**Project Heart Monitor**

<https://github.com/BassantSamy/Heart_Monitor> Bassant Samy

The project “Heart Monitor” is one that is very closely related to real life and shows how the concepts taught in this course correlated to the outside world. Due to that I was very excited to work on this project specifically. For this project I needed the STM32F103C8 chip, the AD8232 heart monitor, an ECG wire and stickers, the USB to TTL module, and jumpers for connections. As for software I needed Keil to develop the c program that runs on the STM32 chip, CubeMx for initializations, Flash loader to download code on the microcontroller, and finally PyCharm for python program creation and debugging.

This Project consists of 2 independent components and these are the GUI and the backend c program that runs on the micro-controller.

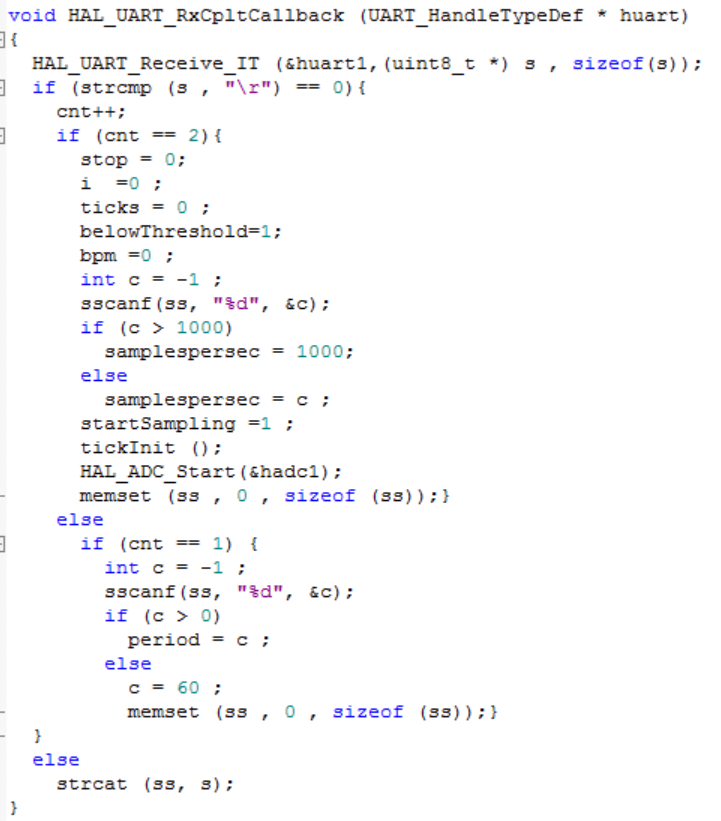
# C Program:

This part was written using Keil IDE and was the n uploaded and run on the microcontroller. This program is responsible for interfacing with the AD8232 heart monitor and the Python program as well through UART. It first captures user input specifying the sampling rate, the duration of sampling, the COM port, the baud rate, added to the start signal from the python code using UART a s a communication medium. It then utilizes the on board ADC in the STM32 chip to sample the heart monitor readings on specified intervals corresponding to the specified sampling rate. It uses these readings to calculate the BPM and sends incrementally the values and the BPM back to the python code.

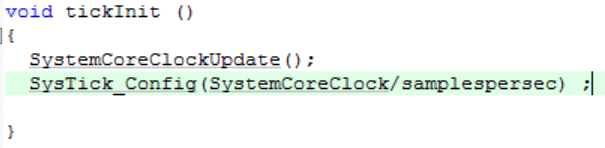
The code is entirely based on interrupts and none of the code is in the main. The code is mainly formed of the following functions:

1. void HAL\_UART\_RxCpltCallback (UART\_HandleTypeDef \* huart)
2. void tickInit ()
3. void HAL\_IncTick(void)
4. void takeSample()
5. void checkBPM(uint32\_t adcResult)
6. void minuteDone()

