Embedded ECG Heart Monitor

Project Report

Embedded Systems

Mohamed A. Abdel Hamed

900163202

Dr. Mohamed Shalan

Spring 2020

Overview & Description

Presented is a real-time embedded ECG heart monitor system application. The embedded side is to operate on an STM32F1 microcontroller (e.g. Black Pill) while the desktop-side is to run on any Python3- supporting environment. The ECG sensor being used is the SparkFun AD8232. The application provides the two basic functionalities of a heart monitor, namely, heart activity plotting, and heart beat rate reporting. Furthermore, the app enables the user to control some side features such as sampling and baud rates in addition to the COM port used. The application also provides a graphical control panel for these features.

This report is dedicated to discussing the following:

I. **Development Environment & Technology**
II. **System Features**
III. **Design & Implementation**
IV. **User Guide & Sample Runs**
V. **Future Development**

I. Development Environment & Technology

Software

Programming languages used:

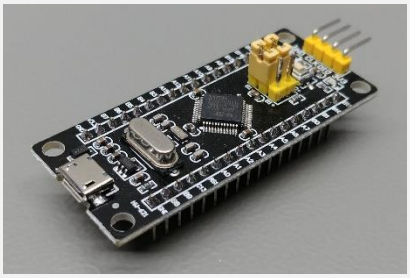
- **Black Pill app:** developed in C
- **PC app:** developed using Python 3.8.2

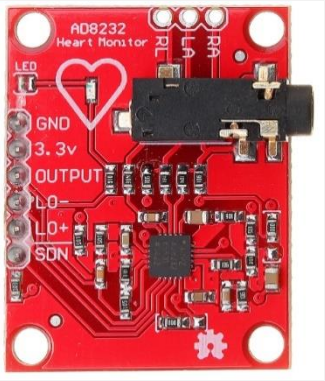
Development software used:

- **Black Pill app: Keil uVision IDE + CubeMX**
- **PC app:**
  - **pySerial:** used for microcontroller-to-PC communication over a serial USB COM port
  - **matplotlib:** used for real-time plotting
  - **pySimpleGUI:** used for graphical interface

Hardware

- **Microcontroller:** STM32F1 Black Pill
- **ECG Sensor:** SparkFun AD8232
- **Serial Cable**: prolific USB-UART cable





II. System Features

Functional Features:

1. **Serial Port Selection:** in the beginning, the user is prompted with the available COM ports for them to select the one, to which the microcontroller is hooked.
2. **Baud Rate Control:** along with the port, the user is also prompted to set the desired baud rate (115200 is recommended).
3. **Sampling Rate Control:** at any point during operation (except for when ECG data collection is ongoing), a user can set the desired sampling rate to be used for collecting heart activity data.
4. **Collect & Plot ECG Data for 1 Minute:** a user can start the one-minute ECG data collection mode where the plot window is launched and updated real-time. Meanwhile, any other interaction is frozen until the plot window is closed.
5. **Report Heart Beat Rate:** after at least one data collection run, the application can report the average HBR based on the collected data.

Non-functional Features:

1. **Leads-off Detection:** the system detects if the sensor pads were not attached properly at some point and produces a flat middle line in the plot.
2. **Premature Data Collection Cancelling:** in case the user closes the plot window midway through an ongoing data transmission, the desktop app signals the microcontroller to halt the data transmission from its side.

III. Design & Implementation

Low-level utilities:

I. Command Extraction:

all commands read from the serial port byte by byte, and are delimited by '$' as shown below. Note that all commands are received in blocking mode due to the synchronous flow of the app. The only exception to this is the premature tear-up signal which is received in interrupt mode.

```
// The three supported commands
char SSR[] = "SSR";
char C1MWD[] = "C1MWD";
char RHBR[] = "RHBR";
```

```
void pc_get_cmd()
{
  memset(cmd, 0, sizeof(cmd));
  strcpy(cmd,"");
  char b[2];
  do
  {
    HAL_UART_Receive(&huart1, (uint8_t *)b, 1, HAL_MAX_DELAY);
    char c = b[0];
    if(*b != '$') strncat(cmd, &c, 1);
  } while (*b != '$');
}
```

II. Command Parsing:

once a command has been extracted, it is then passed for parsing it by its predefined handle (e.g. SSR), and then the corresponding procedure is selected based on that:

```
void parse_cmd()
{
  if (strncmp(cmd, SSR, strlen(SSR)) == 0)
  { ···
  }
  else if (strncmp(cmd, C1MWD, sizeof C1MWD) == 0)
  { ···
  }
  else if (strncmp(cmd, RHBR, sizeof RHBR) == 0)
  { ···
  }
}
```

In the following two sub-section, the implementation of each of the aforementioned System Features are to be discussed.

