**Embedded Systems**

**Heart Monitor Project**

**Maha Moussa - 900161246**

**Project Description:**

I developed a heart monitor that samples the ECG signal obtained from the ECG sensor. The Sensor transmits the ECG reading to the blue pill, which does some processing on the received signal. This processing includes: converting the analog signal into a digital signal. The digital signal is then used for further analysis including: calculating the BPM rate and transmitting 1 minute worth of digital samples to the PC through UART. On the PC, the digital signal can be plotted against time.

The application allows the user to choose the Com Port, baud rate and sampling rate of preference.

**System Components:**

USB to TTL

MCU

ECG Sensor

ADC

**ECG Sensor:**

The ECG Sensor used is the AD8232. It will be used to collect the ECG signal. The ECG will be driving the ADC in the microcontroller.

**MCU**:

The microcontroller used is the STM blue pill (STM32F103). The microcontroller has 2 built in ADC. The ADC will be used to convert the ECG signal into digital signal and sample the signal according to the sampling rate input from the user.

**USB to TTL:**

The MCU will communicate its output to the PC’s serial link the using the UART.

**Main functions and their description:**

**C Code:**

*void HAL\_ADC\_ConvCpltCallback(ADC\_HandleTypeDef\* hadc){*

*adc\_val\_prev = adc\_val;*

*adc\_val = HAL\_ADC\_GetValue(&hadc1);*

*}*

* Once the adc completed the conversion of a sample, it causes an interrupt during which the program reads the converted value

*void HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef\* huart){*

*sscanf(rec\_val, "%d", &input\_rate);*

*if(input\_rate>0x9){*

*GenerateRate();*

*flag = 1;*

*HAL\_UART\_AbortReceive\_IT(&huart1);*

*}*

*else{*

*HAL\_UART\_Receive\_IT(&huart1,(uint8\_t\*) rec\_val, sizeof(rec\_val));*

*}*

*}*

* the sampling rate set by the user is sent over UART. When Receive occurs and interrupt happens. The interrupt calls another function to set the time between two consecutive samples and a flag is set to indicate the start of taking samples.

*void GenerateRate(){*

*if(input\_rate>1000)*

*input\_rate=1000;*

*gen\_rate = (1000/input\_rate);*

*sample\_ticks=0;*

*}*

* this Function sets a counter, gen\_ rate, that determines the number of ticks between each 2- consecutive samples

*SystemCoreClockUpdate();*

*returnCode= SysTick\_Config(SystemCoreClock/1000);*

* these two lines of code were used to configure the systick interrupt to happen every 1msec. This way it would be easier to calculate the number of ticks between 2 consecutive samples and the number of ticks required to collect on minute worth of data.

*void SysTick\_Handler(void)*

*{*

*if(min\_ticks==60000 ){*

*if(tr\_flag==0x0){*

*char s[]= "BPM value is\n";*

*HAL\_UART\_Transmit(&huart1, (uint8\_t \*)s, strlen(s), HAL\_MAX\_DELAY);*

*sprintf(Bpm, "%d\r\n", sum\_cnt);*

*HAL\_UART\_Transmit(&huart1, (uint8\_t \*)Bpm, strlen(Bpm), HAL\_MAX\_DELAY);*

*tr\_flag =0x1;*

*}*

*flag=0;*

*HAL\_ADC\_Stop\_IT(&hadc1);*

*}*

*else if (flag == 1){*

*min\_ticks = min\_ticks+1;*

*if(sample\_ticks==gen\_rate-1){*

*sprintf(msg, "%d\r\n", adc\_val);*

*HAL\_UART\_Transmit(&huart1, (uint8\_t \*)msg, strlen(msg), HAL\_MAX\_DELAY);*

*sample\_ticks = 0;*

*if(adc\_val> 0xBB8){*

*bpmFlag=1;*

*}*

*if( (adc\_val< 0x1F4) && (bpmFlag ==1) ){*

*bpmFlag=0;*

*sum\_cnt = sum\_cnt+1;*

*}*

*}*

*else {*

*sample\_ticks = sample\_ticks+1;*

*}*

*}*

*HAL\_IncTick();*

*}*

* Inside the IRQ handler of the systick the logic for sampling and collecting data is written. The function first checks whether a minute has passed or not. if not, it checks if the time between 2 consecutive samples has passed. If that is the case, a sample is taken and is transmitted through the UART to the PC.
* Meanwhile, the counter for the BPM gets incremented, whenever the adc\_value goes from low to high. The threshold marking a high sample is 3000 and the threshold for a low sample is 500.
* After a minute has passed the BPM is transmitted to the PC through UART

