Manar Abdelatty | 900152684

Heart Rate Monitor

I. Block Diagram

The application is supposed to collect ECG signals from the AD8282 heart monitor sensor. The collected signal is then sent to the PC for display. The PC interacts with the microcontroller via a predefined set of commands. These commands include setting ADC sample rate, collecting one minute worth of data, and calculating the heart beats per minute.

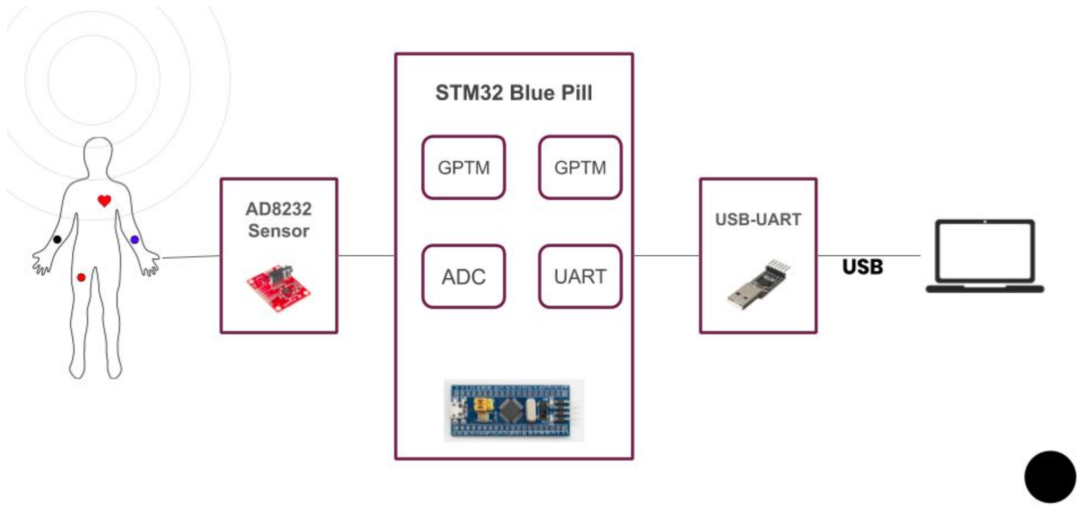


Fig.1 Block diagram.

The block diagram is shown if fig.1. First, the AD8232 sensor collects electrical signals from the three electrodes (red, yellow,and green) placed on the right hand, left hand, and right leg respectively. The sensor then amplifies this signal which is sampled and digitized by the STM32 internal ADC module. Two general purpose timers are utilized for this application. The first one is for controlling the ADC sample rate and triggering the ADC conversion and the second one for counting for one minute duration to mark the start and the end of the data collection process. One UART module is also utilized to facilitate serial communication between the STM32 and the PC application. The USB-UART is used as a bridge to be able to provide usb connectivity to the PC with UART interface.

II. Requirements

The following requirements are specified:

1) Serial communication.

Communication between the PC and the microcontroller is done via UART protocol with a minimum baud rate of 40 KHz. The minimum baud rate calculations are shown below.

```
Maximum ADC sample rate  = 1000 KSPS
ADC sample size = 32-bit

UART frame format is 8N1 which means for every ADC sample two UART frames are
generated.The total number of bits for each sample is calculated as follows:

Frame bits = 1-bit start + 8-bit data + 1-bit stop = 10-bits
Number of frames/sample = 32 / 8 = 4 frames for each sample
Number of bits/sample = 4 * 10-bits = 40-bits

Number of bits/second = Max Sample rate * number of bits per sample
Number of bits/second = 1000 KSPS * 40-bits = 40000 Hz = 40 KHz

Accordingly, the UART min baud rate is 20000 KHz.
```

2) Command Set.

Communication between the PC and the microcontroller is attained by three commands with a fixed length of 16-bytes. The command size is fixed so that the microcontroller can generate an interrupt only when it has received a complete command. The supported commands are shown in table.1. The data command and hbpm command are padded with extra characters so that their max size is 16 bytes.

Command	**Description**	**Max Size**
rate={new_rate};	Sets the ADC sample rate to new_rate value in Hz. For example sending “rate=1000;” sets the ADC sample rate to 1000 KHz.	10 bytes
data;	Collects one minute worth of data from the ECG sensor and sends it to the PC.	5 bytes

hbpm;	Computes heart beats/minute and sends it to the PC.	5 bytes

Table.1 Supported commands.

