Heart Monitor

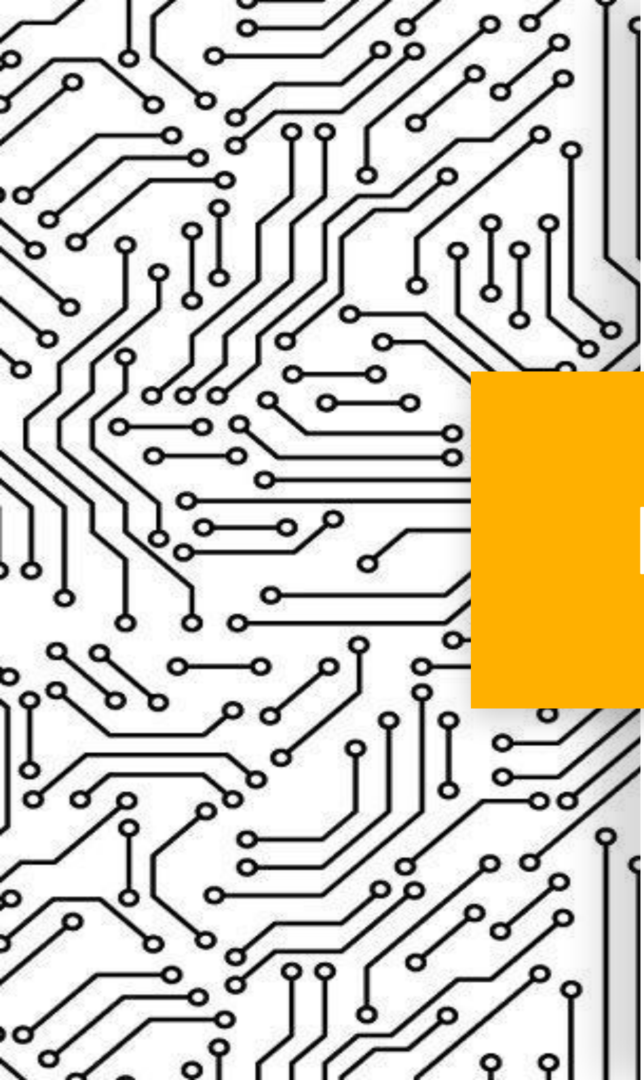
Using STM32F103C8 and AD8232

Outline

- **Overview**
- **Python Application**
  - GUI
  - Code structure (classes, etc)
- **Embedded Application**
  - Hardware Architecture
  - Software Architecture
  - Code structure
  - Configurations
    - Baud rate configuration
    - Power saving configurations
  - BPM Calculation

Overview

This is a heart monitor application using **stm32f103c8** microcontroller and **AD8232** integrated signal conditioning block for ECG. The project is split into an embedded application written in C language to be downloaded on the MCU, and a python application to communicate with the MCU over a serial link.