Functional Features:

i. Serial Port Selection:

This is readily achieved through the pySerial utility tools as shown below:

```
# Get list of available ports
comports_list = list(map(lambda p: p.device, serial.tools.list_ports.comports()))
```

Note that it takes some time initially to detect the available ports (~5 seconds)

ii. Baud Rate Control:

This is specified when the serial port is opened initially:

```
serial_p = serial.Serial(port=com_port, baudrate=baud_rate,
                         bytesize=8, timeout=2, stopbits=serial.STOPBITS_ONE)
```

iii. Sampling Rate Control:

Sampling rate controlled through the period of GPTIM2 which triggers the ADC1 conversions with each elapsed period when uC is in data collection mode.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if(!is_collecting_data) return;
  if (htim->Instance == TIM2)
  {
    // Check LO- & LO+
    if((HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_8) ||
      HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_9))) {
      HAL_UART_Transmit(&huart1, (uint8_t *)"!\n",
        strlen("!\n"), HAL_MAX_DELAY);
    }
    else HAL_ADC_Start_IT(&hadc1);
  }
```

When a user sets the heart beat rate through the control panel, a formatted command is sent to the microcontroller (via UART), which gets parsed and the auto-reload register (ARR) of the GPTIM2 is adjusted according to the following formula:

- ARR = $\frac{1000}{X}$; $where\ X = desired\ sampling\ rate$

Given that the GPTIM2 source clock is 8 MHz, and the prescaler is fixed at (8K – 1), so the timer trigger rate becomes equivalent to the sampling rate. The allowed range for the sampling rate is between 1 and 1000.

iv. Collect & Plot ECG Data for 1 Minute:

When this mode is triggered by the user, the microcontroller stops waiting for new commands and keeps sending numerical data representing the pulse magnitude relative to the ADC range (12 bits = 0 to 4095). Every data reading is sent in a new line as shown:

```
sprintf(out, "%d\n", adc_value);
HAL_UART_Transmit(&huart1, (uint8_t *)out, strlen(out), HAL_MAX_DELAY);
```

When the uC is done sending data, it sends a pre-set ACK msg to signal the desktop app to stop updating the plot.

```
char ack_msg[] = "OK\n";
```

```
void done_ack()
{
  HAL_UART_Transmit(&huart1, (uint8_t *)ack_msg, sizeof ack_msg, HAL_MAX_DELAY);
}
```

The real-time plot update is done by periodically redrawing the plot to reflect any new changes. In this app, the plot is refreshed every 10 ms:

```
ani = animation.FuncAnimation(fig, animate, fargs=(xs, ys), interval=10)
```

So that achieves a refresh rate of around 100 fps. It is advised to reduce that rate for devices (PCs) that cannot support that refresh rate.

v. Report Heart Beat Rate:

A beat is defined as a pulse that's at least ~60% of the ADC scale in magnitude that follows another pulse that is below that threshold. Instantaneous HBR is calculated as follows:

$$\boldsymbol{HBR_{inst}(bpm) = \frac{60000}{(t_{pulse})_i - (t_{pulse})_{i-1}};} \quad \boldsymbol{t_{pulse}\ is\ in\ ms}$$

The final result is the average of all instantaneous HBRs.

Due to limitations on the operation guarantees at the user side, the HBR average is reset before each data collection to prevent previous poorly operated (e.g. disconnected pads) runs from corrupting the HBR.

Non-functional Features:

i. **Leads-off Detection:** this is done by digitally reading the GPIO pin value of the LO+ & LO- sensor signals, and if they're high, the microcontroller sends a special pre-set character '!' to signal the PC app to flatten the plot at that point.

