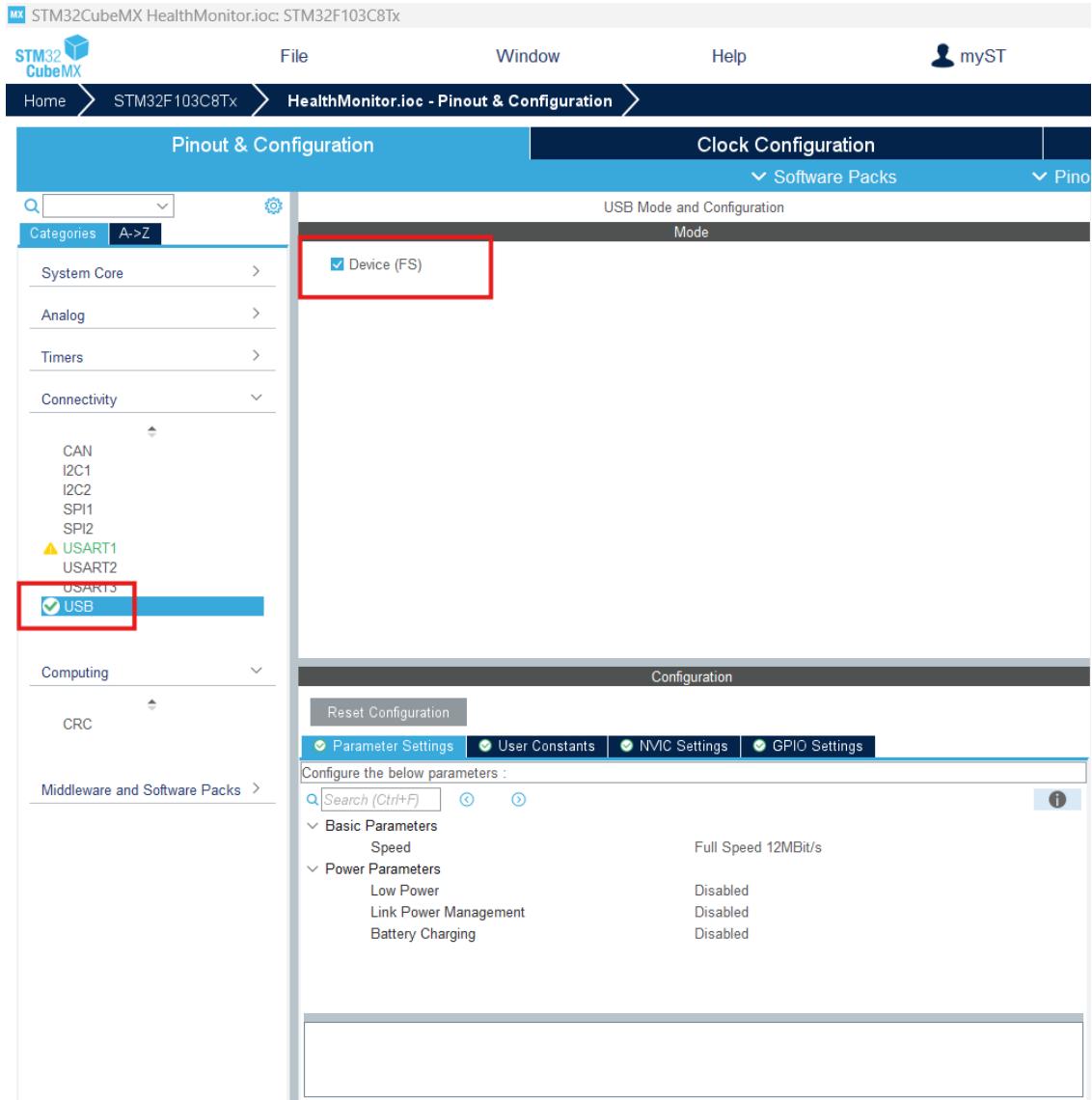


The application was developed using Keil MDK Software for stm32f103c8 microcontroller. And pin configuration was done on STM32 CUBEMX, we are collecting data from AD8232 sensor from ADC to STM32F103 and then data is transmitted to any PC/Raspberry pi.

## 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

STM32CubeMX HealthMonitor.ioc: STM32F103C8Tx

File Window Help myST

Home > STM32F103C8Tx > HealthMonitor.ioc - Pinout & Configuration

Pinout & Configuration Clock Configuration

Software Packs Pinou

Q Categories A-Z

CRC

Middleware and Software Pac... ▾

- AIROC-Wi-Fi-Bluetooth-STM...
- FATFS
- FP-SNS-MOTENVWB1
- FP-SNS-SMARTAG2
- FP-SNS-STBOX1
- FREERTOS
- I-CUBE-Cesium
- I-CUBE-FS-RTOS
- I-CUBE-ITTIADB
- I-CUBE-Mongoose
- I-CUBE-embOS
- I-CUBE-wolfMQTT
- I-CUBE-wolfSSH
- I-CUBE-wolfSSL
- I-CUBE-wolfTPM
- I-Cube-SoM-uGOAL
- USB\_DEVICE**
- X-CUBE-ALGOBUILD
- X-CUBE-ALS
- X-CUBE-BLE1
- X-CUBE-BLE2
- X-CUBE-BLEMGR
- X-CUBE-DPower
- X-CUBE-EEPRMA1
- X-CUBE-GNSS1
- X-CUBE-ISPU
- X-CUBE-MEMS1
- X-CUBE-NFC4
- X-CUBE-NFC6
- X-CUBE-NFC7
- X-CUBE-RT-Thread\_Nano
- X-CUBE-SAFEA1
- X-CUBE-SFXS2LP1
- X-CUBE-SMBUS
- X-CUBE-SUBG2
- X-CUBE-TCPP
- X-CUBE-TOF1

USB\_DEVICE Mode

Class For FS IP: Communication Device Class (Virtual Port Com)

Configuration

Reset Configuration

Parameter Settings Device Descriptor User Constants

Configure the below parameters :

Search (Ctrl+F) ⓘ

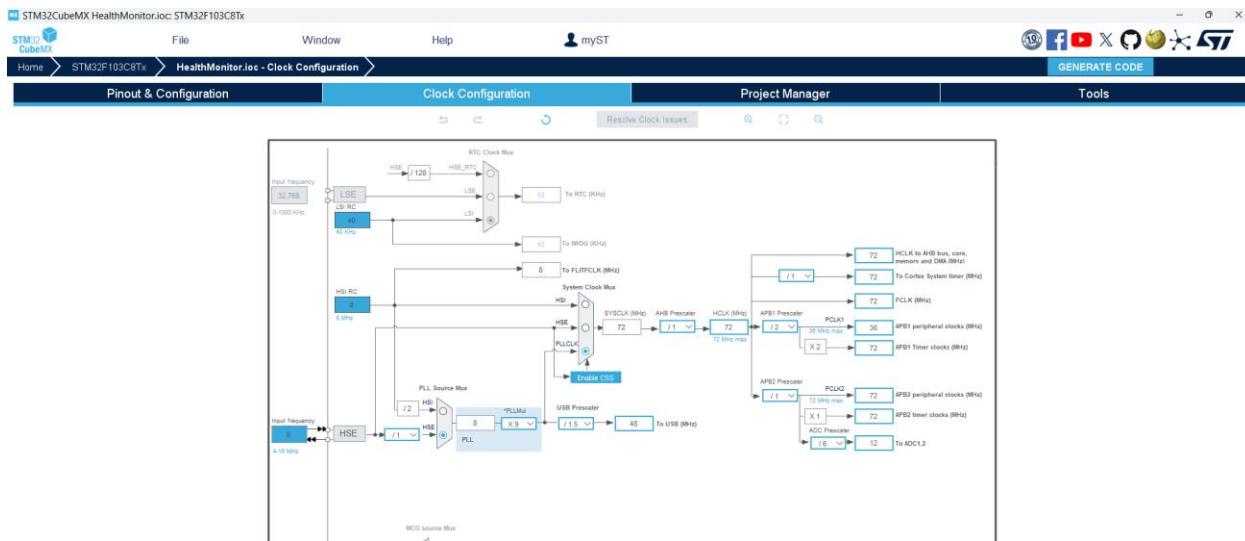
Basic Parameters

- USBD\_MAX\_NUM\_INTERFACES (Maximum number of interfaces) 1
- USBD\_MAX\_NUM\_CONFIGURATION (Maximum number of configurations) 1
- USBD\_MAX\_STR\_DESC\_SIZ (Maximum size for string descriptions) 512 bytes
- USBD\_SELF\_POWERED (Enabled self power) Enabled
- USBD\_DEBUG\_LEVEL (USBD Debug Level) 0: No debug message

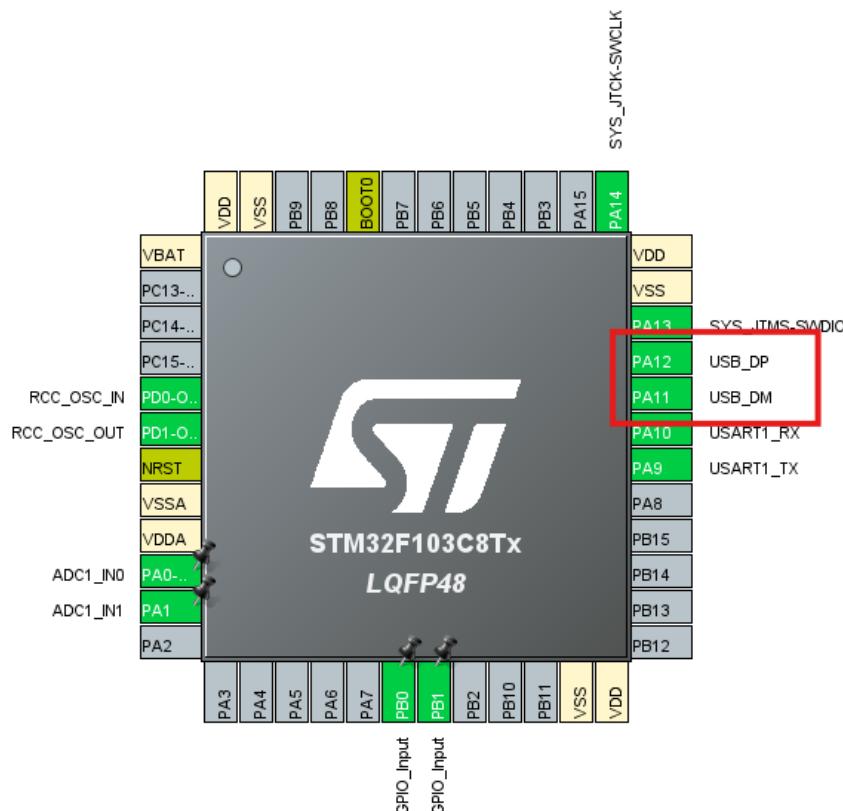
Class Parameters

USB CDC Rx Buffer Size	1024 Bytes
USB CDC Tx Buffer Size	1024 Bytes

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

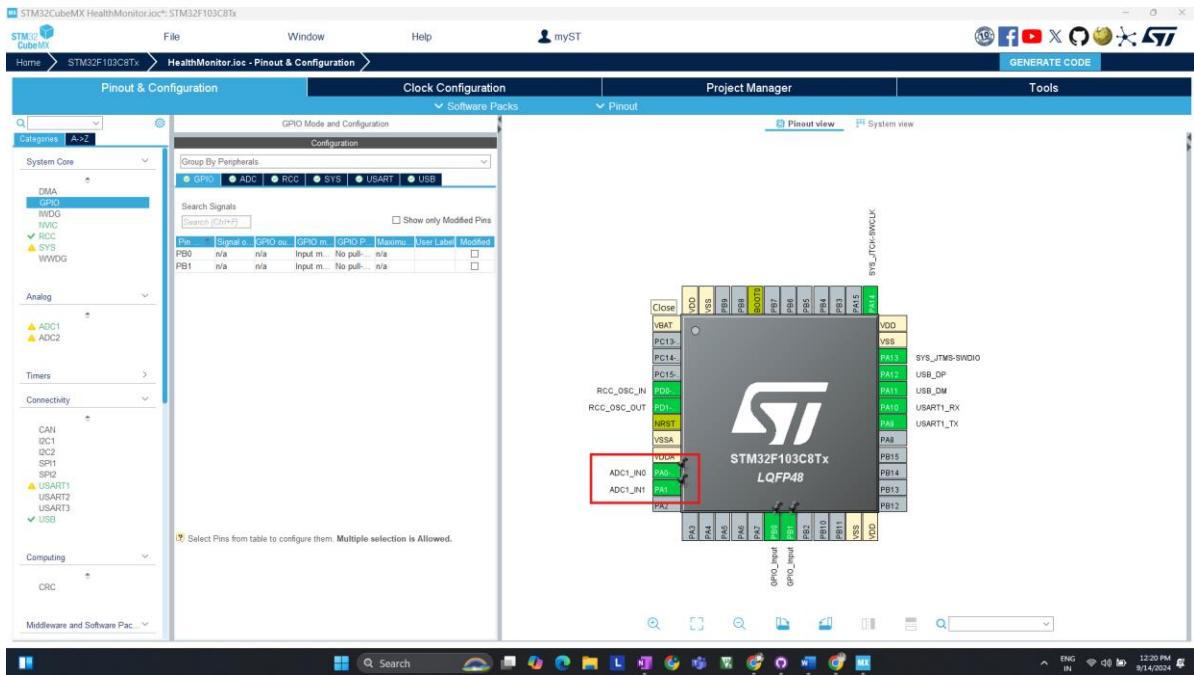


Finally, the clock is set to maximum here. As you can see above, the USB clock is automatically adjusted to 48 MHz