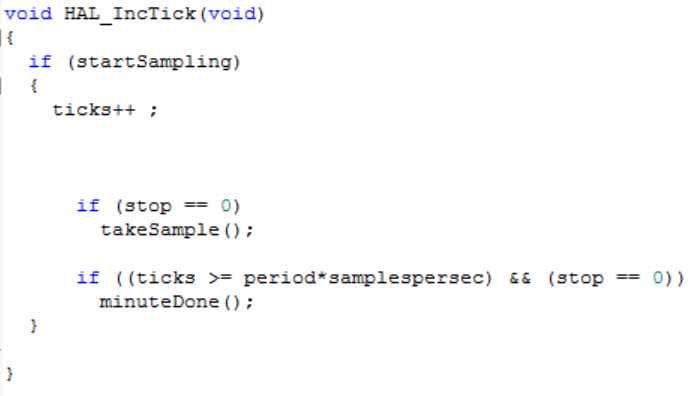
## Code Overview:



The first function is the UART receive callback function. This is the function that is able to identify the user input corresponding to the sampling time and the sampling rate. It receives from the UART digit by digit and concatenates all digits to a character array “ss” up until it receives and endl this is when it checks if this is the first input then it is he sampling time else it is the sampling rate. If it receives the sampling time it changes it into an in and stores it into the corresponding variable (period) after validating that it is not a negative number; if so it sets period to default of 60 mins. If it received the sampling rate it also validates that it is less than 1000 and greater than 0 before transforming it into an int and placing it in the suitable variable (samplespersecond). It also then signals the start of the operation/conversion by starting both the on board ADC and the systick timer. The systick timer is what I use to limit the sampling time to the one set by the user and to specify the intervals in which the on board ADC shall take a sample to set the sampling rate. Finally, I make sure the clear ss in both cases so it would not append to old values next time.

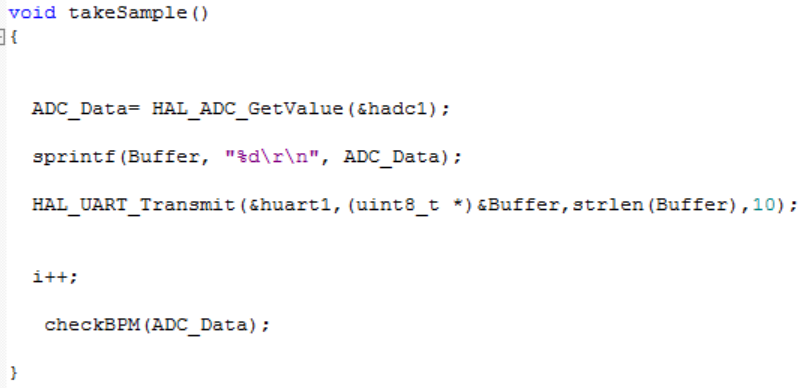


This is the second function which initializes the systick timer and sets the interval of its ticks. This interval is how long shall the ADC wait before it takes the next sample and thus is a function of the user input samplespersecond. This is done by dividing the System core clock (1 minute) by samplespersecond.

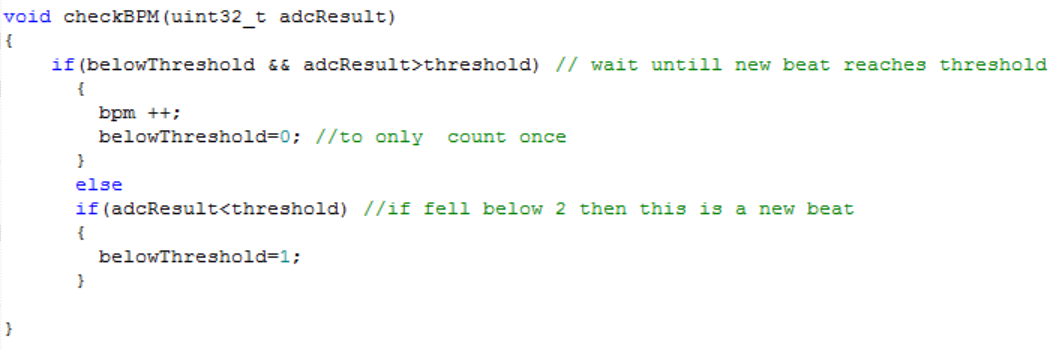


This is the function that is entered every time one tick on the systick timer has passed. It increment a variable ticks which is then monitored to estimate how much time has passed since startSampling and by that trigger events such as taking another sample and stopping the conversion after the set time has passed. Below I check if ticks have reached period \* the samplespersecond. The samplespersecond is corresponding to the time unit of the clock while the period corresponds to how many time