**Python Code:**

*ser = serial.Serial(port=in\_port, baudrate=in\_baudr, bytesize=serial.EIGHTBITS, parity= serial.PARITY\_NONE, timeout=2)*

* *Used to open the Serial Port*

*ports = serial.tools.list\_ports.comports()  
for port, desc, hwid in sorted(ports):  
 print(****" {}: {} "****.format(port, desc))  
in\_port = input(****"Please Choose one of the ports listed above:****\n****"****)*

* This function lists all the ports to which a device is connected to, so that the user can choose a Com port

*def animate(i, ys, sampler):  
 temp\_arr\_2, stop = vals(sampler)  
 if stop == 1:  
 read = ser.readline()  
 read2 = read.decode(****'UTF-8'****)  
 read2 = read2.strip(****"****\n****"****)  
 if (len(read2) > 0 and len(read2) < 5):  
 read3 = int(read2)  
 print(read3)  
 ani.event\_source.stop()  
 for x in temp\_arr\_2:  
 ys.append(x)  
 ys = ys[-x\_len:]  
 line2.set\_ydata(ys)  
 return line2*

* This Is a function that Is continuously called. It reads the samples received through the serial port and initializes an arry with it. This array is then used to plot the graph.

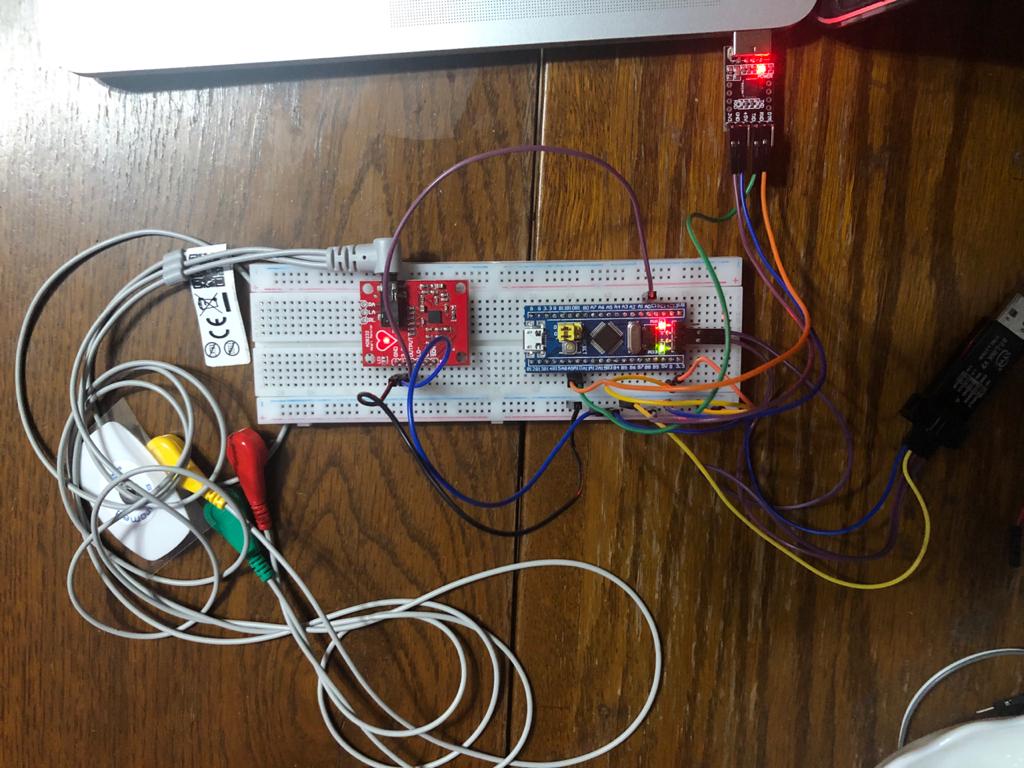
Fig = plt.figure()  
*ax = fig.add\_subplot(1, 1, 1)  
xs = list(range(0, 200))  
ys = [0] \* x\_len  
ax.set\_ylim(y\_range)  
line2, = ax.plot(xs, ys)  
ani = animation.FuncAnimation(fig, animate, fargs=(ys,in\_sampler ), interval=1)  
plt.show()*