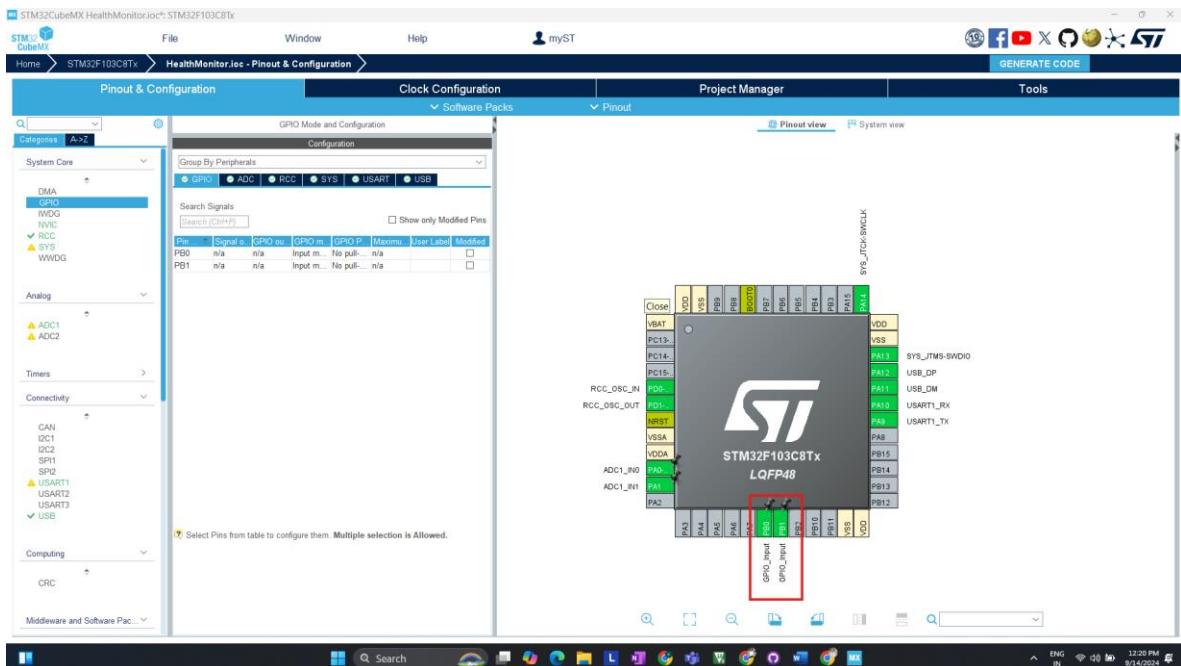


Ensure PA11 (USB\_DM) and PA12(USB\_DP) are connected for USB functionality

- 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.

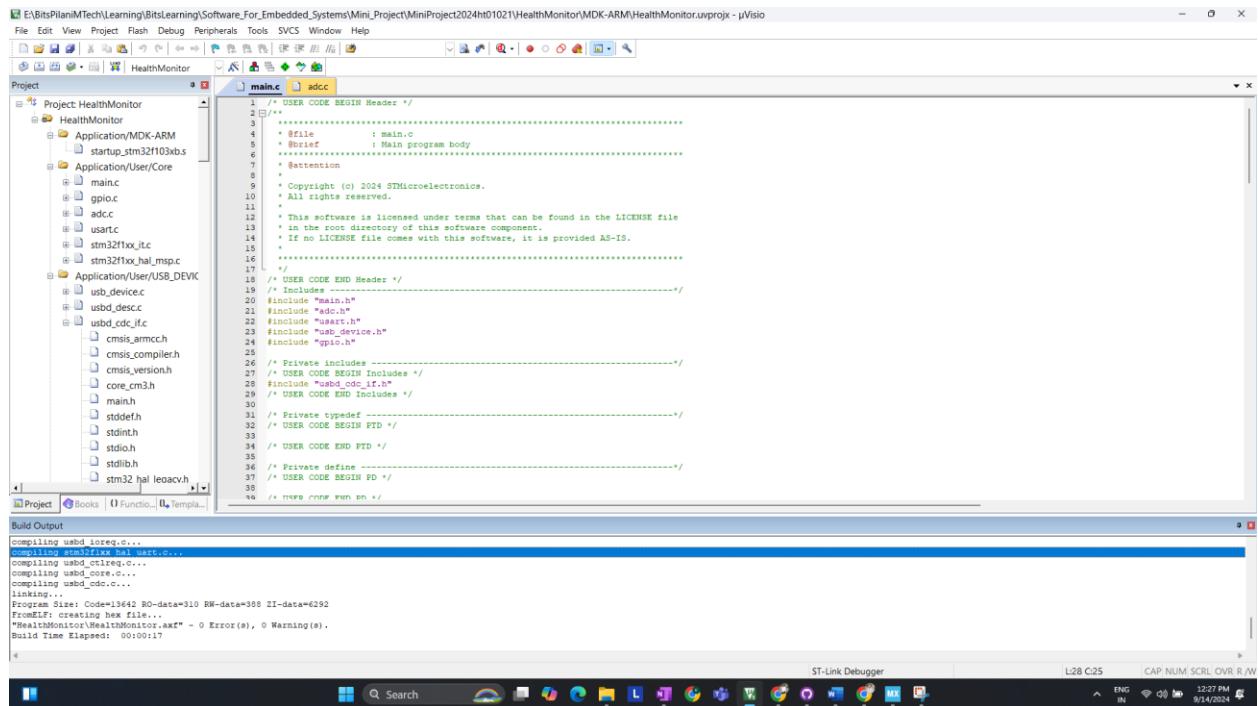


Use PB0 and PB1 on STM32 to connect to LO- and LO+ on the AD8232 for detecting if leads are properly attached.

## 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).



```
/* USER CODE BEGIN Header */
/*
 * File   : main.c
 * Brief  : Main program body
 * Attention
 *
 * Copyright (c) 2024 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 */
/* USER CODE END Header */

#include "main.h"
#include "adc.h"
#include "uart.h"
#include "service.h"
#include "gpio.h"

/* Private includes */
/* USER CODE BEGIN Includes */
#include "usb_desc.h"
#include "usb_if.h"
#include "cmsis_armc.h"
#include "cmsis_compiler.h"
#include "cmsis_version.h"
#include "core_cm3.h"
#include "main.h"
#include "stddef.h"
#include "stdin.h"
#include "stdio.h"
#include "stdlib.h"
#include "stm32f1xx_hal_msp.c"
#include "stm32f1xx_hal_leadc.h"
/* USER CODE END Includes */

/* Private typedef */
/* USER CODE BEGIN FD */
/* USER CODE END FD */

/* Private define */
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro */
/* USER CODE BEGIN MN */
/* USER CODE END MN */

```

Build Output

```
Compiling usbd_inreq.c...
Compiling stm32f1xx_hal_uart.c...
Compiling usbd_ctreq.c...
Compiling usbd_core.c...
Compiling usbd_desc.c...
linking...
Program Size: Code=19638880 Data=310 RW-Data=308 ZI-Data=6292
Free: 10448K
Creating hex file...
"HealthMonitor\HealthMonitor.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:17
```

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.

The screenshot shows the uVisio IDE interface with the following details:

- Project:** HealthMonitor
- Toolchain:** Application/MDK-ARM
- Source Files:** main.c, adc.c, gpio.c, usart.c, stm32f10x\_it.c, stm32f10x\_hal\_msp.c, usbd\_device.c, usbd\_desc.c, usbd\_cdc\_ifc.c, cmsis\_armcc.h, cmsis\_compiler.h, cmsis\_version.h, core\_cm3.h, main.h, stddef.h, stdint.h, stdio.h, stdlib.h, stm32f103x.h, stm32f103x\_it.h, and stm32f103x\_hal.h.
- Code Editor:** The main.c file is open, showing C code. A red box highlights the following code block:

```
105     /* User code mail1 = While from STM32 for USB Testing setup */
106     CDC_Transmit_FS((uint8_t*)msg, strlen(msg));
107     HAL_Delay(1000);
108 }
```

- Build Output:**

```
compiling usbd_cdc_ifc.c...
compiling usbd_desc.c...
compiling stm32f103x_hal_uart.c...
linking HealthMonitor

Program Size: Code=14020 RO=0 Data=312 RW=0 Data=388 ZI=0 Data=6292
FromELF: creating hex file...
"HealthMonitor\HealthMonitor.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:14
Load "HealthMonitor\HealthMonitor.axf"
Erase Done.
```

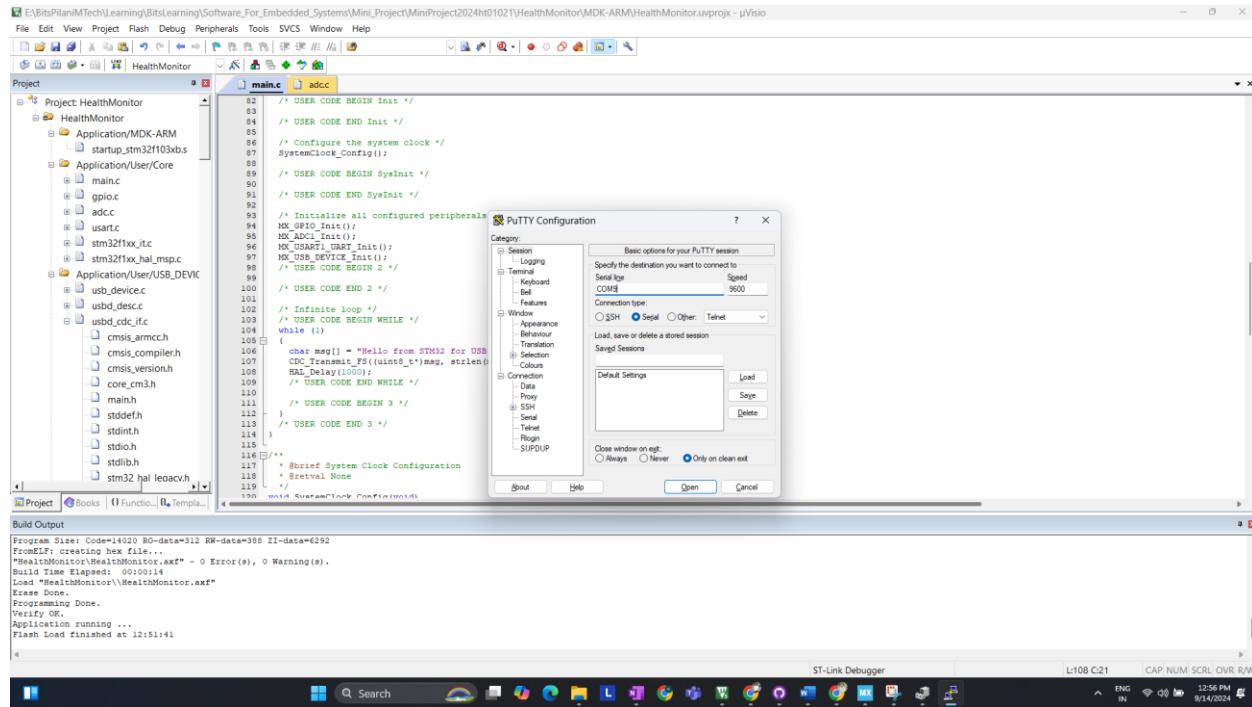
In above screenshot I have compiled and flash code for USB CDC.