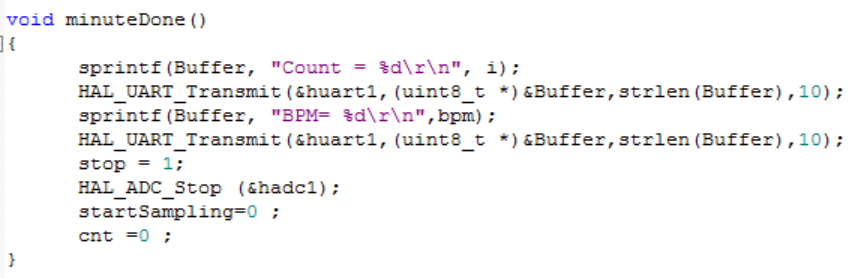
units are needed before the conversion is complete as specified by the user. If so I all the minuteDone function which sets the stop flag to 1 so it would not enter either conditions again. The above condition is held until the minute is done and that is what takes a sample from the ADC every systick timer tick.



This function is called every time a tick has passed as long as the time period set by the user is not surpassed. What it does it that it gets a value from the ADC, transmits it over the UART, increments the counter (counts how many samples are taken for debugging purposes), and calls the function checkBPM.



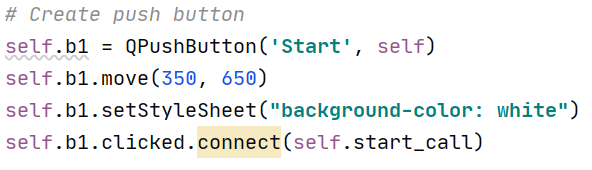
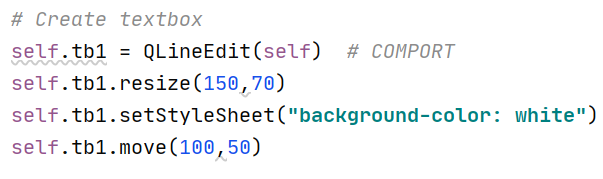
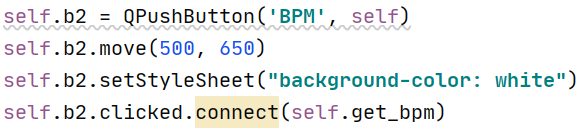
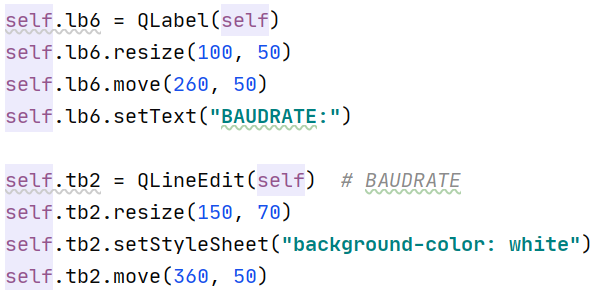
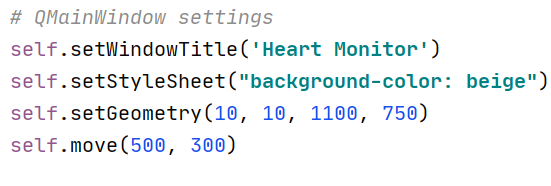
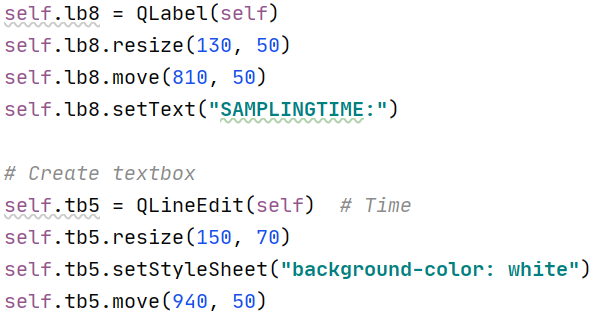
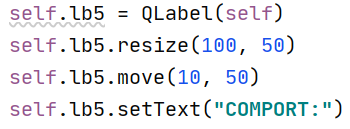
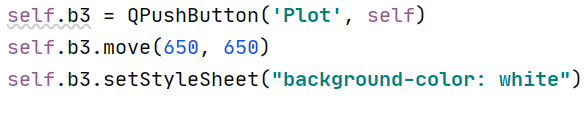
This function is called every time a sample is taken from the ADC, it based on the flag belowthreshold which is initially set to 1, it waits until the adc value surpases the threshold of 3000 meaning that this signal has reached its peak, once so it increments bpm and sets the flag to 0 so it would not count the same beat twice. It then waits until the adc value drops below the threshold signaling a different beat and counts it once passed threshold again.