3) Sample Rate.

The ECG signal sample rate range is **[150 SPS to 1000 SPS].** Typically, 150 SPS is used for most applications while 1000 SPS is used for high precision equipment.

```
Max beats per minute for 30-year old = 190 bpm
Max beats per second for 30-year old = 190 bpm / 60 seconds = 3 bps

Fundamental frequency (fo) = 3 Hz
Cutoff frequency (fc) = 5 * fo = 15 Hz
Sample rate = 10 * fcut_off = 150 SPS
```

4) Setting ADC Sample Rate.

This is achieved by controlling the ADC sample rate via a general purpose timer. The general purpose timer has a fixed pre-scalar and system clock and a variable auto-reload register (ARR) value. The ARR value reflects the counter period and it is set according to the new sample rate as follows:

```
pre-scaler = 3999 , sysclk = 8 MHz
ARR = ((float) sysclk / (pre-scaler + 1) / (float)new_sample_rate) - 1;
```

5) Data Collection.

Collecting ECG signal for one minute and sending it over UART for real-time display. The one-minute duration is marked by a general purpose timer. The counter has fixed values for the prescaler and ARR registers. The timer configurations are calculated as follows.

```
pre-scaler = 8000, sysclk = 8 MHz

This way the timer counter operates at 1 KHz clock. In order to count 60
seconds, the ARR register value is set as follows,

ARR= 60 seconds * 1000 cycles/second = 60000 cycles
```

6) **Computing heart beats per minute (bpm) .**

The heat beats per minute (bpm) is calculated using the ecg 300-method. First, the number of large boxes between two of the ecg signal peaks is obtained where each big box corresponds to 1 second as shown in fig.2. Then, the heart rate is calculated by dividing 300 by the number of large boxes. In order to count the number of large boxes between two peaks, the time distance between the two peaks is measured and divided by 1 second (resolution of the large box).

```
        Bpm = 300 / #of large boxes between two peaks
#of large boxes between two peaks = peaks_distance / 1s
```

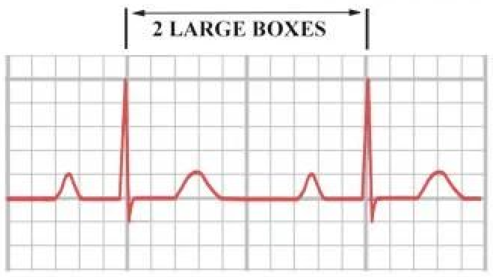


Fig.2 RR-Interval for ECG signal

Since we are dealing with a digital signal, the signal is first processed in order to detect the peaks and measure the distance between two peaks.This is done on three steps shown in fig.3:

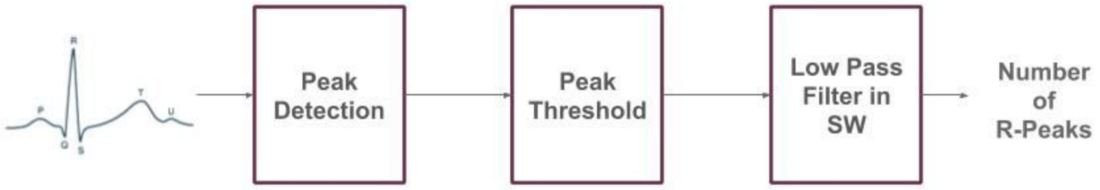


Fig.3 Counting peaks flow.

1) **Peak Detection**: firstly, the peaks are detected by comparing the adc sampled value to its two neighbors; if it is larger than both of them, then we have a local maximum.

2) **Peak Threshold**: Then, in order to count this local maxima as R-peak , it has to satisfy two conditions. The first one is that it passes a certain threshold so that