The screenshot shows a Windows desktop with a software development interface. The title bar reads "E:\BSP\PlanIt\Tech\Learning\BSP\Learning\Software\_For\_EMBEDDED\_Systems\Mini\_Project\MiniProject2024\hi10121\HealthMonitor(MDK-ARM)\HealthMonitor.uvproj - uVisio". The interface includes a top menu bar with File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, Help, and a Device Manager tab. A left sidebar titled "Project" lists various device components like Adustooth, Batteries, Bluetooth, Cameras, Computer, Disk drives, Display adapters, Firmware, Human Interface Devices, IDE ATA/ATAPI controllers, Keyboards, Mice and other pointing devices, Monitors, Network adapters, Ports (COM & LPT), Print queues, Processors, Security devices, and Software components. A central workspace shows code snippets for stddef.h, stdint.h, stdio.h, stdlib.h, and stm32\_hal\_leaccv.h, along with assembly-like code for a user application. At the bottom, a "Build Output" window displays the build process, and a status bar at the bottom right shows "L108 C21" and connectivity icons.

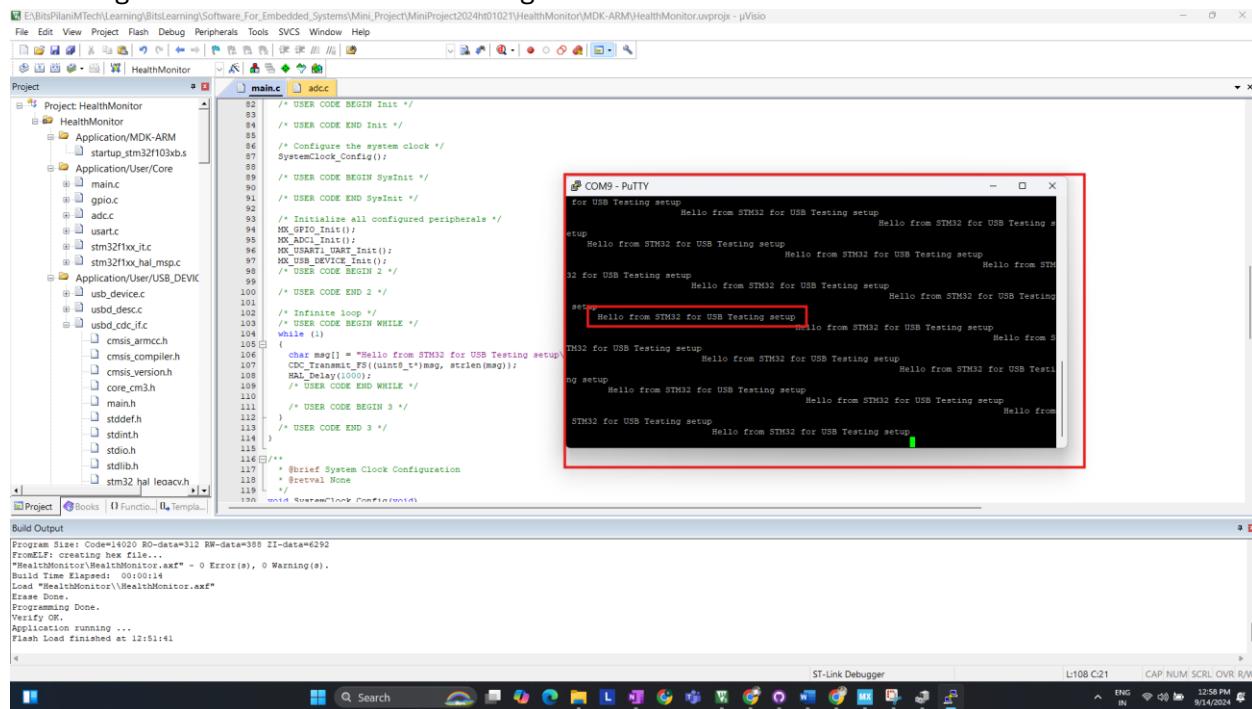
We can detect COM Port (COM9) for Our device (STM32 Blue Pill)

## 2. Checking with putty:

Configuring Serial to COM9 and setting baud rate to 9600

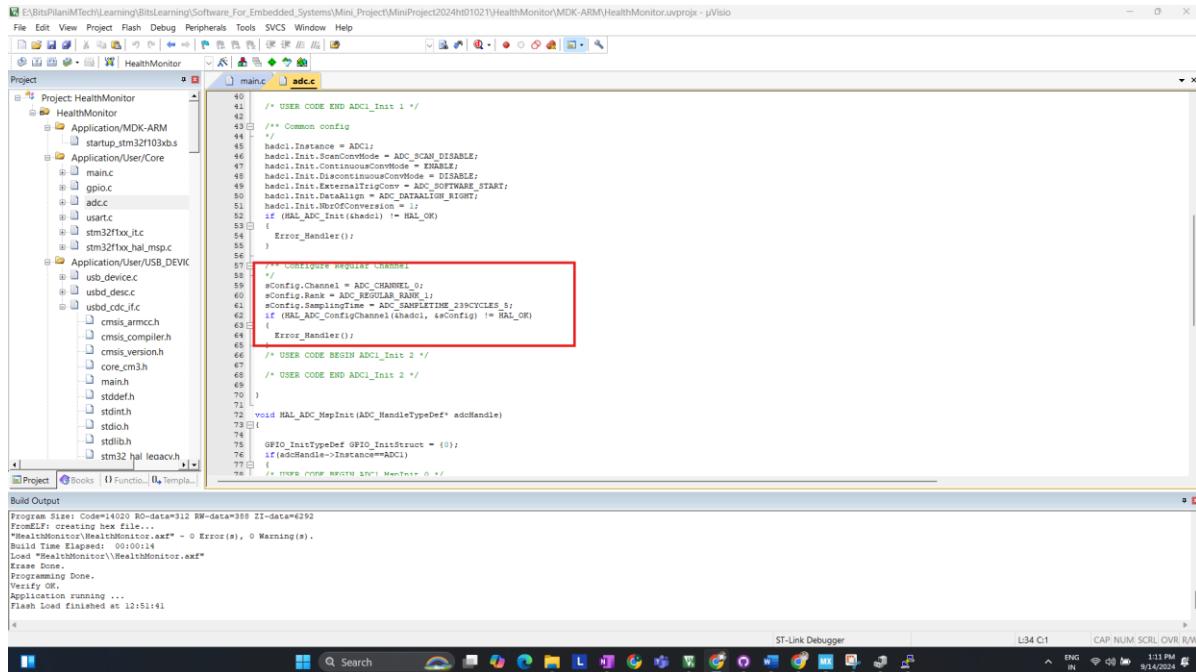


Checking Serial Terminal We are receiving the DATA



### 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

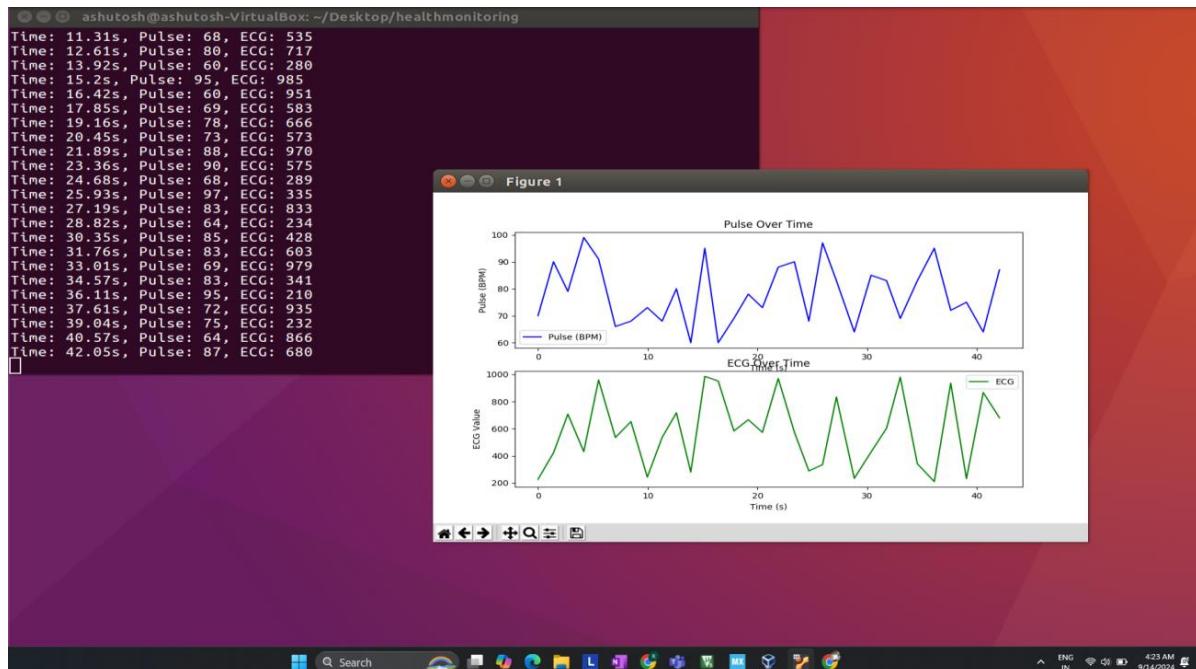


```

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
File Explorer Project HealthMonitor main.c adc.c
Project: HealthMonitor
  Application/MDK-ARM
    startup_stm32f103xs.s
  Application/User/Core
    main.c
    gpio.c
    adcc.c
    usart.c
    stm32f1xx_it.c
    stm32f1xx_hal_msp.c
  Application/User/USB_DEVIC
    usb_device.c
    usb_desc.c
    usbdc_dcifc.c
      cmsis_armcc.h
      cmsis_complier.h
      cmsis_version.h
      core_cm3.h
      main.h
      stddef.h
      stdint.h
      stdio.h
      stdlib.h
      stm32_hal_leoach.v
  Project Books Functions Temp...
Build Output
Program Size: Code=14020 RO=0 Data=312 SW=Data=388 ZI=Data=6292
FromElf: creating hex file...
"HealthMonitor\HealthMonitor.axf" - 0 Error(s), 0 Warning(s).
Build Time: Elapsed: 00:00:14
Load "HealthMonitor\HealthMonitor.axf"
Erase Done.
Programming Done.
Verify OK.
Application running ...
Flash Load finished at 12:15:14
4
ST-Link Debugger L34 C1 CAP NUM SCRLL OVR R/W
ENG IN 111 PM 9/14/2024

```

### 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.

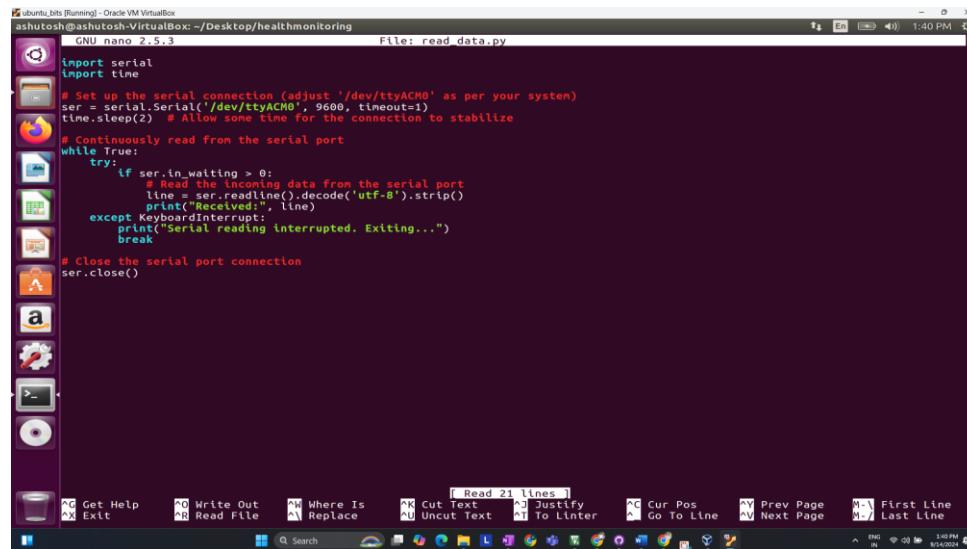
#### 1: Setting Up the Python Environment