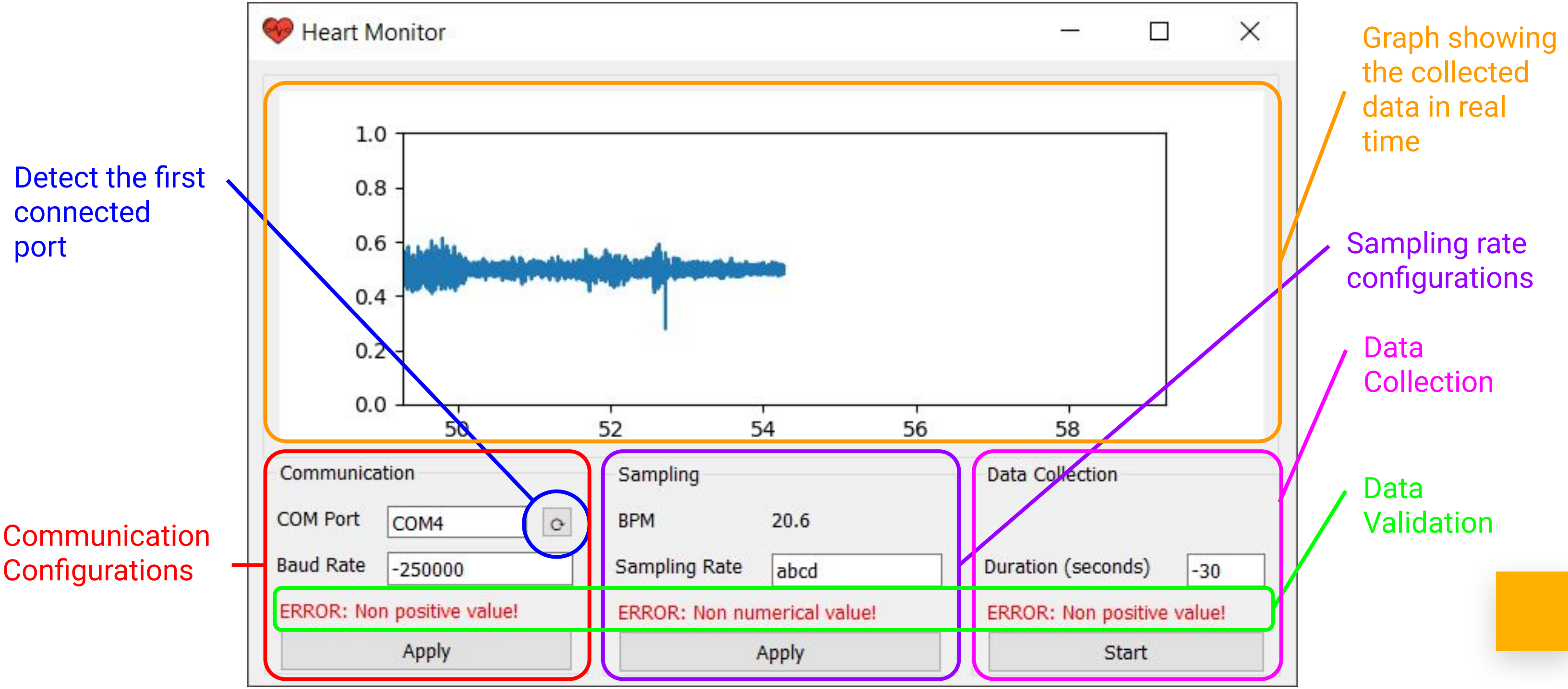


Python Application

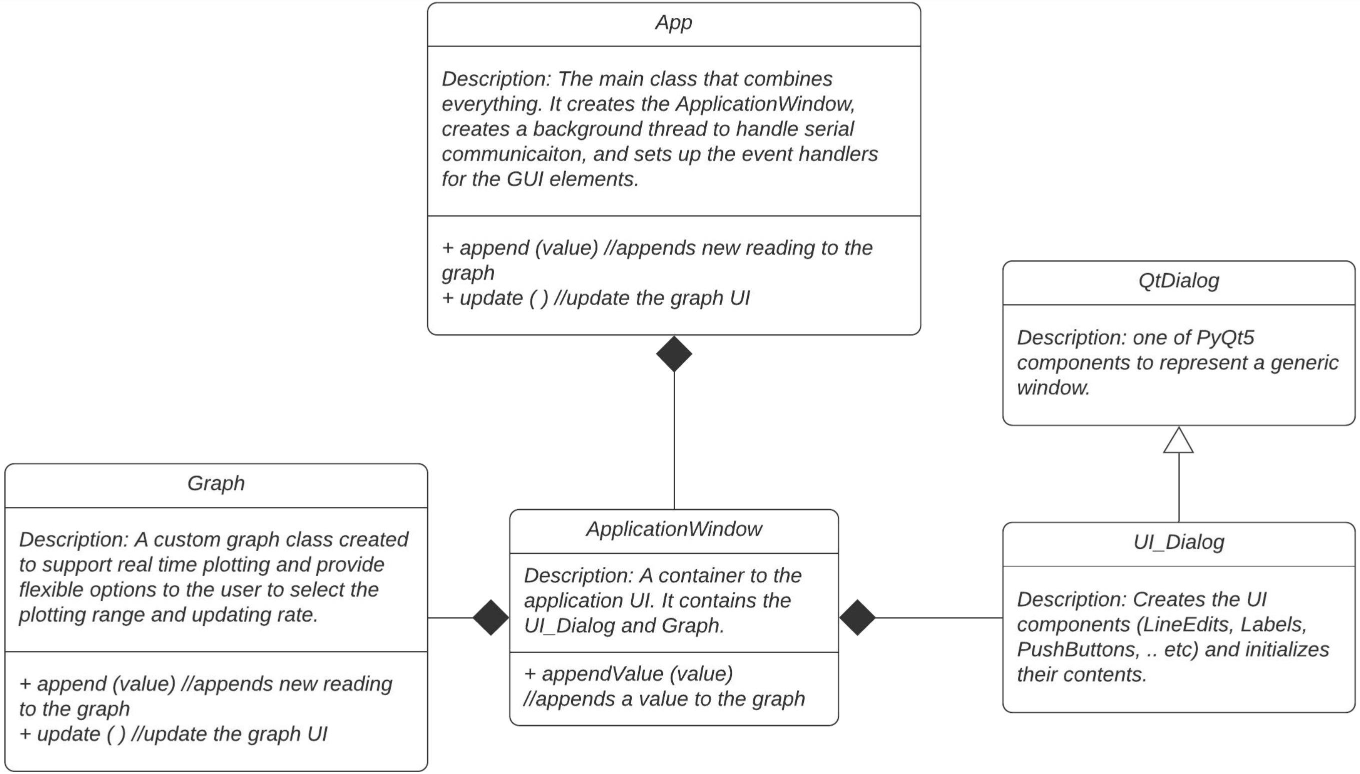
Python Application - Libraries Used

- **PyQt5**
  - Used to design the whole GUI. **QT Designer** was used to design the UI elements visually, then **Pyuic5** utility was used to convert the design to a Python class.
- **Matplotlib**
  - There is no built-in components in PyQt to add the graph to the GUI. There is an external library, QtCharts, that is compatible with PyQt, but it doesn't support real-time updating. So Matplotlib was used for the graph.
- **Pyserial**