* These are the lines that actually plot the graph

def vals(sampler):  
 temp\_arr = []  
 stp\_f = 0  
 i=0  
 sampler= int(sampler)  
 while(i< sampler\*0.2):  
 read = ser.readline()  
 if (read != b''):  
 read2 = read.decode(**'UTF-8'**)  
 read2 = read2.strip(**"**\n**"**)  
 read2 = read2.strip(**"**\r**"**)  
 if(read2.find(**"BPM"**)!=-1):  
 stp\_f = 1  
 print(read2)  
 return temp\_arr, stp\_f  
 if(len(read2)>0 and len(read2)<5):  
 read3 = int(read2)  
 temp\_arr.append(read3)  
 i = i+1  
 return temp\_arr, stp\_f

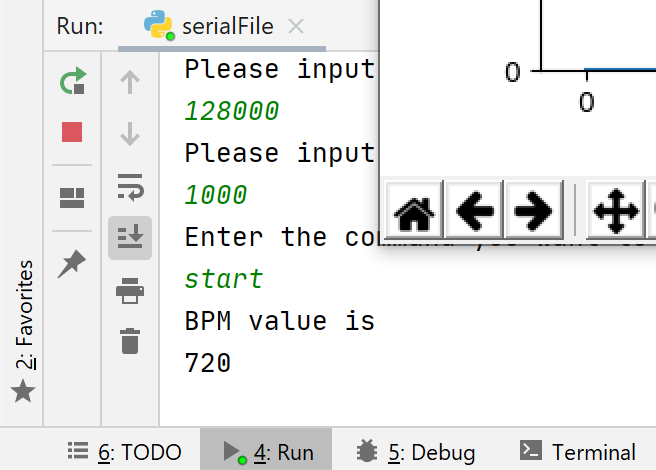
* Plotting an animated graph with a high sampling rate was so slow. Therefore, This function was used to collect a number of samples that is equal to 20% of the sampling rate and then plot these values once. This reduced the overhead on the plotting function