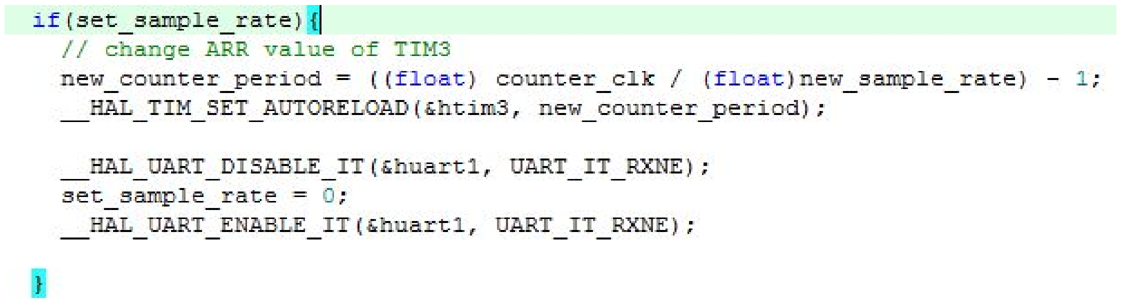
we could eliminate lower peak values. So, the detected peaks are marked as R-peaks if they pass a certain threshold value. This step removes lower peaks like T-peak and p-wave peak.

3) **LPF:** The second condition is that the distance between the detected peak and the previous peak is greater than 0.6s (minimum RR interval). This check is mainly done to eliminate high frequency noise signals. The flow of the R-peak detection is shown in fig.3. The peak detection step detects all peaks in the ecg signal.

III. Implementation & Software Architecture

The embedded application is implemented using *round robin with interrupts* architecture. The *main loop* continuously polls over the three flags to check if a command is received.

- If a rate command is received, a new value for the ARR register of the sample rate timer is computed as shown in fig.3. The new value reflects the timer period and is set using `__HAL_TIM_SET_AUTORELOAD` function.



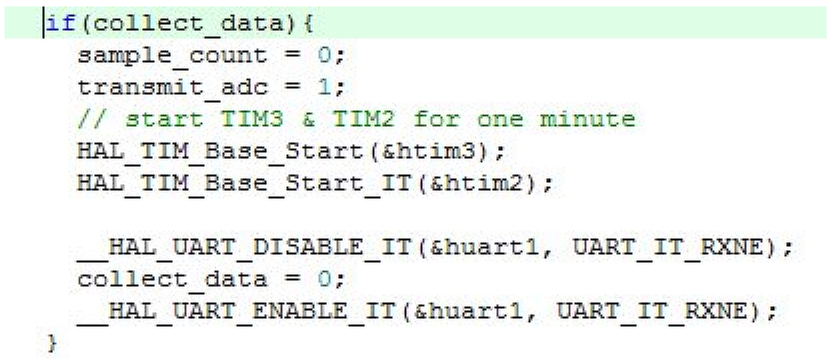
```
if(set_sample_rate){
  // change ARR value of TIM3
  new_counter_period = ((float) counter_clk / (float)new_sample_rate) - 1;
  __HAL_TIM_SET_AUTORELOAD(&htim3, new_counter_period);

  __HAL_UART_DISABLE_IT(&huart1, UART_IT_RXNE);
  set_sample_rate = 0;
  __HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE);

}
```

Fig.3 setting new sample rate.

- If a data command is received, the sample rate timer and the one minute timer are started. The sample rate timer is started with `HAL_TIM_Base_Start` function which starts the timer with disabling the interrupt mode meaning that no interrupts are generated when the timer period elapses. While the one-minute timer is started with `HAL_TIM_Base_Start_IT` function which starts the timer in interrupt mode meaning that an interrupt is generated when the timer period elapses. Consequently, the ADC conversion is started. The data collection is stopped inside the timer period elapsed interrupt.



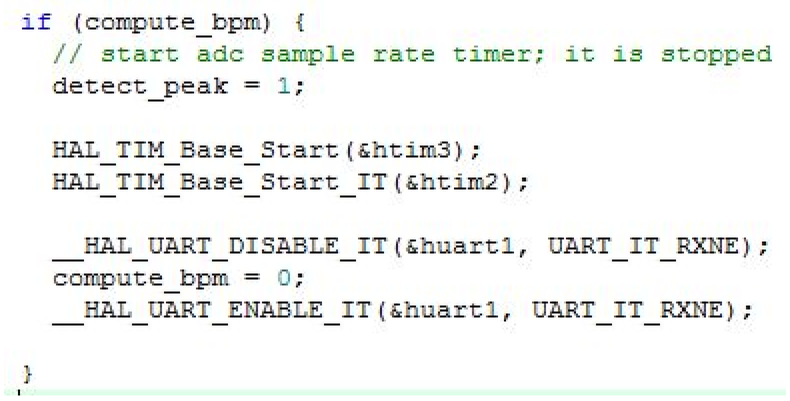
```
if(collect_data){
  sample_count = 0;
  transmit_adc = 1;
  // start TIM3 & TIM2 for one minute
  HAL_TIM_Base_Start(&htim3);
  HAL_TIM_Base_Start_IT(&htim2);

  __HAL_UART_DISABLE_IT(&huart1, UART_IT_RXNE);
  collect_data = 0;
  __HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE);
}
```

Fig.4 Collecting ECG signal data command.