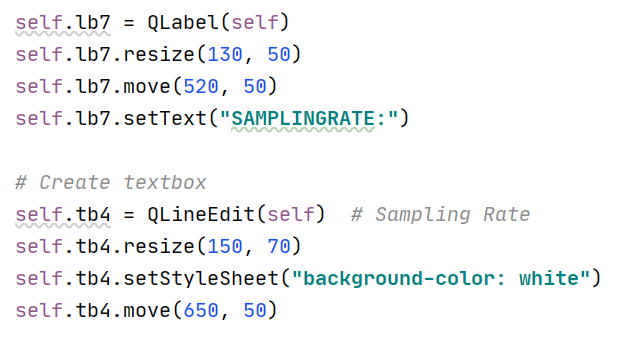


This function is called after the time set by the user has passed, it then accordingly sends the samples count and the BPM over the UART. It then sets the stop bit to one to stop taking samples, stops the ADC, start sampling to 0 to stop incrementing the systicks, and sets cnt to 0 to 0 that counts the number of inputs received (to enable replays).

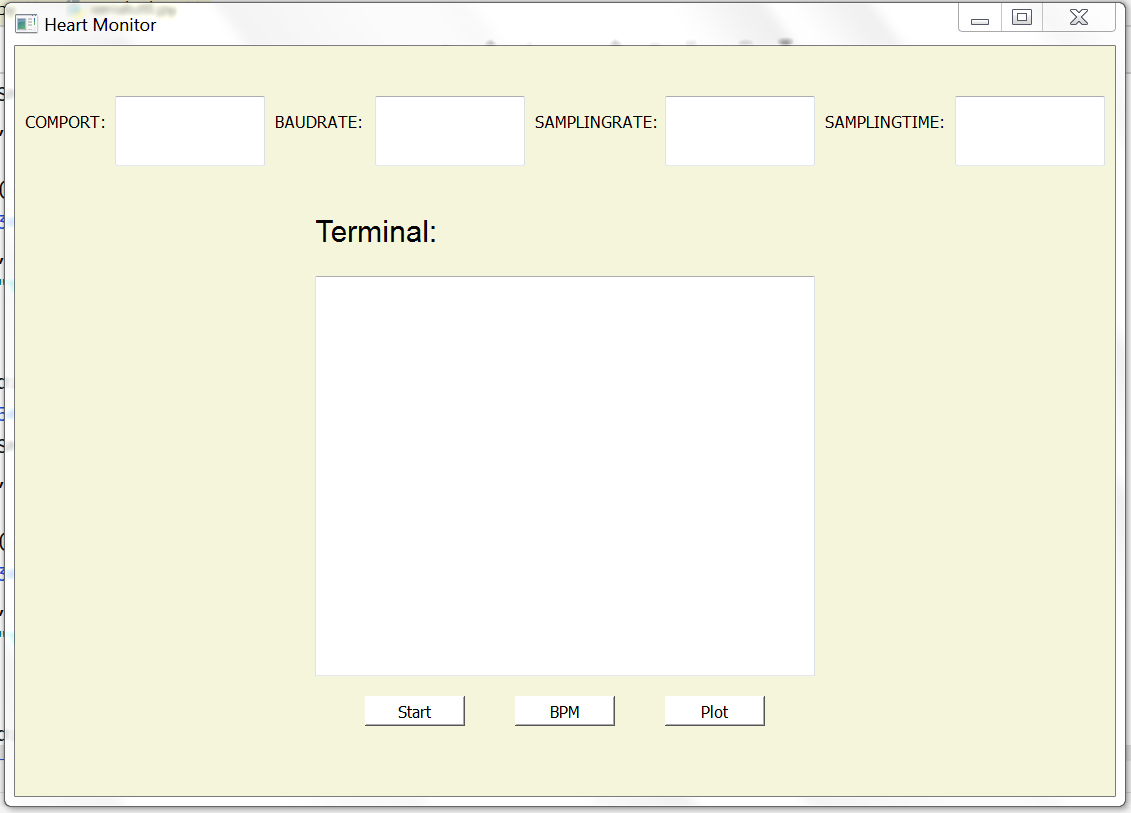
# Python Code:

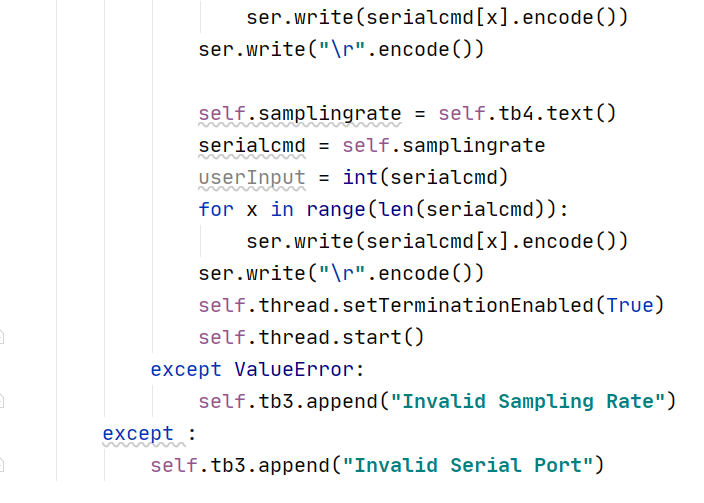
This code is what interfaces with the user to get the sampling rate, the sampling time, the baud rate, and the COM port. It also allows the user to view the samples taken from the ADC and to request a plot of the samples over the time period.