##### 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)



A screenshot of a terminal window titled "ubuntu\_bits [Running] - Oracle VM VirtualBox". The window shows a Python script named "read\_data.py" being run in a terminal. The script uses the "serial" library to open a serial port at "/dev/ttyACM0" (adjustable) with a baud rate of 9600 and a timeout of 1 second. It then enters a loop where it reads data from the serial port every second, prints it to the console, and handles a KeyboardInterrupt exception by printing a message and breaking out of the loop. Finally, it closes the serial port connection.

```
ashutosh@ashutosh-VirtualBox:~/Desktop/healthmonitoring$ File: read_data.py
GNU nano 2.5.3
import serial
import time

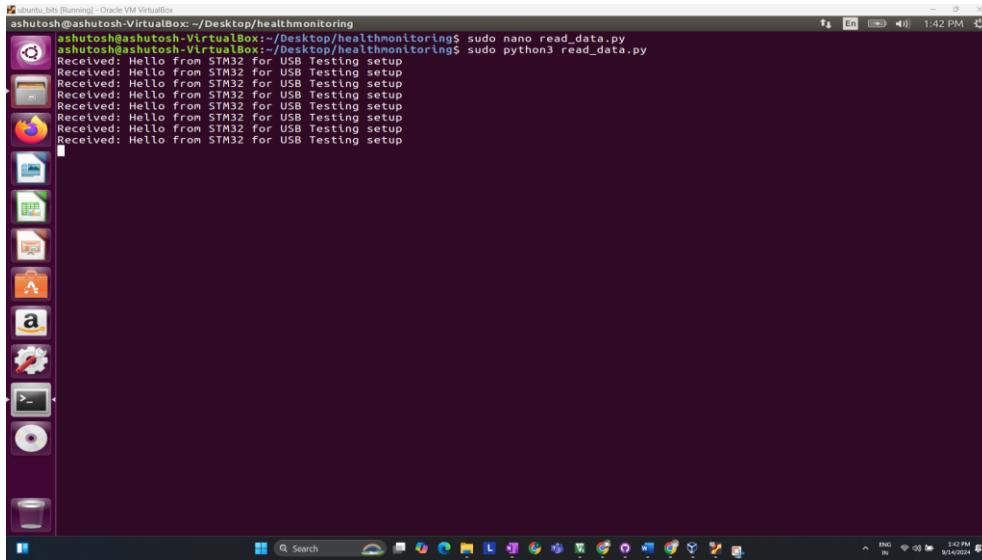
# Set up the serial connection (adjust '/dev/ttyACM0' as per your system)
ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
time.sleep(2) # Allow some time for the connection to stabilize

# Continuously read from the serial port
while True:
    try:
        if ser.in_waiting > 0:
            # Read the incoming data from the serial port
            line = ser.readline().decode('utf-8').strip()
            print("Received: ", line)
        else:
            print("Serial reading interrupted. Exiting...")
            break
    except KeyboardInterrupt:
        print("Serial reading interrupted. Exiting...")
        break

# Close the serial port connection
ser.close()
```

##### Explanation:

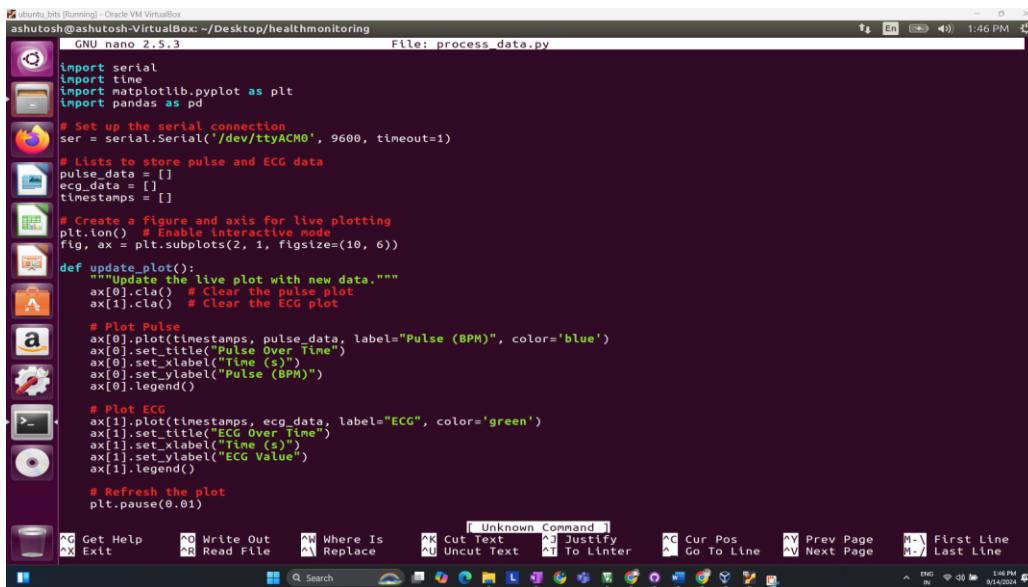
- The `serial.Serial()` function opens the USB serial port.
- The script reads incoming data from the STM32, which is sent every second, and prints it to the console.



```
ashutosh@ashutosh-VirtualBox:~/Desktop/healthmonitoring$ sudo nano read_data.py
ashutosh@ashutosh-VirtualBox:~/Desktop/healthmonitoring$ sudo python3 read_data.py
Received: Hello from STM32 for USB Testing setup
```

### 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.



```
ashutosh@ashutosh-VirtualBox:~/Desktop/healthmonitoring$ File: process_data.py
GNU nano 2.5.3
import serial
import time
import matplotlib.pyplot as plt
import pandas as pd

# Set up the serial connection
ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1)

# Lists to store pulse and ECG data
pulse_data = []
ecg_data = []
timestamps = []

# Create a figure and axis for live plotting
plt.ion() # Enable interactive mode
fig, ax = plt.subplots(2, 1, figsize=(10, 6))

def update_plot():
    """Update the live plot with new data."""
    ax[0].cla() # Clear the pulse plot
    ax[1].cla() # Clear the ECG plot

    # Plot Pulse
    ax[0].plot(timestamps, pulse_data, label="Pulse (BPM)", color='blue')
    ax[0].set_title("Pulse Over Time")
    ax[0].set_xlabel("Time (s)")
    ax[0].set_ylabel("Pulse (BPM)")
    ax[0].legend()

    # Plot ECG
    ax[1].plot(timestamps, ecg_data, label="ECG", color='green')
    ax[1].set_title("ECG Over Time")
    ax[1].set_xlabel("Time (s)")
    ax[1].set_ylabel("ECG Value")
    ax[1].legend()

    # Refresh the plot
    plt.pause(0.01)

^G Get Help      ^W Write Out     ^M Where Is      ^X Cut Text      ^J Justify      ^T To Linter     ^C Cur Pos      ^Y Prev Page    ^K First Line
^X Exit        ^R Read File     ^N Replace      ^U Uncut Text    ^I To Invert    ^G Go To Line    ^V Next Page    ^L Last Line
^Q Unknown Command
```

```

ashutosh@ashutosh-VirtualBox: ~/Desktop/healthmonitoring
GNU nano 2.5.3           File: process_data.py

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()

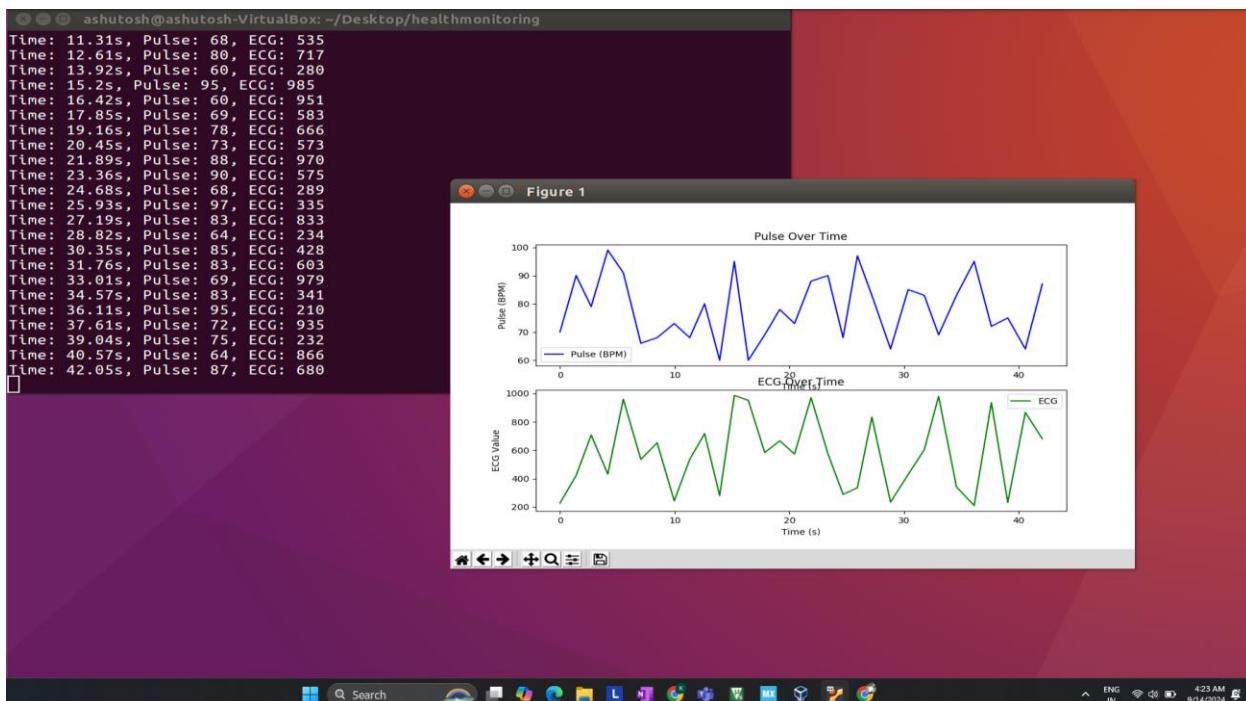
            # Print the received data for logging
            print("Time: {}, Pulse: {}, ECG: {}".format(elapsed_time, pulse, ecg))

            # 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:

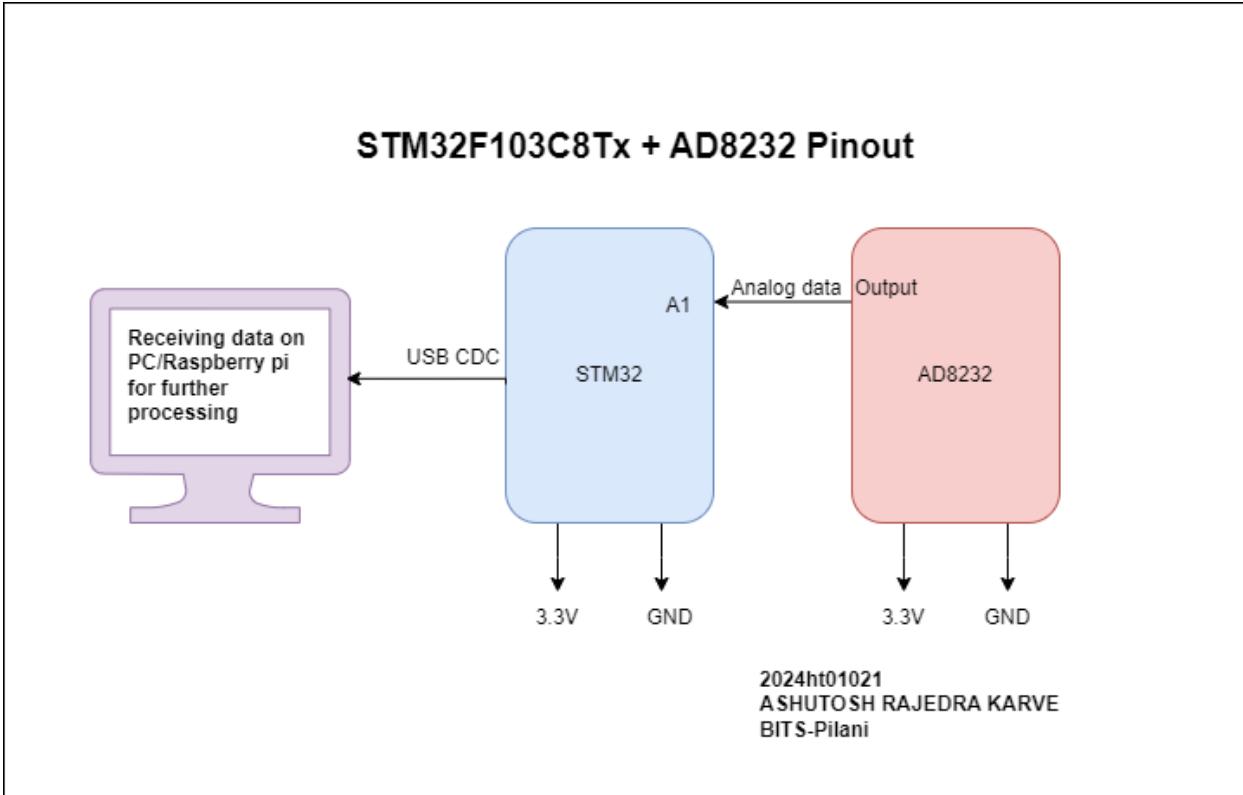


Real-time ECG Plot Testing...

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//

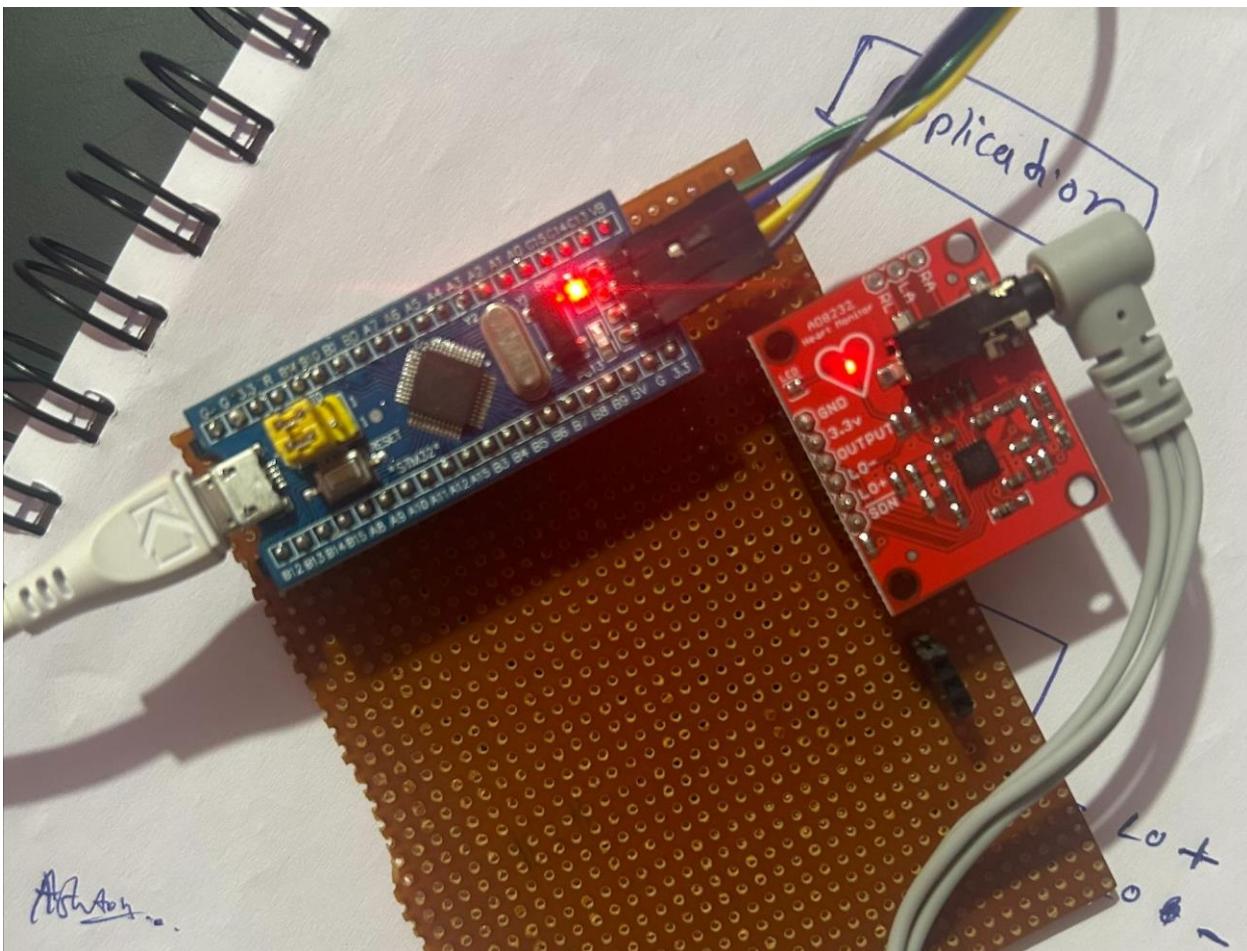
The screenshot shows the µVision IDE interface with the following details:

- Project Explorer:** Shows the project structure under "Project: HealthMonitor".
  - HealthMonitor:** Contains files like startup\_stm32f103xb.s, main.c, gpio.c, usart.c, adc.c, and stm32f1xx\_itc.h.
  - Application/MDK-ARM:** Contains startup\_stm32f103xb.s.
  - Application/User/Core:** Contains main.c, gpio.c, usart.c, adc.c, and stm32f1xx\_itc.h.
  - Application/User/USB\_DEVK:** Contains usb\_device.c, usbd\_desc.c, usbd\_cdc\_if.c, and usbd\_cdc\_if.h, which includes cmsis\_armmc.h, cmsis\_compiler.h, cmsis\_version.h, core\_cm3.h, main.h, stddef.h, stdint.h, stdio.h, stdlib.h, and stm32f1xx\_hal.h.
- Code Editor:** Displays the main.c file with the following code:

```
95 HAL_ADC_Init();
96 HAL_I2C_Init();
97 HAL_USB_DEVICE_Init();
98 /* USER CODE BEGIN 2 */
99
100 /* USER CODE END 2 */
101
102 /* Infinite loop */
103 /* USER CODE BEGIN WHILE */
104 while (1)
105 {
106     /* Infinite loop */
107     /* USER CODE BEGIN WHILE */
108     while (1)
109     {
110         // Start ADC conversion
111         HAL_ADC_Start(hadcl);
112
113         // Poll for ADC conversion completion
114         if (HAL_ADC_PollForConversion(hadcl), HAL_MAX_DELAY) == HAL_OK
115         {
116             // Get the ADC value from PA1 (A1)
117             uint32_t AdcValue = HAL_ADC_GetValue(hadcl);
118
119             // Convert the ADC value to a string
120             char buffer[10];
121             sprintf(buffer, "%lu\n", AdcValue); // Send raw value
122
123             // Transmit the data over USB
124             CDC_Transmit_FS((uint8_t*)buffer, strlen(buffer));
125         }
126
127         // Stop the ADC
128         HAL_ADC_Stop(hadcl);
129
130         // Delay adjusted for higher sampling rate (2ms for 500Hz sampling rate)
131         HAL_Delay(2);
132
133     } // more code run until ...
134 }
```
- Build Output:** Shows the build process:

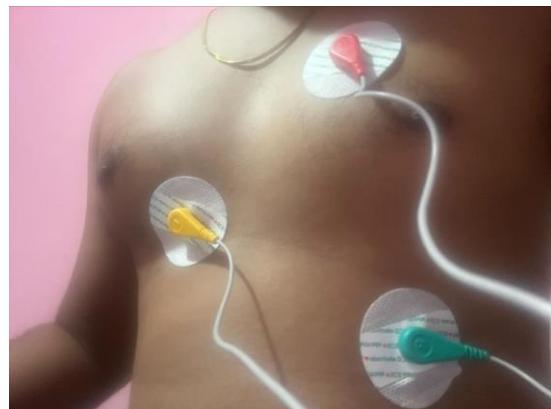
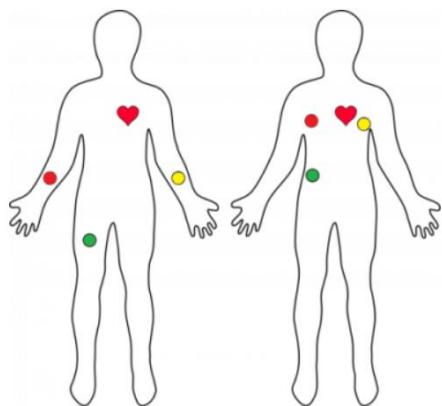
```
Project Size: Code=151668 Ro-data=328 RW-data=388 ZI-data=6292
File2Elf: creating hex file...
*HealthMonitor\HealthMonitor.axf - 0 Errors(s), 0 Warning(s).
Load Time Elapsed: 00:00:14
Build "HealthMonitor\\HealthMonitor.axf"
Elapsed: 00:00:14
Programming Done.
Verify OK.
Application running ...
Flash Load finished at 23:25:03
```
- Bottom Bar:** Includes the ST-Link Debugger status, system information (L132 C2, CAP NUM SCRLL OVR R/W), and system icons.

## 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**.

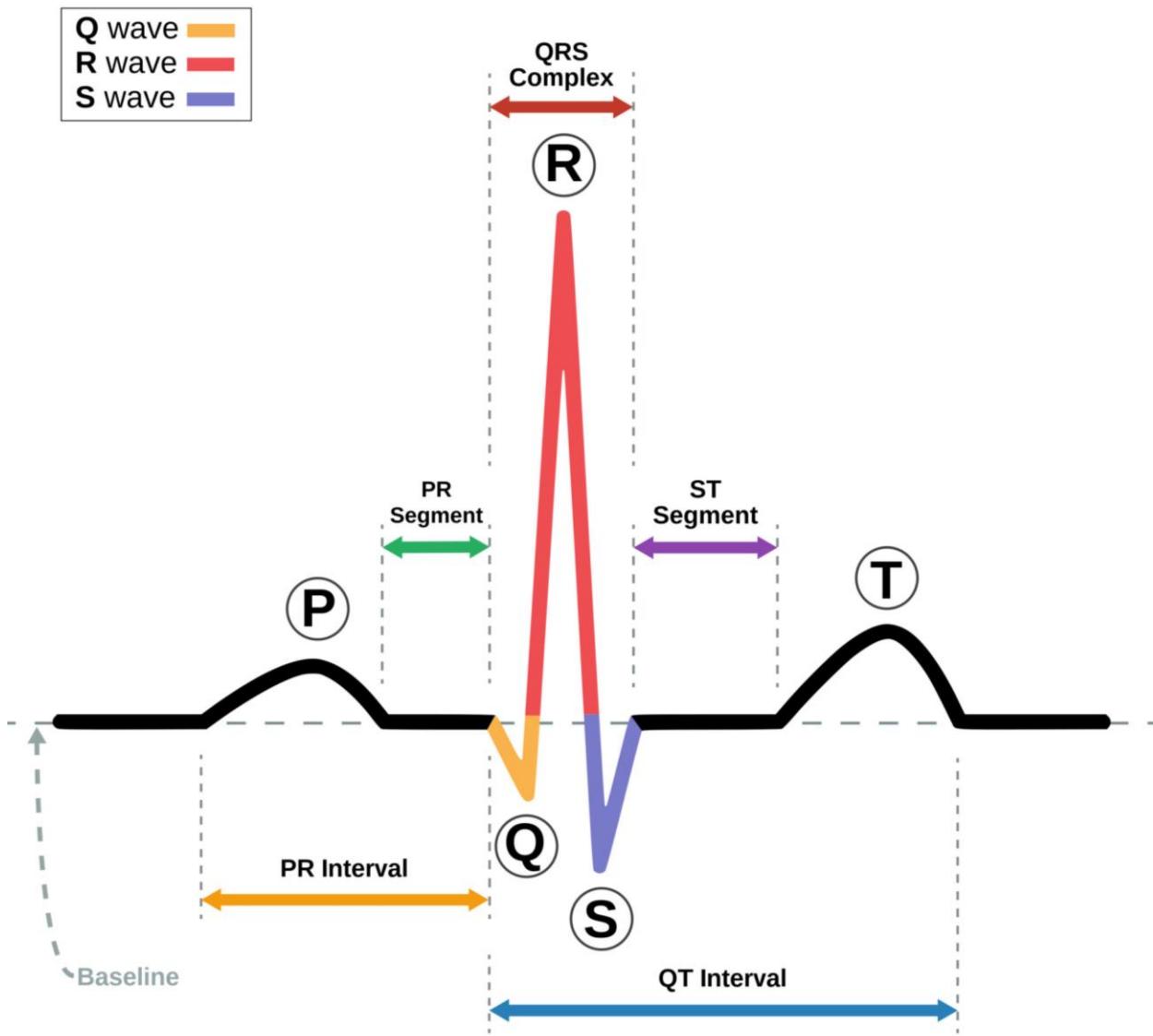
It represents the time between two **QRS complexes**.

- **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



## Raw data which I am receiving on putty

The screenshot shows a software environment for developing and debugging a project named 'HealthMonitor'. The top menu bar includes 'View', 'Project', 'Flash', 'Debug', 'Peripherals', 'Tools', 'SVCS', 'Window', and 'Help'. Below the menu is a toolbar with various icons. A file browser window is open, showing files like 'main.c' and 'adc.c' under the 'HealthMonitor' project. In the foreground, a terminal window titled 'COM9 - PuTTY' is displaying raw data. The data consists of a series of numerical values: 440, 393, 395, 448, 493, 518, 474, 420, 391, 420, 466, 515, 497, 458, 415, 385, 437, 491, 310, 491, 4, followed by a closing brace '}' and the text 'DELAY)'. At the bottom of the terminal window, there is some code: ' // transmit the data over USB', ' CDC\_Transmit\_FS((uint8\_t\*)buffer, strlen(buffer));', and ' // end raw v'. The file browser also shows other header files: 'cmsis\_version.h', 'core\_cm3.h', 'main.h', and 'stddef.h'.

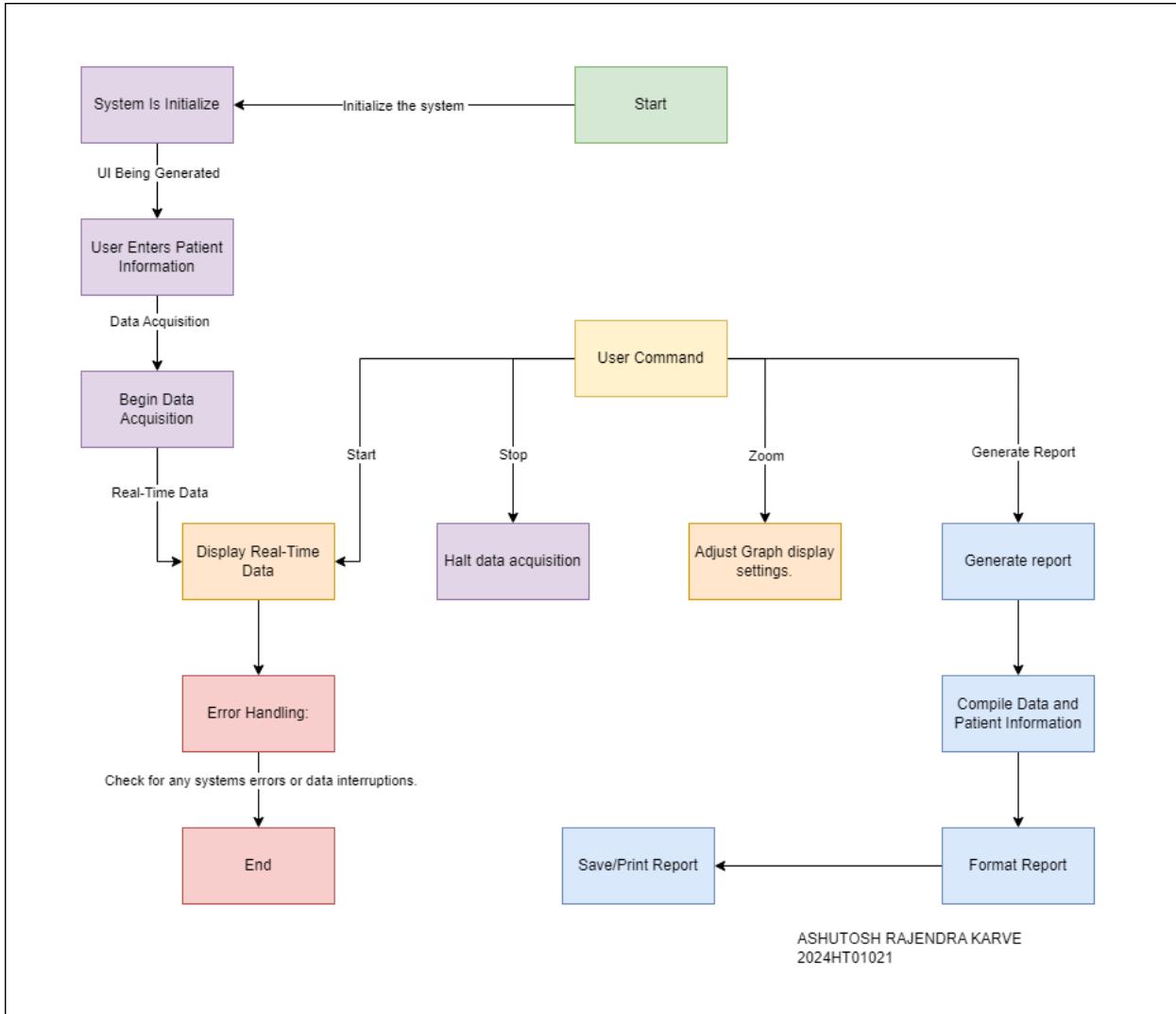
## Observations from Putty Display of ECG Data

### 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.

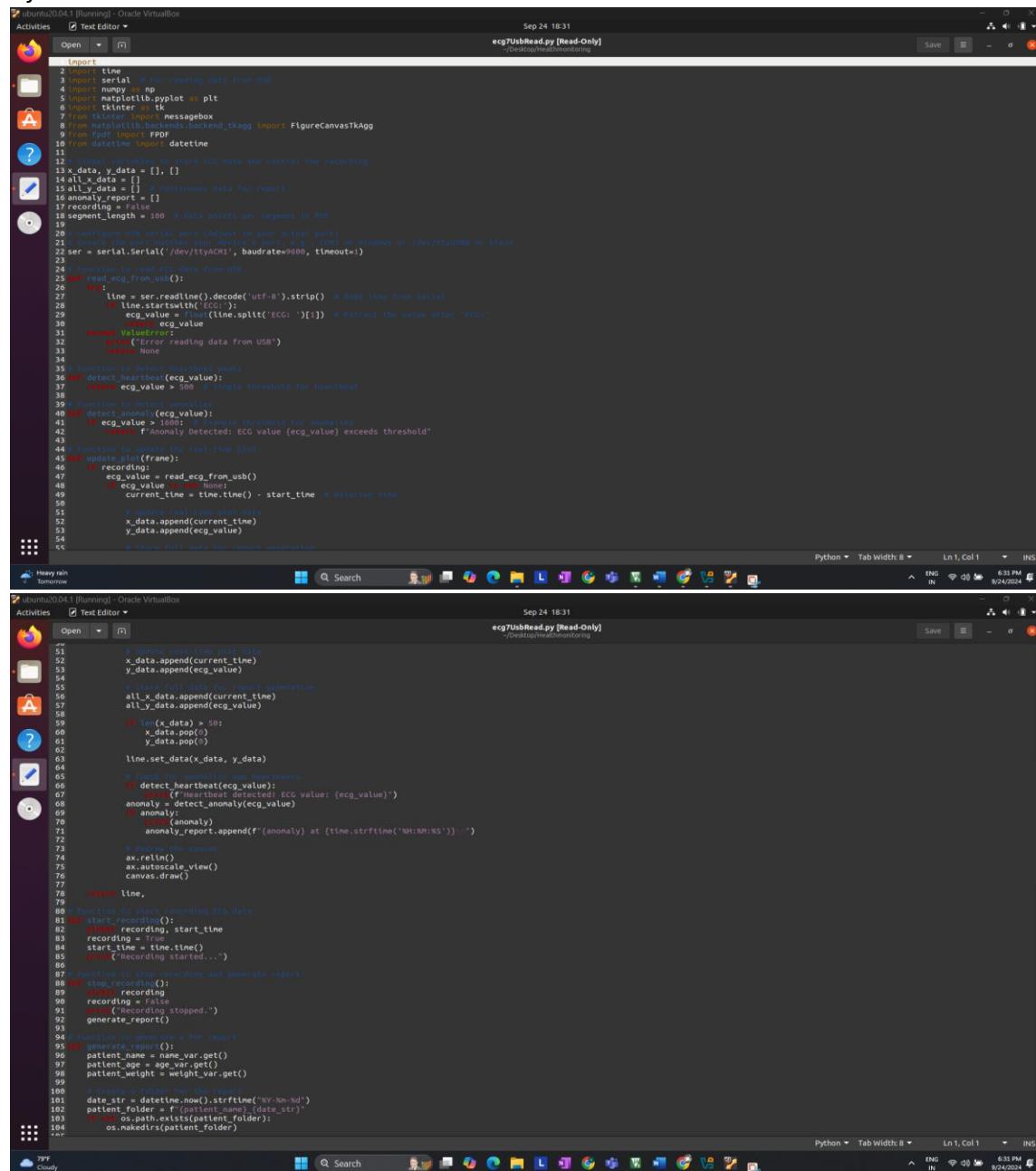
**Report Generation:** Compiles data and patient information, formats it, and allows for saving or printing.

## 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>>