```
// Check LO- & LO+
if((HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_8) ||
  HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_9))) {
  HAL_UART_Transmit(&huart1, (uint8_t *)"!\n",
    strlen("!\n"), HAL_MAX_DELAY);
}
else HAL_ADC_Start_IT(&hadc1);
```

ii. **Premature Data Collection Cancelling:** this is achieved by sending a pre-agreed character '#' to the uC to signal it to stop. This character is received in **UART interrupt mode** because we cannot afford to block for an interruption signal during non-stop transmission.

IV. User Guide & Sample Runs

This guide assumes the user has the hardware described before (along with a means to download the hex code to the uC) in addition to the required software (including the Python packages).

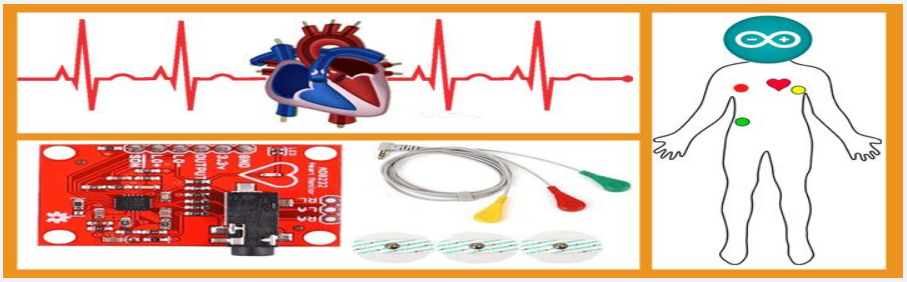
❖ Below are the steps to use the system:

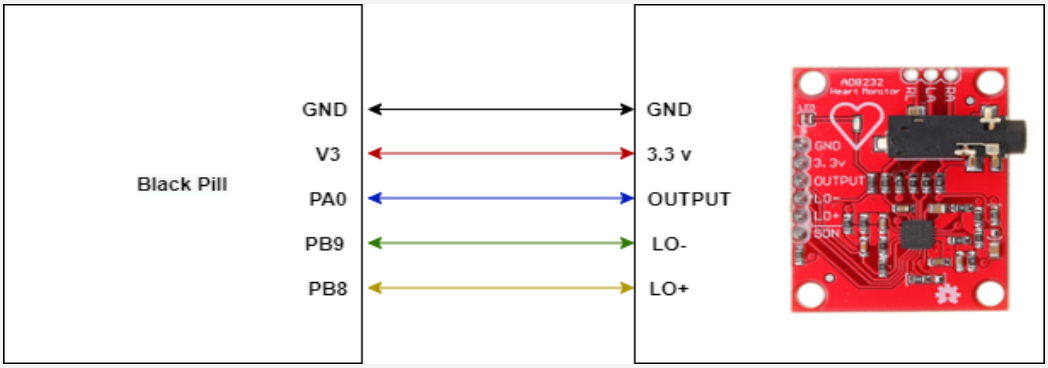
i. Make sure all connections are as shown below.

ii. Download the hex code to the Black Pill uC (e.g. though serial programming)

iii. Put the uC into operation mode (if you're using SP) and click the Reset button.

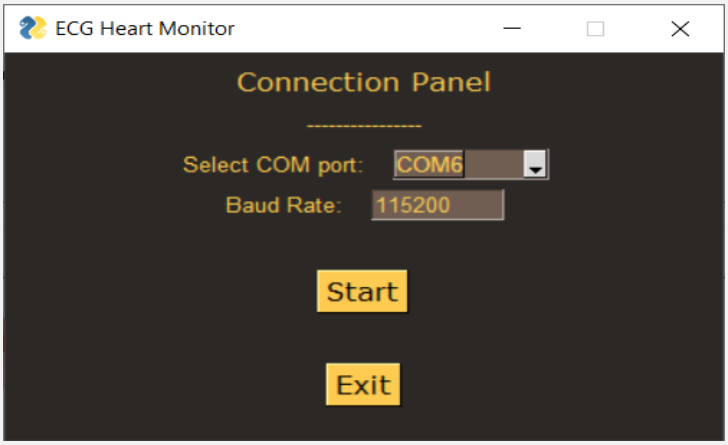
iv. Go to the "**main_app**" directory and run the following command "**python main.py**"



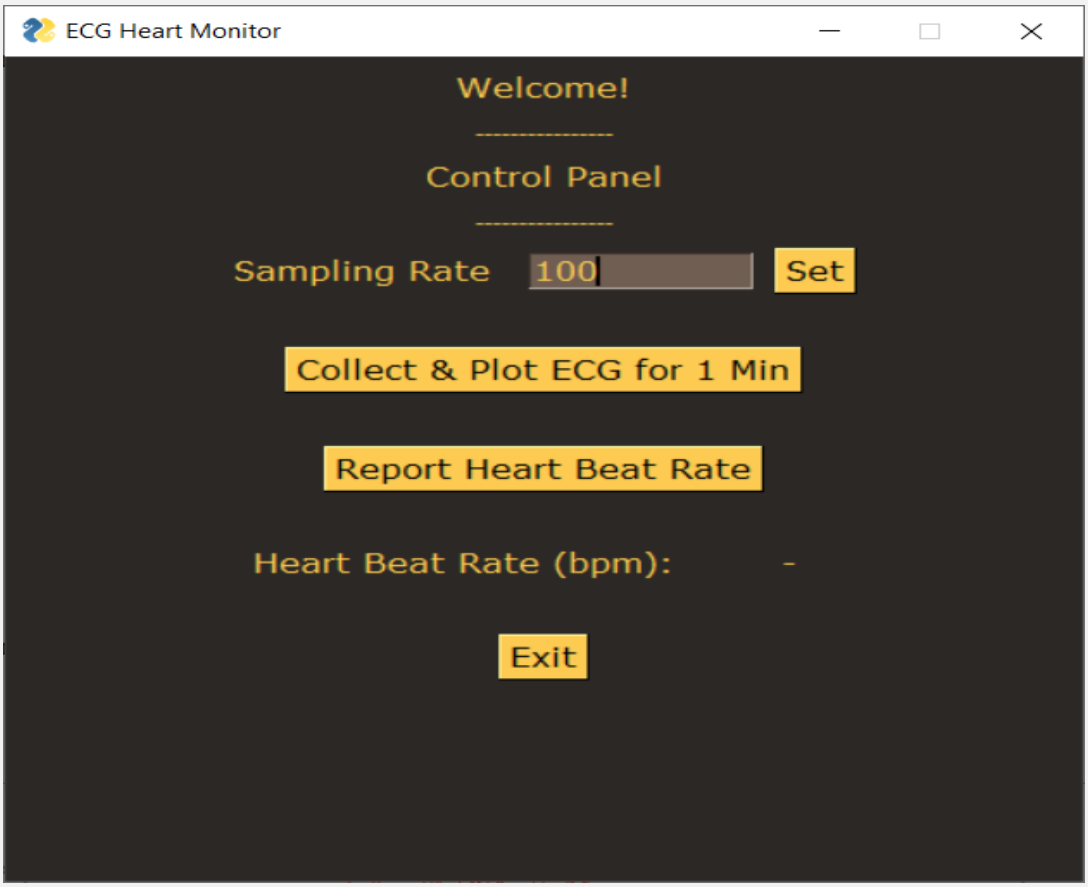


Sample Run:

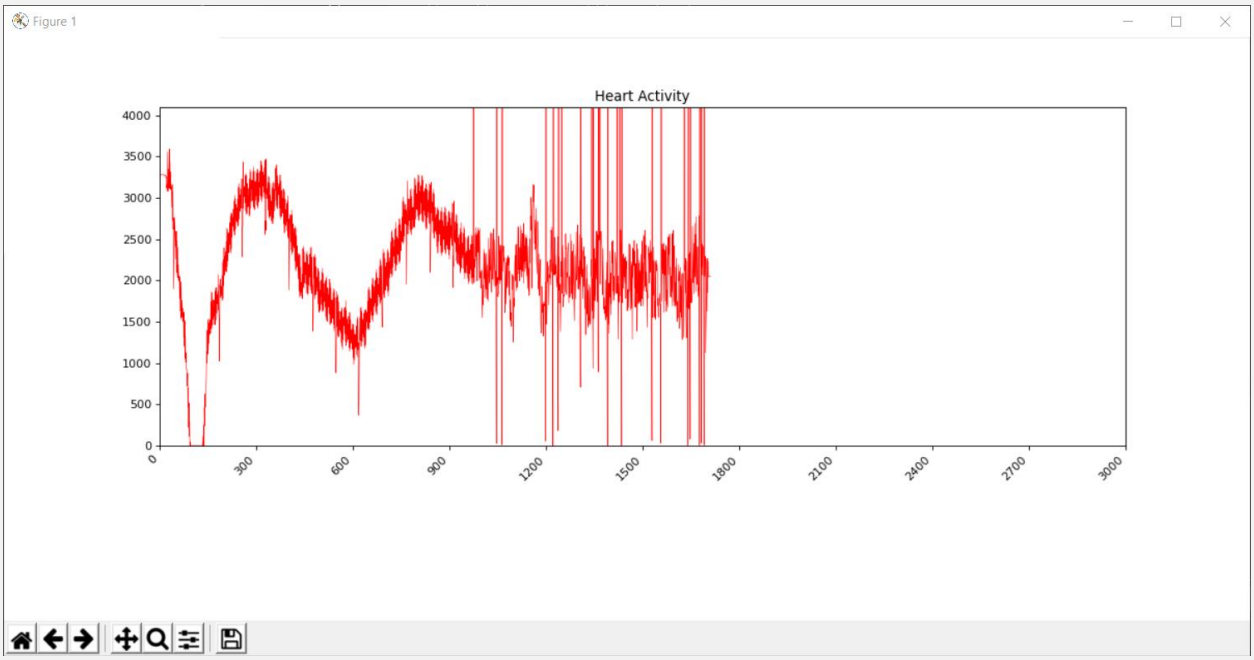
1. **Connection settings:**



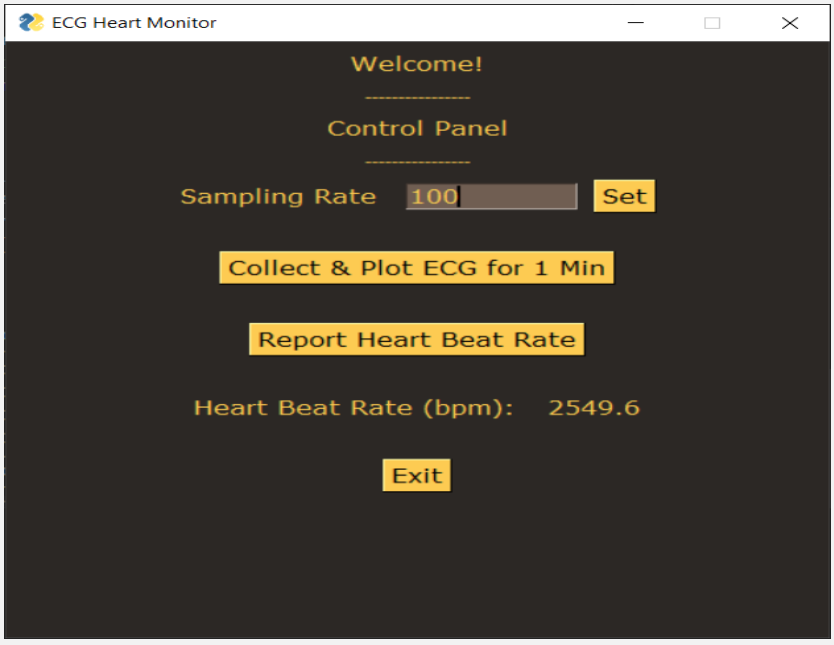
2. **Sampling rate:**



3. Data collection & plotting



4. Heart beat rate:



Note: HBR is staggeringly high due to worn-out pads which cause great disturbance in the sensor output.

V. Future Development & Improvement

The following could be a potential improvement on the presented system:

1. Using DMA for managing ADC conversions instead of interrupts
2. Utilizing the SDN (shut down) signal of the AD8232 sensor to save power when we're not sampling its output.
3. Using a smoother technique for updating graph data in real-time
4. Adding a layer of fault tolerance around port binding in case of failures to connect to UART.

References

- **AD8232 Sensor Datasheet**
- **STM32F1 Datasheet**
- **https://learn.sparkfun.com/tutorials/graph-sensor-data-with-python-and-matplotlib/update-a-graph-in-real-time**
- **https://pythonhosted.org/pyserial/index.html**