  - Used for sending and receiving data between the application and the embedded application over the serial link.

Python Application - GUI



Python Application - Code Structure



Python Application - General Comments

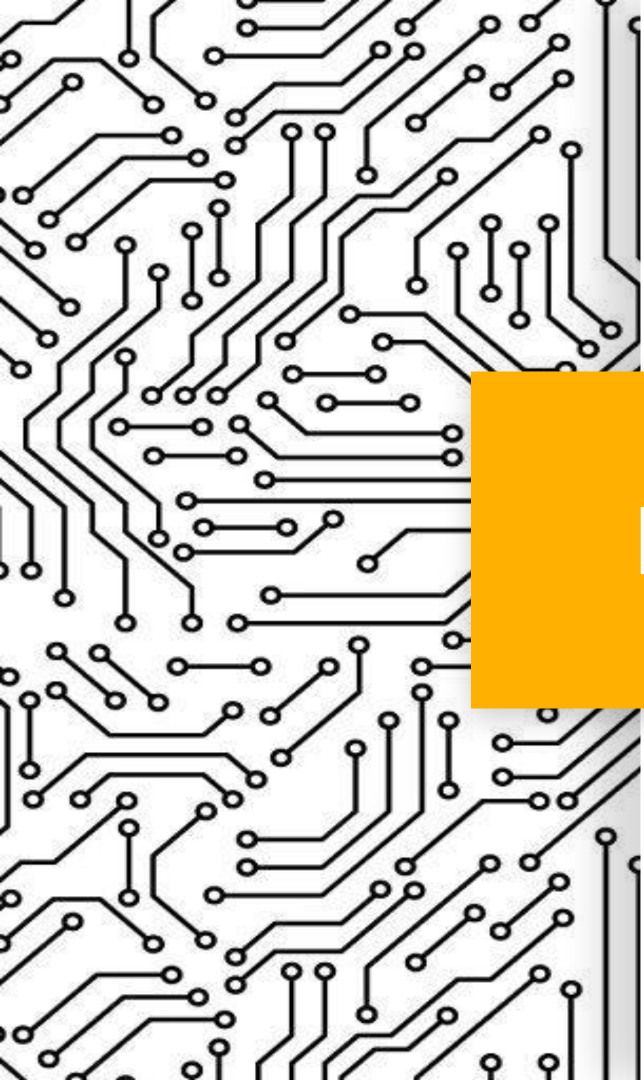
- The GUI is designed for an average user who is not experienced with communication or embedded software:
  - The UI elements are categorized by function into groups for easier user experience.
  - The UI elements contain the default values for communication, with **automatic port detection**.
  - Label text elements are added to report any error in the data validation to the user.