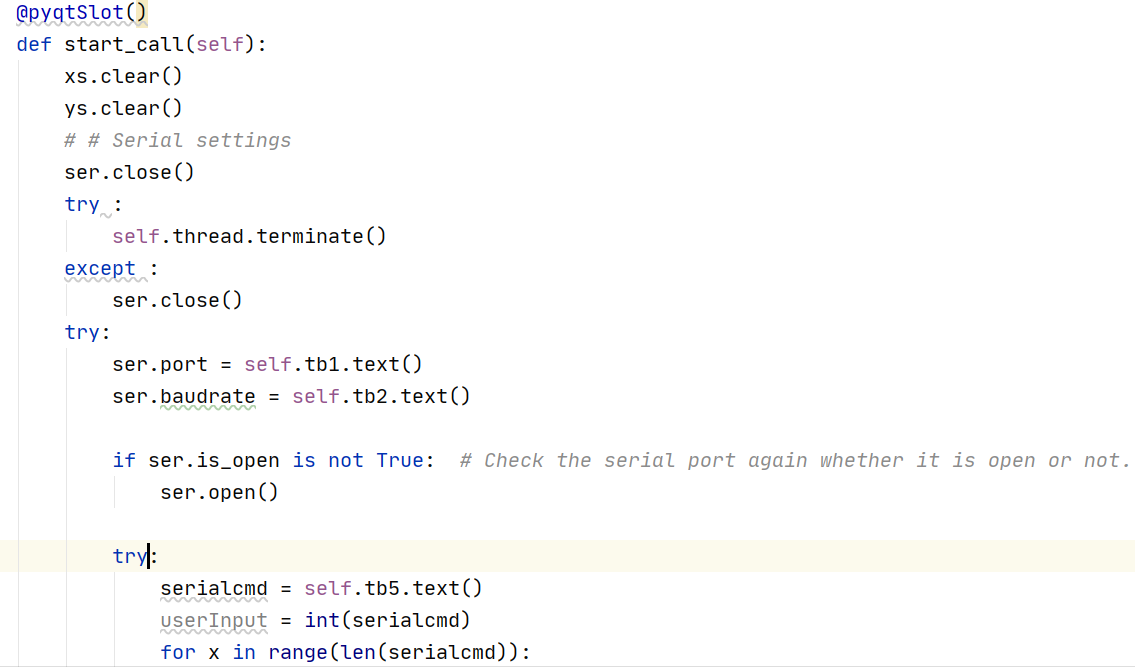




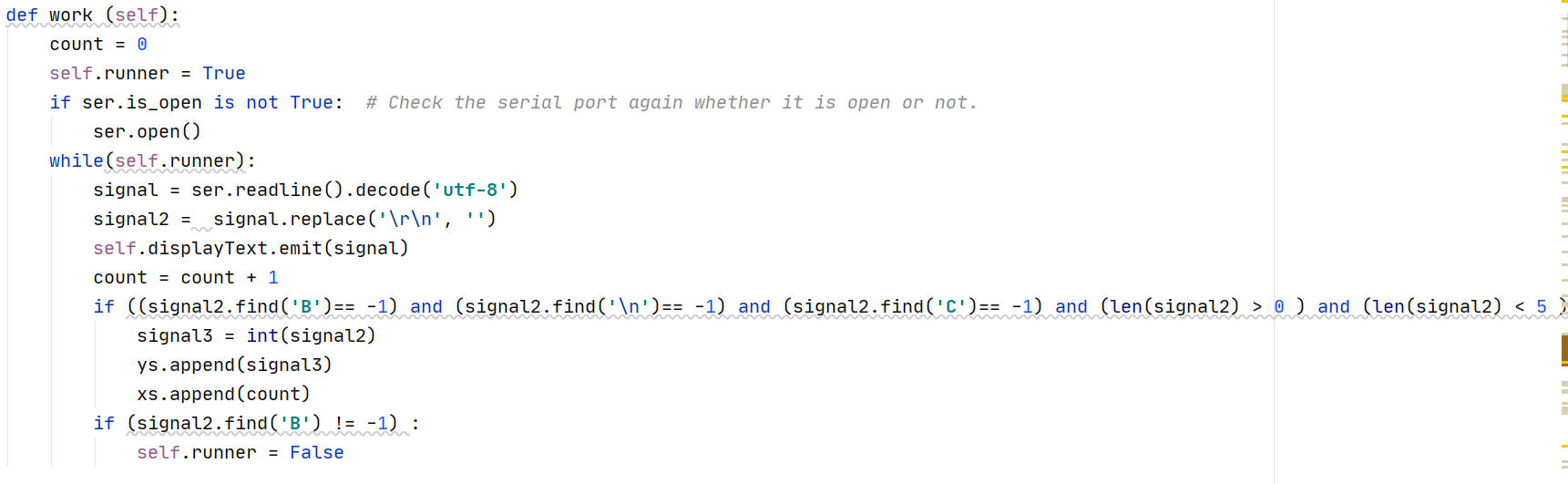
In the above code I set the GUI interface elements such as textboxes, buttons, and labels. To produce a window as such:



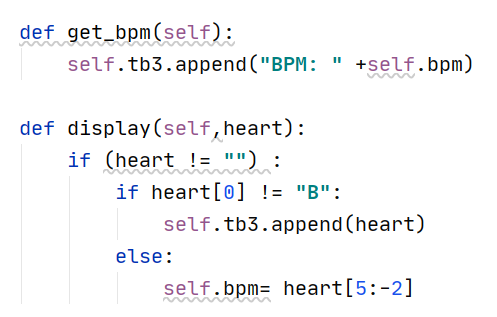




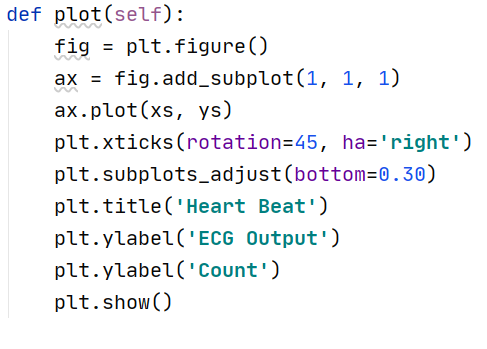
In the above code snippet I show the function that is called when the start button is clicked. It starts by sending the sampling time and the sampling rate digit by digit terminated by a \r to correspond with my keil protocol. After that I start a thread that receives the ECG signal. Please note that all my input I validated, and if not correct I prompt the user with the error where he is then enabled to re-enter the values.



This is the function that runs on a separate thread and receives data from the UARRT. If it does not receive rubbish input, which is probable with high sampling rates, it places them in a list that is then plotted if the user requests a plot. If it receives “B” , ie the last line corresponding to the BPM, it stops the thread and closes the port.