```
#!/usr/bin/python3
# Import required modules
import serial # For reading data from USB
import numpy as np
import matplotlib.pyplot as plt
from tkinter import Tk
from tkinter import messagebox
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from fpdf import FPDF
from datetime import datetime
# Global variables
x_data, y_data = [], []
all_x_data = []
all_y_data = []
continuous_data = False
normal_ecg_value = 500
recording = False
segment_length = 100 # Data points per segment in PDF
# Configure your serial port (adjust to your actual port)
# Ensure the port matches your device's port, e.g., COM3 on Windows or /dev/ttyUSB0 on Linux
ser = serial.Serial('/dev/ttyACM1', baudrate=9600, timeout=1)

# Function to read ECG data from USB
def read_ecg_from_usb():
    try:
        line = ser.readline().decode('utf-8').strip() # Read line from serial
        if line.startswith('ECG:'):
            ecg_value = float(line.split('ECG:')[1]) # Extract the value after 'ECG:'
            ecg_value
    except ValueError:
        print("Error reading data from USB")
        return None
    return ecg_value

# Function to detect heartbeat peaks
def detect_heartbeat(ecg_value):
    if ecg_value > 500:
        return True
    else:
        return False

# Function to detect anomalies
def detect_anomaly(ecg_value):
    if ecg_value > 1000: # Example threshold for anomalies
        return "Anomaly Detected: ECG value (ecg_value) exceeds threshold"
    else:
        return "Normal ECG"

# Function to update the real-time plot
def update_plot(frame):
    if recording:
        current_time = time.time()
        if ecg_value is not None:
            current_time = time.time() - start_time # Relative time
            x_data.append(current_time)
            y_data.append(ecg_value)
        line.set_data(x_data, y_data)
        frame.canvas.draw()

# Main loop
while True:
    ecg_value = read_ecg_from_usb()
    if ecg_value is not None:
        current_time = time.time() - start_time # Relative time
        x_data.append(current_time)
        y_data.append(ecg_value)
        line.set_data(x_data, y_data)
    if detect_heartbeat(ecg_value):
        if continuous_data:
            all_x_data.append(current_time)
            all_y_data.append(ecg_value)
        else:
            x_data.pop()
            y_data.pop()
            line.set_data(x_data, y_data)
    if detect_anomaly(ecg_value):
        anomaly = detect_anomaly(ecg_value)
        if anomaly:
            anomaly_report.append(f"(anomaly) at {time.strftime('%H:%M:%S')}")
    if recording:
        ax.relim()
        ax.autoscale_view()
        canvas.draw()
    if len(y_data) == segment_length:
        recording = False
        start_recording()
        recording, start_time = True, time.time()
        start_time = time.time()
        print("Recording started...")
    if recording:
        start_recording()
        recording, start_time = True, time.time()
        start_time = time.time()
        print("Recording stopped...")
        generate_report()
    if recording:
        generate_report()
        patient_name = name_var.get()
        patient_age = age_var.get()
        patient_weight = weight_var.get()
        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)
    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)
```

ubuntu20.04.1 [running] - Oracle VM VirtualBox

Activities Text Editor

Save

Sep 24 18:31

ecg7UsbRead.py [Read-Only]  
-/Desktop/HealthMonitoring