- OOP principles were used to separate the UI creation and logic from the main application.

Python Application - General Comments

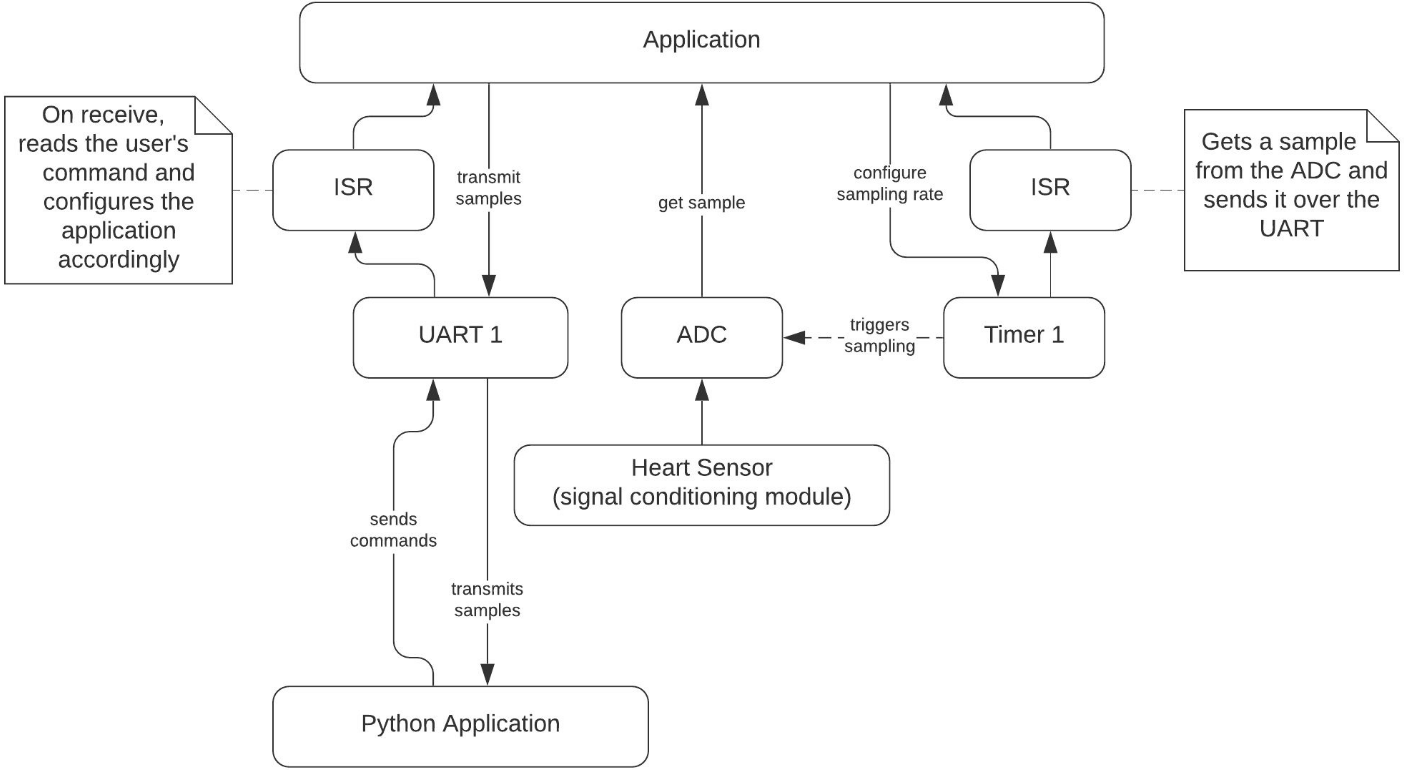
- The serial communication between the application and the board is handled in a **separate background thread** without interfering with the UI logic.