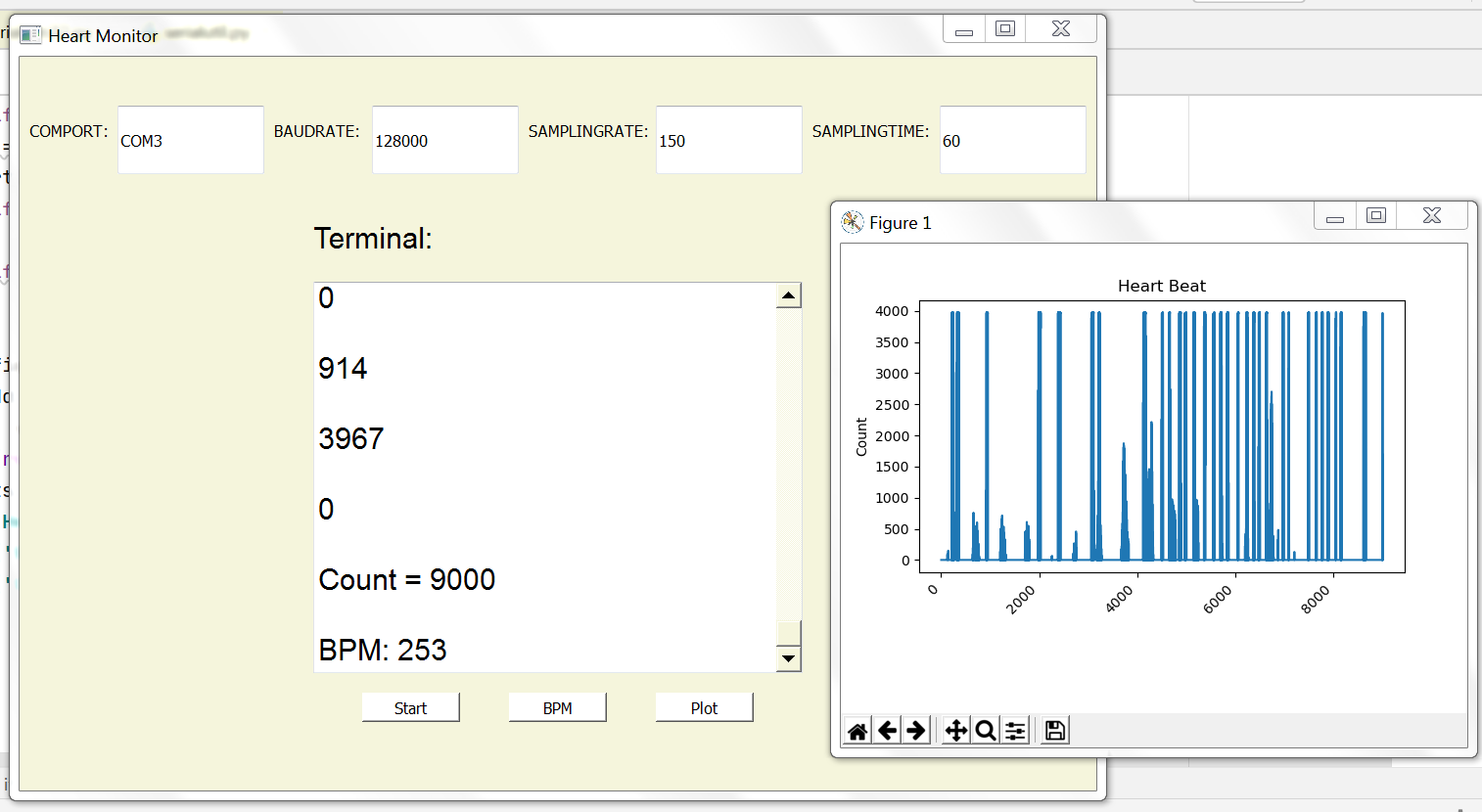


This is what gets called if the user presses the bpm button. It just displays the bpm reading that was achieved when the display function received a line starting with B.