.

- If a hbpm command is received, the ADC timer is started to start the ADC conversion. Also, the one minute timer is started to measure the distance between the ECG signal peaks. Inside the ADC callback function, the `countPeaks()` function is called to calculate the heart beats per minute. After the hpm is calculated, the two timers are stopped to stop the ADC conversion.



```
if (compute_bpm) {
  // start adc sample rate timer; it is stopped
  detect_peak = 1;

  HAL_TIM_Base_Start(&htim3);
  HAL_TIM_Base_Start_IT(&htim2);

  __HAL_UART_DISABLE_IT(&huart1, UART_IT_RXNE);
  compute_bpm = 0;
  __HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE);

}
```

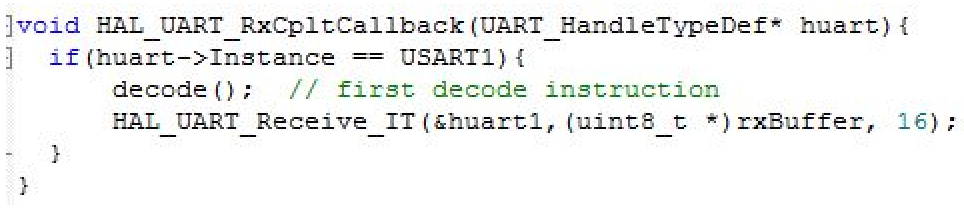
Fig.5 computing bpm command.

To avoid data sharing problems, the UART interrupts are disbaled whenever one of the three flags is reset using `__HAL_UART_DISABLE_IT`. This prevents interrupts from preempting the main loop while setting the flags to zero. Then, the receive interrupt is enabled again after resetting the flag using `__HAL_UART_ENABLE_IT`.

The application has three main interrupts:

1) UART Receive Interrupt

This interrupt is fired whenever the UART buffer has received 16-bytes which marks the receipt of a complete command from the PC application.Inside this interrupt the received command is decoded by `decode()` function which parses the received command and sets one of the following three flags: `set_sample_rate`, `collect_data` and, `compute_bpm` indicating whether it is a `rate`, `data`, or `hbpm` command.

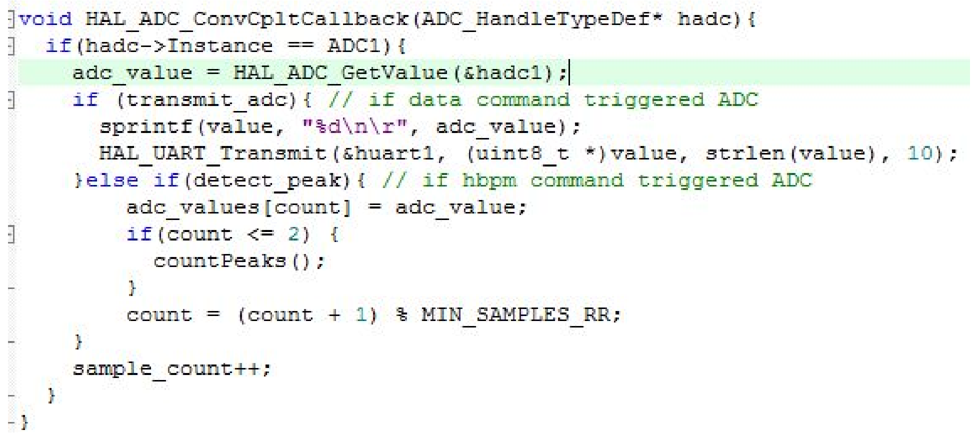


```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef* huart){
  if(huart->Instance == USART1){
      decode();  // first decode instruction
      HAL_UART_Receive_IT(&huart1,(uint8_t *)rxBuffer, 16);
  }
}
```

Fig.6 UART_Receive Interrupt callback function.