- A custom graph class is created to support real time plotting and provide flexible options to the user.
    - The updating rate is customizable. I used **24 updates/second,** which is very sufficient for smooth user experience. Please note that the update rate of the graph doesn’t affect the effective sampling rate. It shows the data at full resolution.



Embedded Application

Embedded Application - Hardware Arch.



Embedded Application - Software Arch.

- The software architecture is best described by **Round Robin with Interrupts** software architecture, with the main function handling the sampling, the command handling, and the power mode switching. The program has two interrupts:
  - UART: the interrupt is **Aperiodic**. It's only triggered when a user sends a command.
  - Timer: the interrupt is **periodic**. It's used to trigger the ADC sampling and transmission.
- The software is simple and definitely doesn't require a real-time operating system.

Embedded Application - Code Structure

- **Main function:**
  - Handles the sampling and UART transmission of samples
  - Handles the user's command parsing and configuration
  - Puts the microcontroller in the sleep mode when no data is being transmitted. The microcontroller wakes up on receiving an interrupt (a user command). To put the microcontroller in sleep mode and save as much power as possible, the following is done:
    - Stop the timer
    - Stop the ADC
    - Suspend the systick
    - Put the Microcontroller in sleep mode

Embedded Application - Code Structure

- **Timer ISR:** Triggers the conversion and transmission of samples. Sets a ready flag to true, and the actual conversion and transmission is handled by the main function.

- **UART ISR:** On receiving data, the ISR sets a flag to true, then the main function dcodes the command and configure the microcontroller accordingly.

Embedded Application - Baud Rate

- *"In short, ECG signals, even those high frequency intra-QRS complex notches and slurs, will be limited to 2,000 Hz to 2,500 Hz. So a sampling frequency of **5,000 Hz** will be perfect for high-frequency signals "*
  https://www.researchgate.net/post/What_is_the_minimum_acceptable_sampling_frequency_for_ECG_signals

- UART 1 is using 8N1 configuration (10 bits/frame)
  One sample: 4 digits + '\n' = 5 frames (50 bits)

- At baud rate = 9600, the max sampling rate is:
  9600/50 = 192 samples/second (Not sufficient!)