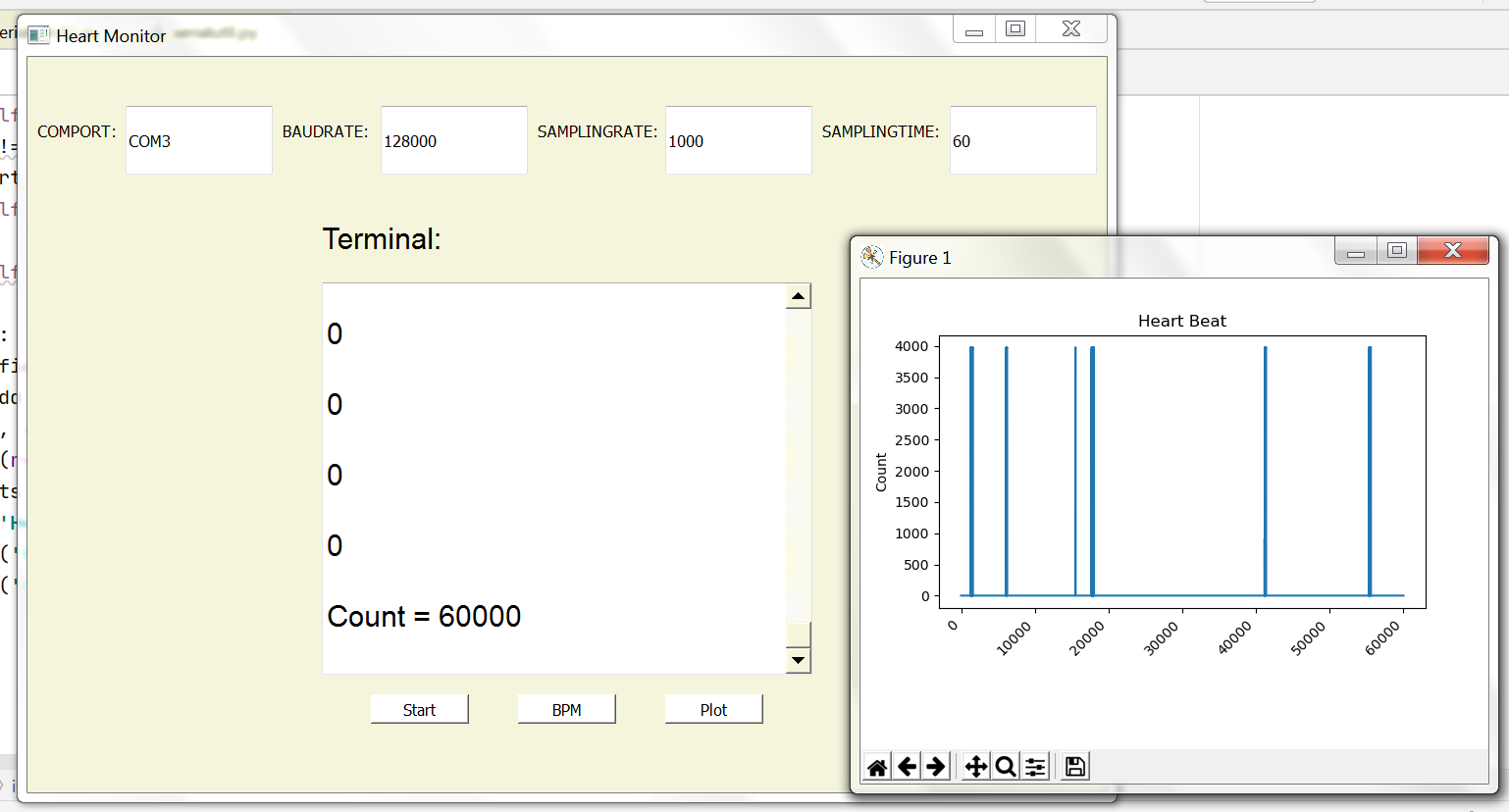


Finally this is what is called if the user presses the plot button. It just calls the plot function under matplotlib with the lists that were created when receiving the UART values as described previously.

**Results:**



The count in the above image is 9000 which is 60 \* 150 (sampling time \* sampling rate). Also please note that this is not a real graph of a heart signal this signal is produced by simply tapping over the ECG sensors.



Here I set the sampling rate to the maximum over the 60 mins which produces 60,000 samples.

# Connections:

This section describes the required connections for this project. First, I connected the AD8232 with Grd and 3.3v Vcc as required (theses are supplied by the USB to TTL module). Then I connected the output pin of the AD8232 to the AD0 pin of the microcontroller which is the input to the first on-board ADC. I also needed to connect the Tx, Rx, Vcc, and Grd of the board with these of the TTL module. Finally, I connected the ECG wire coming out of the AD8232 to the specified spots for acquiring heart rate which include the wrists, the chest, and the right thigh.