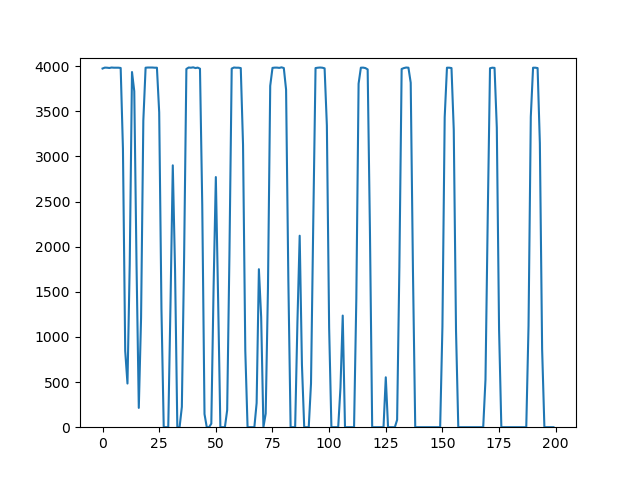
**Connections:**

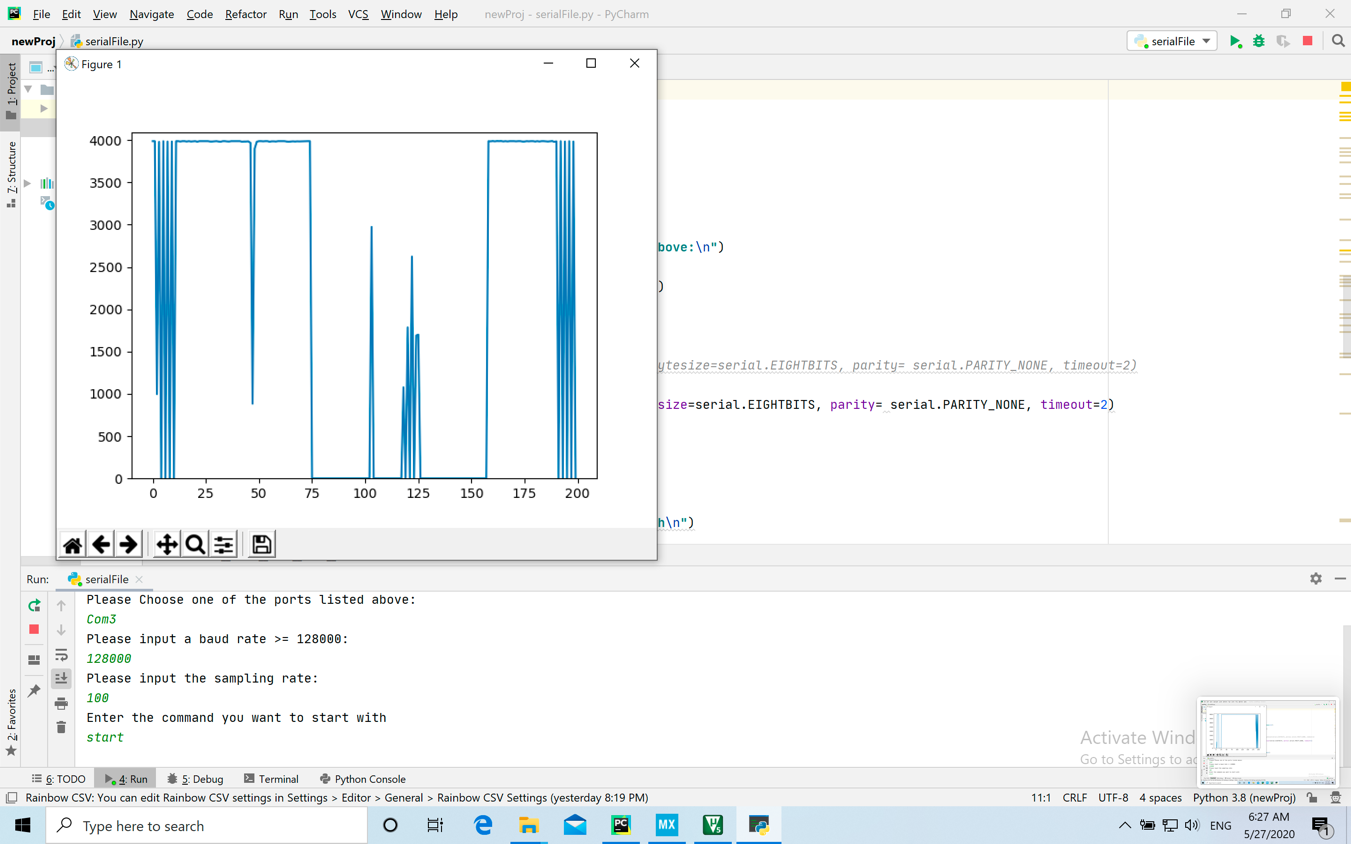
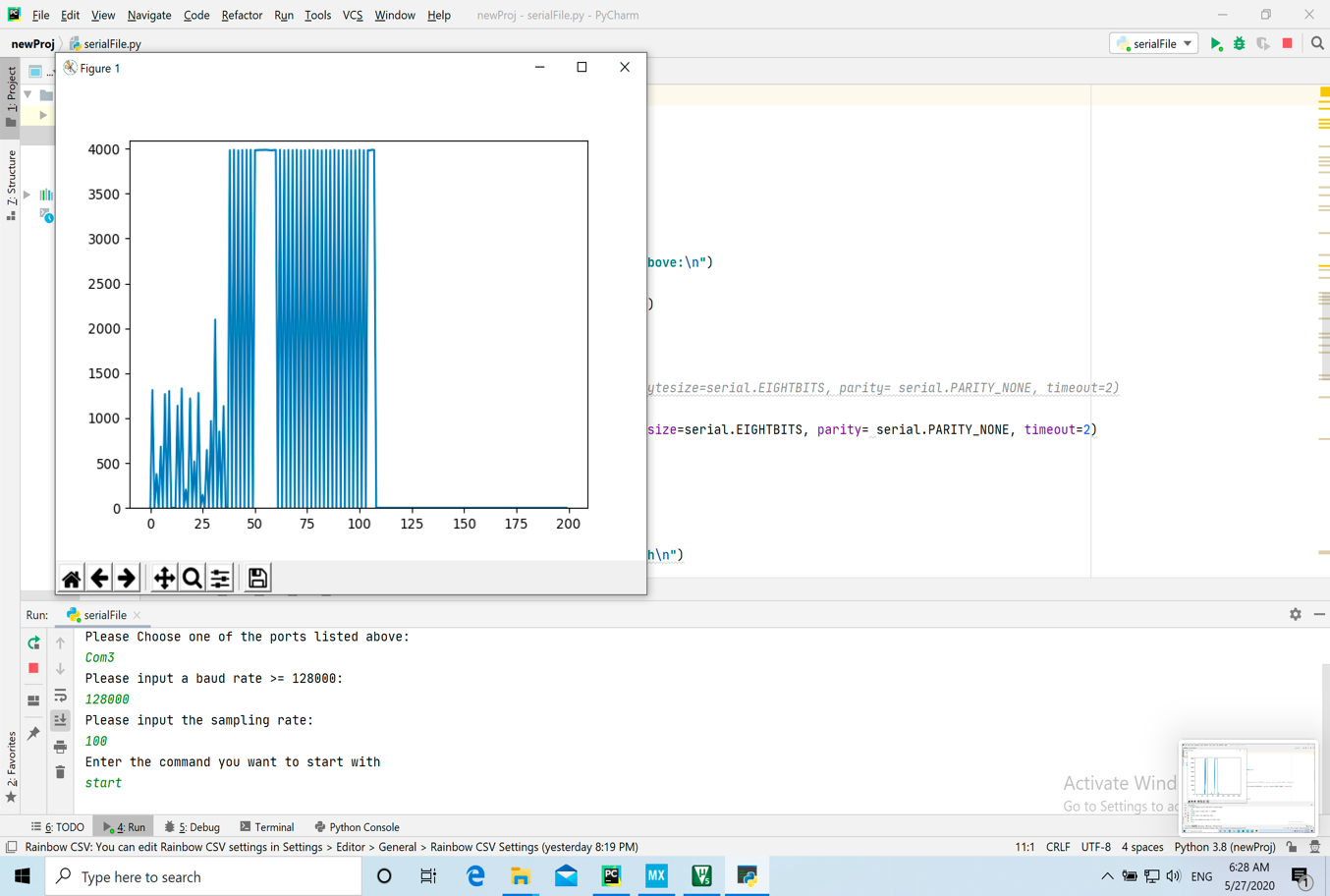