- Needed baud rate = 50 * 5000 = **250,000 bits/sec**

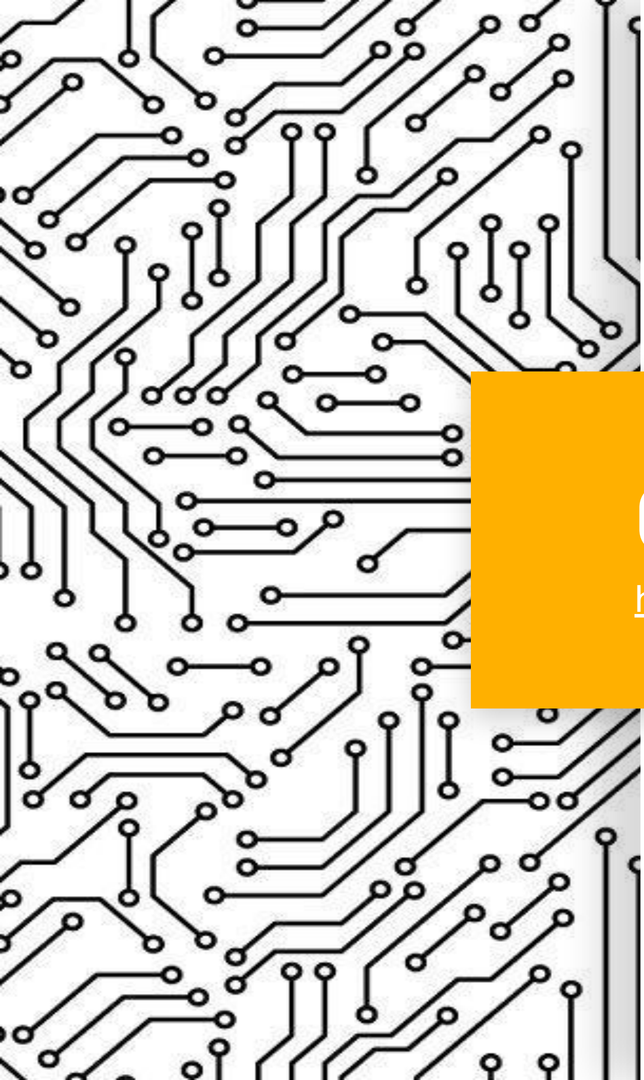
Embedded Application - Power Saving

- **Clock frequency:**
  - ADC maximum sampling rate: 5000 samples/second
  - ADC minimum clock frequency:
    - 12 cycles/sample * 5000 samples/second = 60KHz
  - Since we're using 8MHz clock, it's very safe to set the ADC prescaler to divide by 8.

- **Sleep Mode:**
  - The application sends data on-demand only, when the user asks for 1-minute worth of data. That means that the microcontroller might be idle for a long time when it's not sampling or sending any data. The microcontroller is put to sleep mode and only wakes up when the user sends any commands.
  - The LED is configured to be ON only with normal mode of operation.

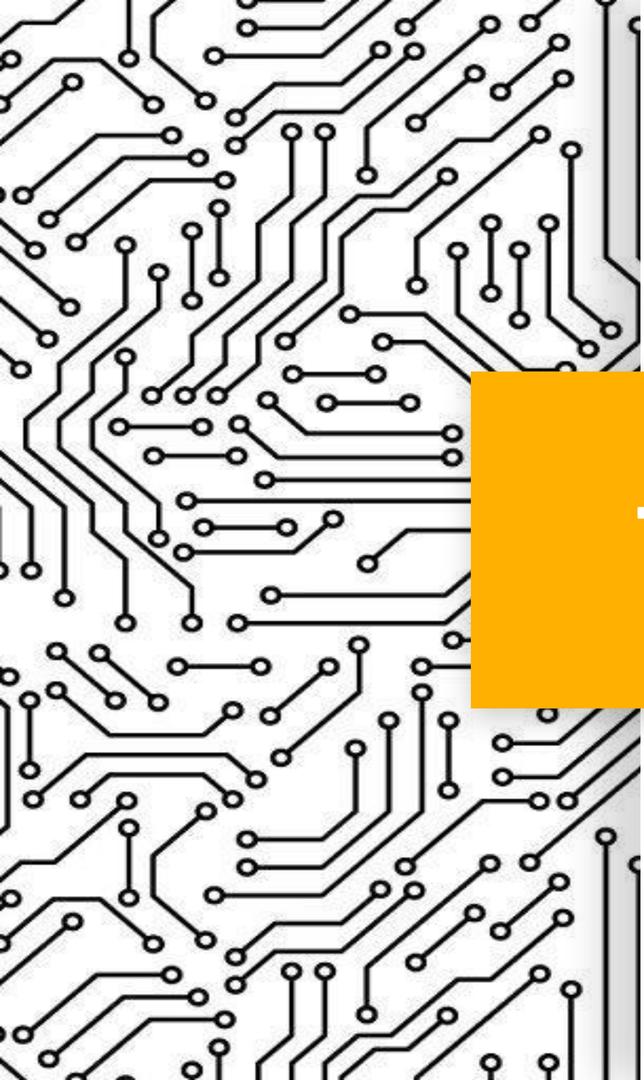
BPM Calculation

- After doing some research, **Pan Tompkins Algorithm** is the most popular algorithm for QRS detection.
- Libraries to use
  - Py-ecg-detectors (https://pypi.org/project/py-ecg-detectors/)
- References:
  - https://cnx.org/contents/YR1BUs9_@1/QRS-Detection-Using-Pan-Tompkins-algorithm
  - https://www.heighpubs.org/hjcr/pdf/acr-aid1018.pdf



GitHub Repository

https://github.com/Alaa-MK/STM32-AD8232-Heart-Monitor-App



Thank You!