```
400
401     date_str = datetime.now().strftime("%Y-%m-%d")
402     patient_folder = f'{patient_name}_{date_str}'
403     if not os.path.exists(patient_folder):
404         os.makedirs(patient_folder)
405
406     # Filenames for PDF (HTML)
407     report_filename = os.path.join(patient_folder, f'{patient_name}_ECG_Report.pdf')
408
409     # Create PDF
410     pdf = FPDF()
411     pdf.add_page()
412
413     # Add sections info to the PDF
414     pdf.set_font("Arial", size=12)
415     pdf.cell(200, 10, txt="ECG Anomaly Detection Report", ln=True, align="C")
416     pdf.cell(200, 10, txt=f"Patient Name: {patient_name}", ln=True, align="L")
417     pdf.cell(200, 10, txt=f"Patient Age: {patient_age} years", ln=True, align="L")
418     pdf.cell(200, 10, txt=f"Patient Weight: {patient_weight}", ln=True, align="L")
419     pdf.cell(200, 10, txt=f"Date: [{time.strftime('%Y-%m-%d')}]", ln=True, align="L")
420     pdf.cell(200, 10, txt=f"Total anomalies detected: {len(anomaly_report)}", ln=True, align="L")
421
422     # Write the anomaly segments and add each segment to the PDF
423     total_points = len(all_x_data)
424     num_segments = total_points // segment_length
425
426     y_offset = 80 # Offset for the y-position of each segment in the PDF
427
428     for l in range(num_segments + 1):
429         start_idx = l * segment_length
430         end_idx = min(start_idx + segment_length, total_points)
431
432         # Start idx < end_idx:
433
434         fig_segment, ax_segment = plt.subplots(figsize=(8, 2))
435         ax_segment.plot(all_x_data[start_idx:end_idx], all_y_data[start_idx:end_idx], color='blue', lw=1)
436         ax_segment.set_title(f'ECG Waveform Segment {l + 1}')
437         ax_segment.set_xlabel('Time (seconds)')
438         ax_segment.set_ylabel('ECG Value')
439         ax_segment.grid(True)
440
441         # Save the segment as a PNG
442         segment_filename = os.path.join(patient_folder, f'ecg_segment_{l + 1}.png')
443         fig_segment.savefig(segment_filename, bbox_inches='tight')
444
445         # Add the image to the PDF
446         pdf.image(segment_filename, x=10, y=y_offset, w=180)
447         y_offset += 65 # Move the y-position down for the next segment
448
449         # Add a new page if the y_offset exceeds page height
450         if y_offset > 270:
451             pdf.add_page()
452             y_offset = 65
453
454     # Save the anomalies to the PDF
```

ubuntu20.04.1 [Running] - Oracle VirtualBox

Activities Text Editor

Open Save

eg7UsbRead.py [Read-Only]  
-/Desktop/Healthmonitoring

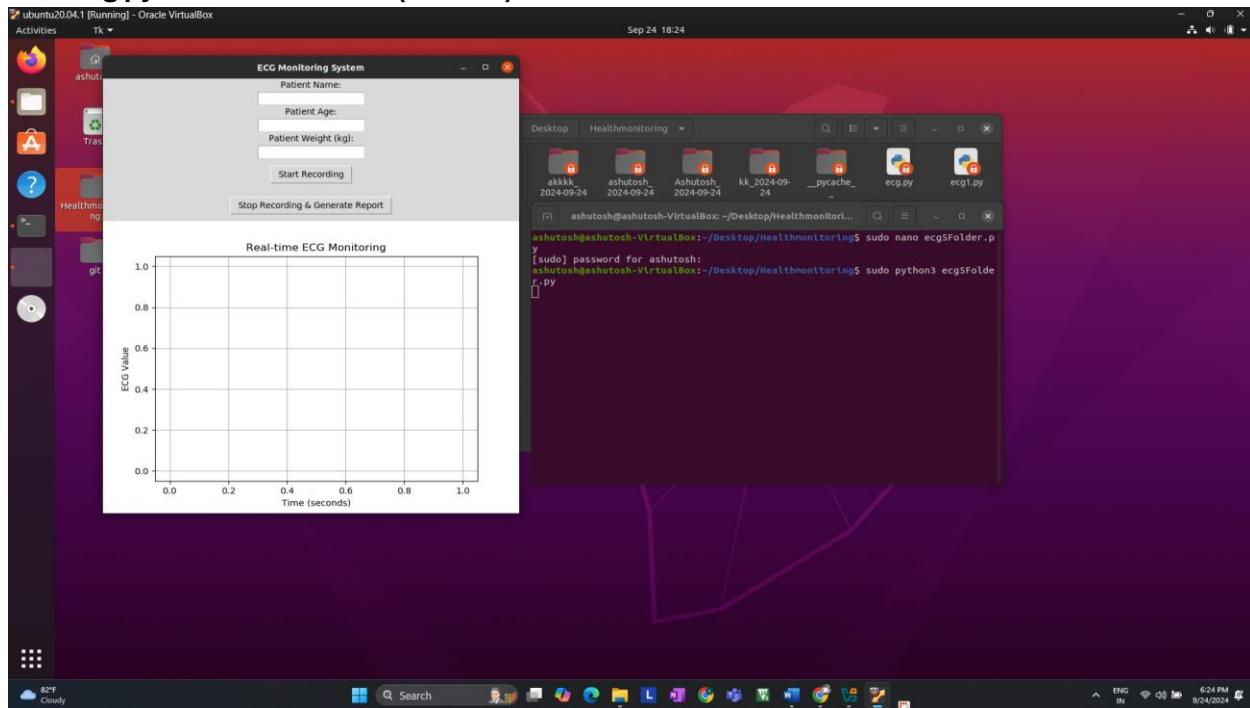
Sep 24 18:32