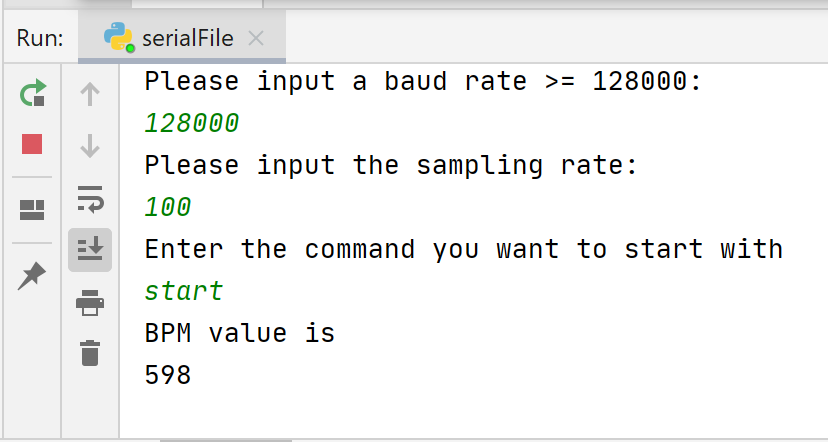
**Results:**

For 1000 sampling rate:





 For 100 sampling rate: 



**Difficulties faced and what I Learnt:**

* Handling multiple interrupts at the same time. When 3 interrupts were activated at the same time the program was so slow
* Had to use a high baud rate, Because a low one with the amount of data being sent wasn’t very efficient. The time taken to transmit was longer than the time between two interrupts. Therefore, other necessary parts of the code weren’t executed
* Not very familiar with python. Unnoticeable mistakes because of being used to the C++ syntax occurred and caused the code to crash

**Assumptions:**

* The sampling rate would be in the range of 100 to 1000
* The user would type Start and then the graph and the BPM would be automatically shown