2) ADC Conversion Complete Interrupt

This interrupt is generated every time the ADC successfully completes one conversion.The digitized value is saved by using the `HAL_ADC_GetValue` function. Inside the interrupt callback function, two flags are checked to indicate which command triggered the conversion.If it is a data command, the digitized value is then sent over uart to the PC for real-time display using `HAL_UART_Transmit`. If it is a hbpm command, the converted ADC values are buffered and the `countPeaks()` function is called to calculate the bpm. Then, the calculated bpm is sent to the PC application via `HAL_UART_Transmit.`

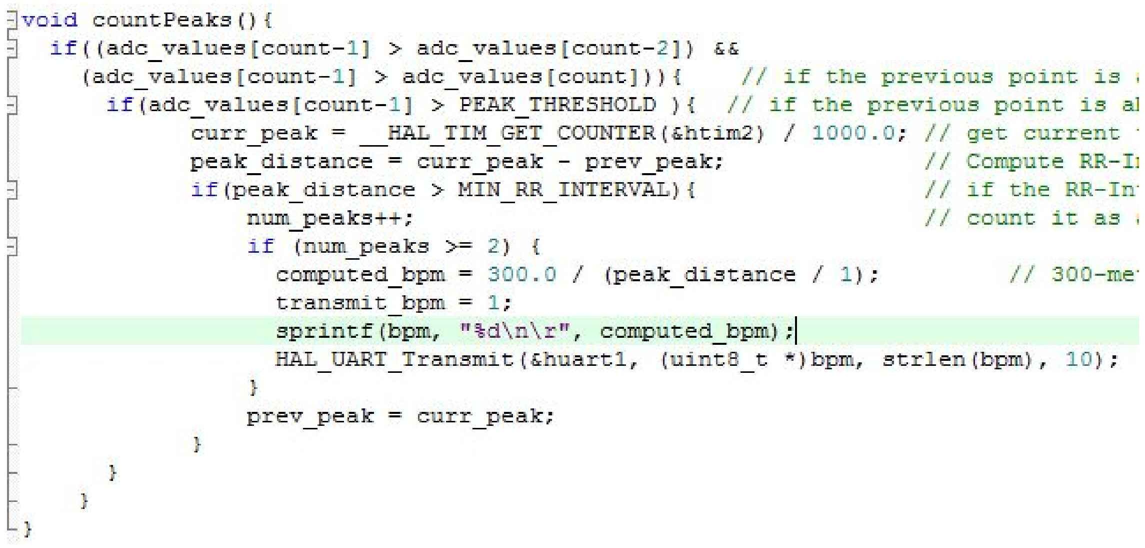


```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc){
  if(hadc->Instance == ADC1){
    adc_value = HAL_ADC_GetValue(&hadc1);
    if (transmit_adc){ // if data command triggered ADC
      sprintf(value, "%d\n\r", adc_value);
      HAL_UART_Transmit(&huart1, (uint8_t *)value, strlen(value), 10);
    }else if(detect_peak){ // if hbpm command triggered ADC
        adc_values[count] = adc_value;
        if(count <= 2) {
          countPeaks();
        }
        count = (count + 1) % MIN_SAMPLES_RR;
    }
    sample_count++;
  }
}
```

Fig. 7 ADC_Conversion_Complete interrupt callback function.

The `countPeaks()` detects the R-peaks in the ecg signal, computes the time difference between two peaks, and finally calculates the beats per minute (bpm) using the 300-method. All peaks in the signal are detected by comparing the adc converted value to its two neighbors (left and right neighbors);if it is larger than the two, then it is marked as an R-peak if it satisfies the following two conditions: (1) it is a larger than a certain threshold (2) if the time difference between this peak and the prev peak is larger than

0.6s (minimum RR interval) .The bpm is calculated using the 300 method wherein the heart rate is obtained by dividing 300 by the peak distance in time. The calculated bpm is then sent over UART to the python application. After that, the two timers are stopped to end the ADC conversion. The implementation of the function is shown in fig.8.