```
160 pdf.set_font("Arial", size=12, style="B")
161 pdf.cell(100, 10, "ECG Data", ln=True, align="L")
162 pdf.set_font("Arial", size=10)
163 conclusion = f"The ECG data indicates {len(anomaly_report)} anomalies. Further medical review is recommended."
164 pdf.multi_cell(80, 10, conclusion)
165
166 # Save the PDF
167 pdf.output(report_filename)
168 msg = f"Report saved: {report_filename}"
169 messagebox.showinfo("Report", f"ECG report saved as {report_filename}")
170
171 # Set up the Tkinter GUI
172 root = tk.Tk()
173 root.title("ECG Monitoring System")
174
175 # Create GUI with patient details
176
177 age_var = tk.StringVar()
178 weight_var = tk.StringVar()
179
180 tk.Label(root, text="Patient Name:").pack()
181 tk.Entry(root, textvariable=name_var).pack()
182
183 tk.Label(root, text="Patient Age:").pack()
184 tk.Entry(root, textvariable=age_var).pack()
185
186 tk.Label(root, text="Patient Weight (kg):").pack()
187 tk.Entry(root, textvariable=weight_var).pack()
188
189 # Create a button for recording
190 start_button = tk.Button(root, text="Start Recording", command=start_recording)
191 start_button.pack(pady=10)
192
193 stop_button = tk.Button(root, text="Stop Recording & Generate Report", command=stop_recording)
194 stop_button.pack(pady=10)
195
196 # Set up the Matplotlib Figure and axes for the real-time ECG plot
197 fig, ax = plt.subplots()
198 time = ax.plot([], [], label='ECG Data', color='blue')
199
200 ax.set_title("Real-time ECG Monitoring")
201 ax.set_xlabel("Time (seconds)")
202 ax.set_ylabel("ECG Value")
203 ax.grid(True)
204
205 # Embed the plot in the tkinter window
206 canvas = FigureCanvasTkAgg(fig, master=root)
207 canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
208
209 # Animation to update the plot with incoming data
210 from matplotlib.animation import FuncAnimation
211 anim = FuncAnimation(fig, update_plot, interval=200)
212
213 # Set up the Tkinter main loop
214 root.mainloop()
```

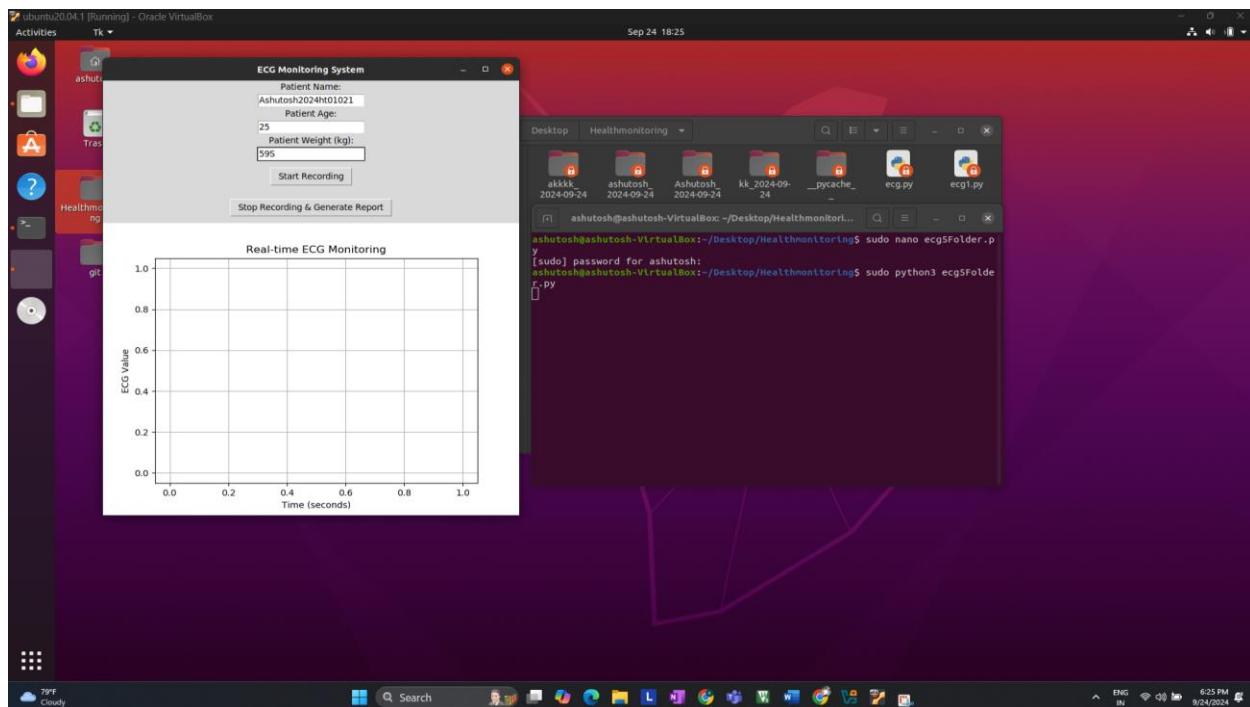
Python Tab Width: 8 Ln 176, Col 26

79F Cloudy 632 PM IN 9/24/2024

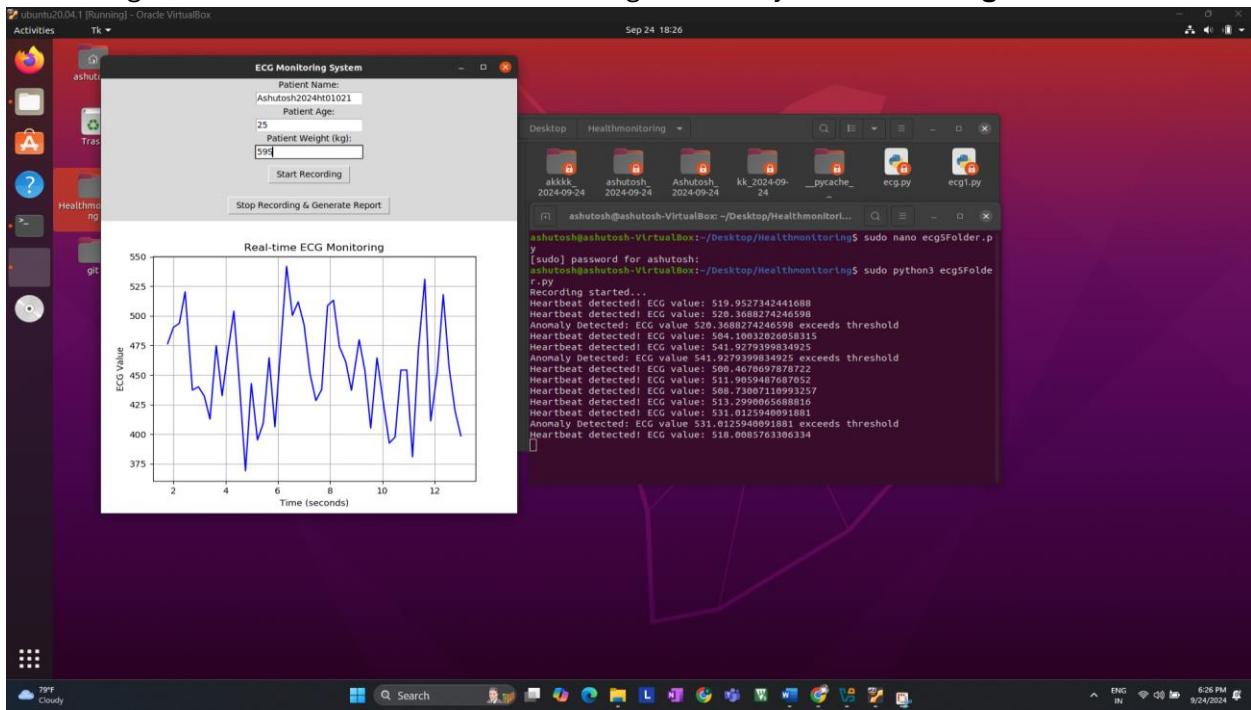
## Running python code in Linux(Ubuntu) Machine



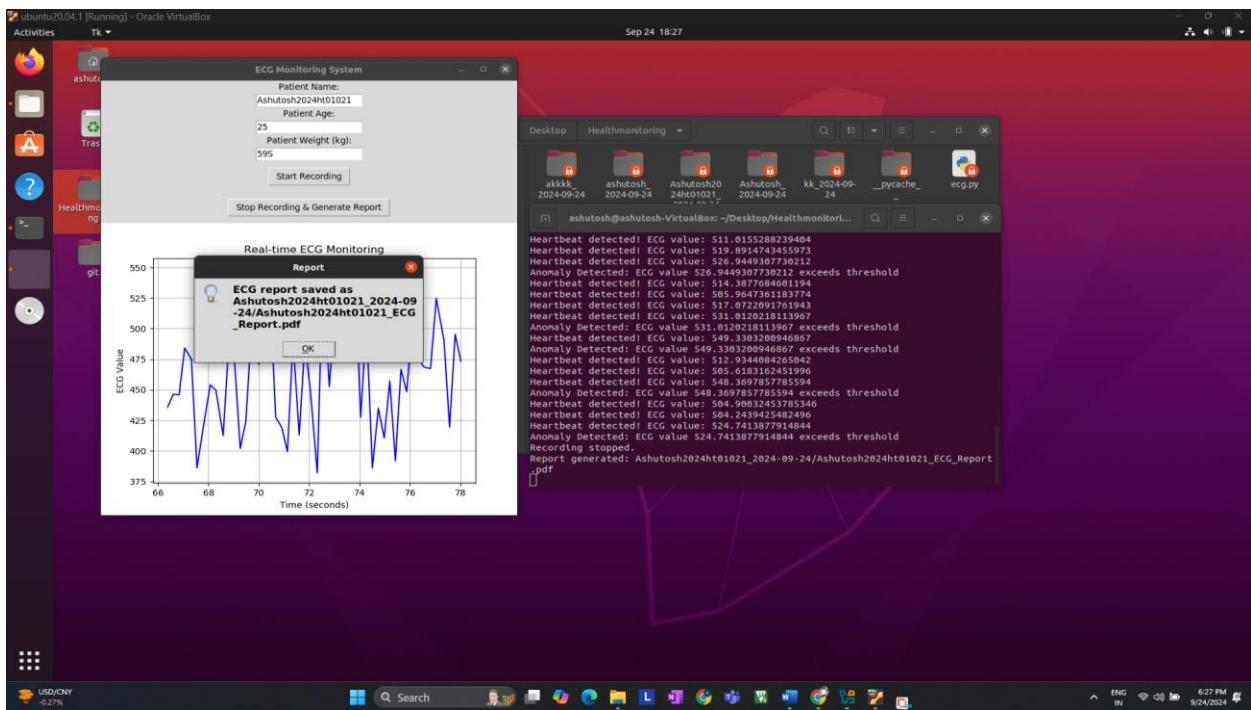
After Running the code, Fill the container with the patient/user details.



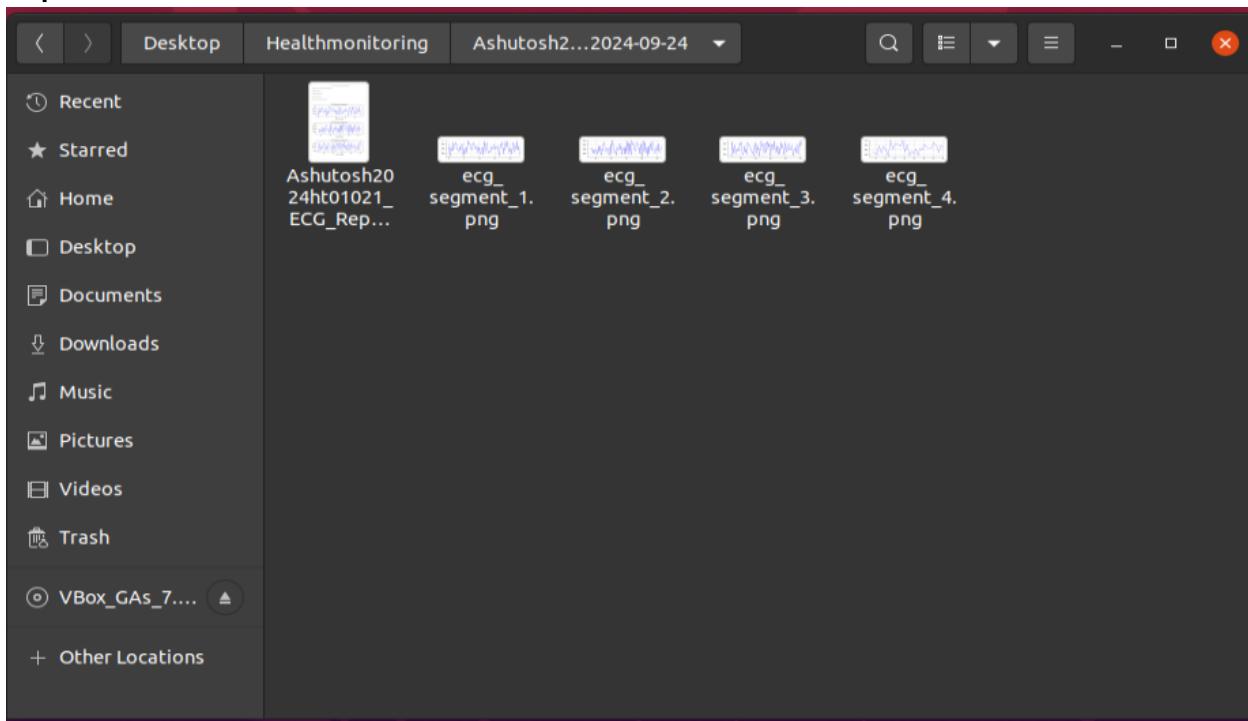
After filling the details and ECG Diodes connecting to the body **start Recording**



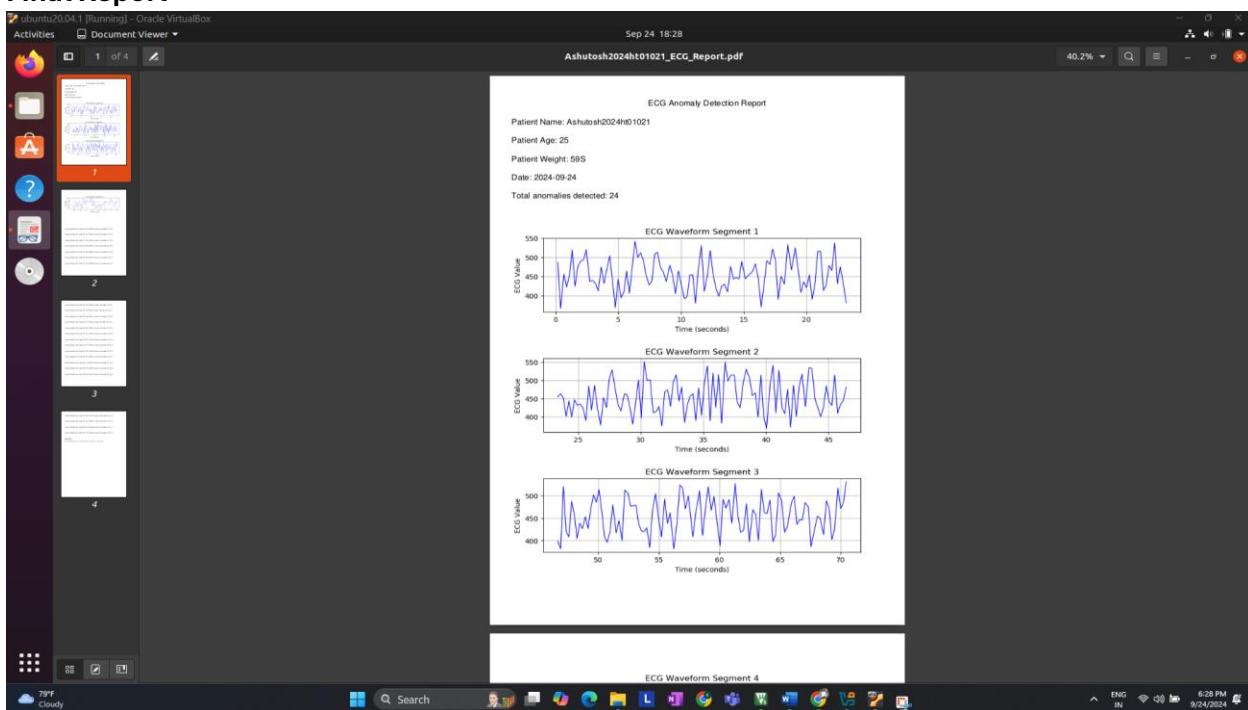
Click **Stop Recording to Generate Report**, then you will ask to save report, click **OK**

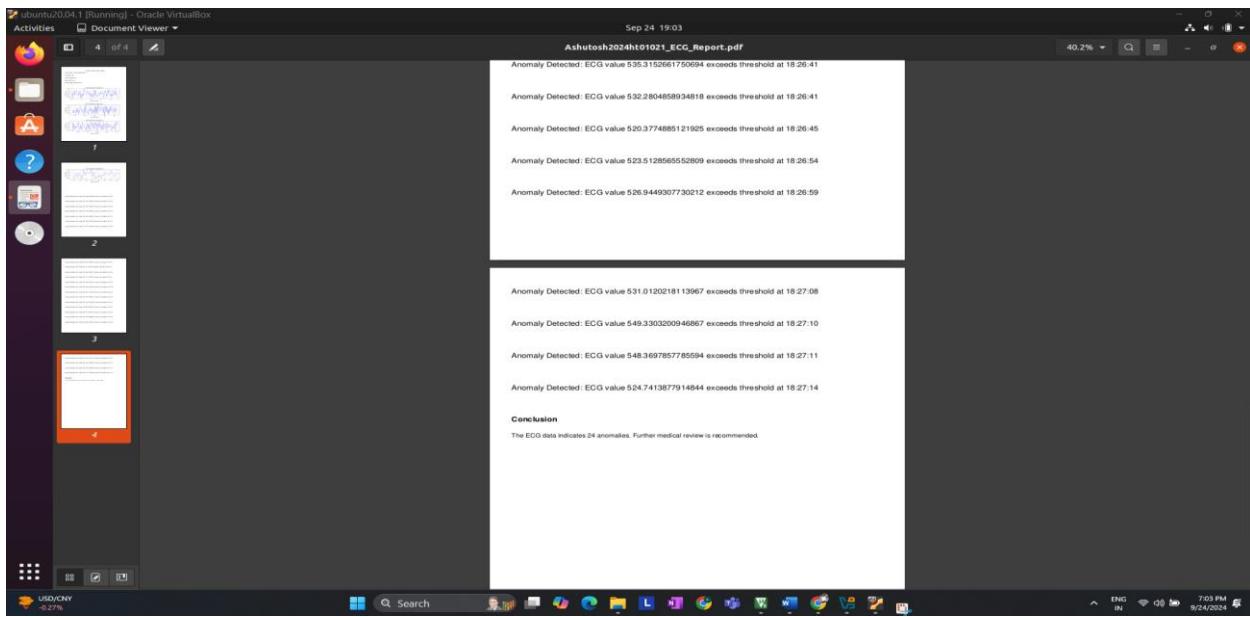


**Report** will be **saved** on same location in **New Folder** with Patient/User name.



## 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](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>