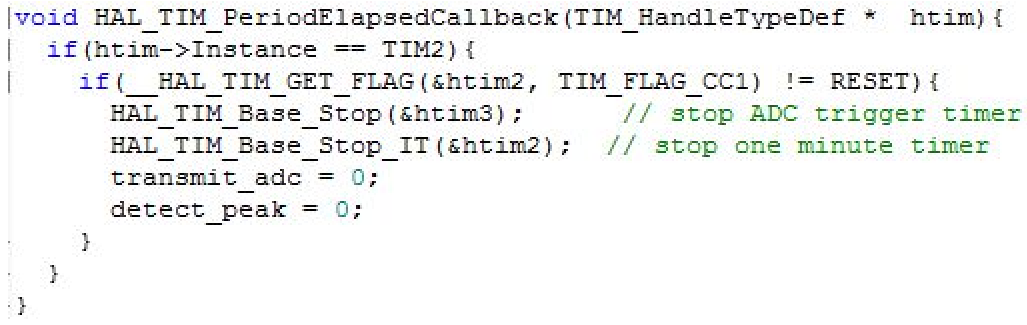


```
void countPeaks(){
  if((adc_values[count-1] > adc_values[count-2]) &&
    (adc_values[count-1] > adc_values[count])){    // if the previous point is
      if(adc_values[count-1] > PEAK_THRESHOLD ){  // if the previous point is al
            curr_peak = __HAL_TIM_GET_COUNTER(&htim2) / 1000.0; // get current
            peak_distance = curr_peak - prev_peak;             // Compute RR-I
            if(peak_distance > MIN_RR_INTERVAL){               // if the RR-In
                num_peaks++;                                   // count it as
                if (num_peaks >= 2) {
                  computed_bpm = 300.0 / (peak_distance / 1);         // 300-me
                  transmit_bpm = 1;
                  sprintf(bpm, "%d\n\r", computed_bpm);
                  HAL_UART_Transmit(&huart1, (uint8_t *)bpm, strlen(bpm), 10);
                }
                prev_peak = curr_peak;
            }
      }
    }
}
```

Fig.8 Count peaks function.

3) Timer Period Elapsed Interrupt

This interrupt is fired when the one-minute timer period has elapsed. This marks the end of the data collection ,so inside this interrupt the sample rate timer is stopped as well as the one-minute timer. The ADC sample rate timer is stopped using `HAL_TIM_Base_Stop` , while the one minute timer is stopped using `HAL_TIM_Base_Stop_IT` since it was operating in the interrupt mode. Aslo, the two flags `transmit_adc` and `detect_peak` are reset to indicate the end of the ADC triggering process.



```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *  htim){
  if(htim->Instance == TIM2){
    if(__HAL_TIM_GET_FLAG(&htim2, TIM_FLAG_CC1) != RESET){
      HAL_TIM_Base_Stop(&htim3);      // stop ADC trigger timer
      HAL_TIM_Base_Stop_IT(&htim2);  // stop one minute timer
      transmit_adc = 0;
      detect_peak = 0;
    }
  }
}
```

Fig. 8 Timer_Period_Elapsed Interrupt callback function.

IV. Python Application

The python application is responsible for the following functionalities:

- Displaying available COM ports and selecting one to communicate with the microcontroller.
- Communicating serially with the microcontroller via the defined command set.
- Displaying ECG collected signal real time.
- Providing UI elements for the user to communicate with the microcontroller

To realize the above functionalities, the python application utilizes the `pyserial` package for establishing serial communication with the microcontroller, `matplotlib.animation` for plotting the ECG signal, and `tkinter` for providing the UI elements.

1) pyserial

Pyserial is used mainly for facilitating the serial communication. Five main functions from the function are utilized. The first one is `serial.tools.list_ports.comports()` function which lists the available COM ports. To open a port, a new serial object is created using the `serial.Serial()` constructor. Reading from an opened port is done by calling `readline()` on the serial object. Writing to the port is done by calling `write()` on the serial object. Finally, in order to close the port, the `close()`function is called on the serial object.

2) Matplotlib.animation

This package is used for plotting the ECG signal real time. Three functions are utilized from this package. The first one is `animation.FuncAnimation` which is responsible for updating the ECG plot. This function runs a background daemon which updates the ECG plot every time a fixed time interval passes. The `_start()` function is used when the second window is displayed to start the animation function. The `_stop()` function is called when the window is closed to stop the animation daemon from running. The ecg plot is updated every 1ms to account for the maximum sample rate (1000 Hz) which sends a new sample every 1 ms.

3) tkinter

The tkinter app is composed of two main windows. The first one is shown in fig.9. First, it contains a dropdown list for displaying the available COM ports. Pressing the select button opens the selected port. The window also has three buttons: Set rate, compute bpm, collect data. Pressing set rate, sends the rate command to the microcontroller with the value inside the text field. Pressing compute bpm sends the hbpm command to the microcontroller and displays the computed rate. Pressing collect data opens a new window shown in fig.10. The second window contains the ECG signal plot which is updated every time a new sample is received.

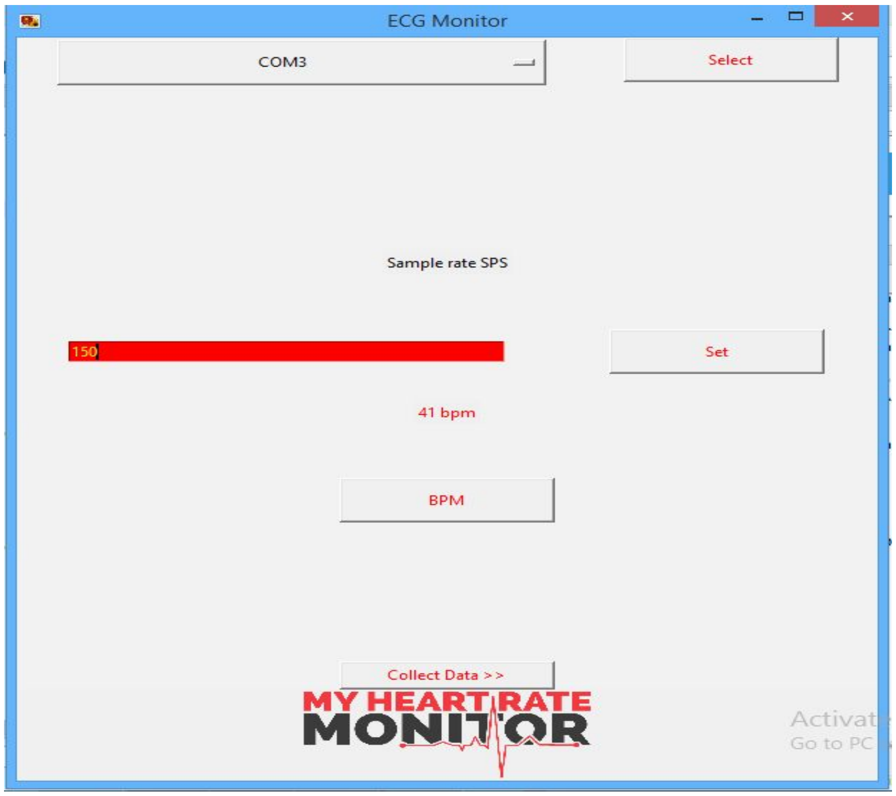


Fig.9 Main Window.

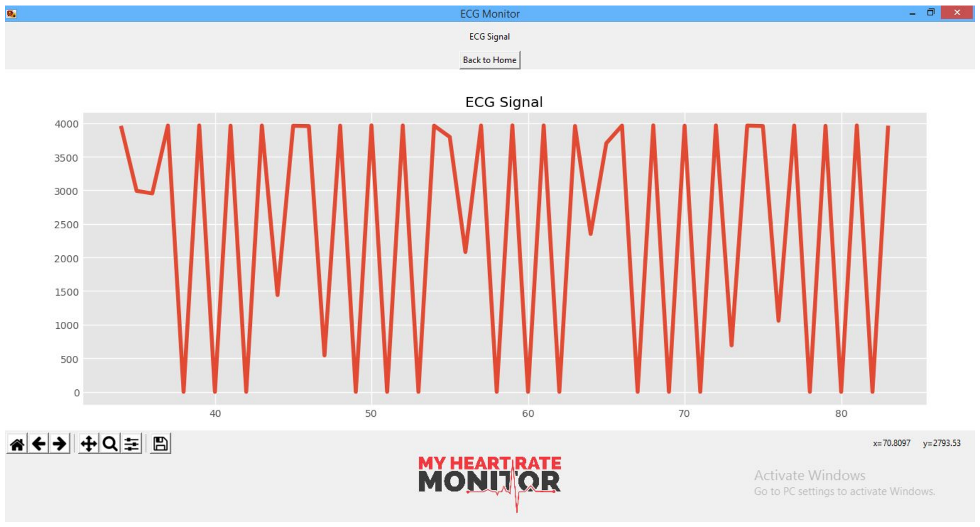


Fig.10 Second Window.